

Эффективная работа™

Java 2

Джуди Бишоп

bhv
Addison Wesley
ПИТЕР

СЕРИЯ

Эффективная работа™

ПИТЕР®

Judith Bichop
Computer Science Department
University of Pretoria

Java Gently

3RD EDITION



Addison-Wesley

An imprint of **Pearson Education**

Harlow, England • London • New York • Reading, Massachusetts • San Francisco
Toronto • Don Mills, Ontario • Sydney • Tokyo • Singapore • Hong Kong • Seoul
Taipei • Cape Town • Madrid • Mexico City • Amsterdam • Munich • Paris • Milan

Джуди Бишоп

Эффективная работа™

Java 2

2002

Санкт-Петербург
Москва ▪ Киев ▪ Харьков ▪ Минск

bhv®

ПИТЕР®

Джуди Бишоп

Эффективная работа: Java 2

Перевод с английского под редакцией А. Падалки

Редакторы	*Т. Мартынюк, И. Карпышенко, Е. Насырова*
Художник	*Н. Биржаков*
Корректор	*И. Карпышенко*
Технический редактор	*З. Лобач*

ББК 32.973-018.1я7
УДК 681.3.06(075)

Бишоп Д.

Б67 Эффективная работа: Java 2. — СПб.: Питер; К.: Издательская группа BHV, 2002. — 592 с.: ил.

ISBN 966-552-107-1
ISBN 5-318-00741-4

Предлагаемое издание посвящено вопросам программирования на языке Java. Изложены общие принципы программирования, а также базовые сведения о языке Java 2, его основных библиотеках и утилитах. Рассмотрены такие важные темы, как структуры данных, разработка объектно-ориентированного программного обеспечения, структурное программирование, создание графического пользовательского интерфейса, организация обработки событий, реализация сетевых функций, а также одновременное использование в программе нескольких подпроцессов. В книге содержится подборка протестированных примеров, в которых рассматриваются пути решения реальных задач. Благодаря детальному описанию алгоритма легко понять код каждой программы. Каждую главу завершает перечень вопросов и задач, которые позволят проверить и закрепить полученные знания.

Original English language Edition Copyright © Pearson Education Limited, 2001
© Издательская группа BHV, Киев, 2002
© Издательский дом «Питер», 2002

Права на издание получены по соглашению с Addison-Wesley Longman.

Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственность за возможные ошибки, связанные с использованием книги.

ISBN 966-552-107-1
ISBN 5-318-00741-4
ISBN 0-201-71050-1 (англ.)

ЗАО «Питер Бук». 196105, Санкт-Петербург, ул. Благодатная, д. 67в.
Лицензия ИД № 01940 от 05.06.00.
ООО «Издательская группа BHV».
Свидетельство о занесении в Государственный реестр серия ДК № 175 от 13.09.2000.
Налоговая льгота – общероссийский классификатор продукции ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 07.02.02. Формат 70×100$^{1}/_{16}$. Усл. п. л. 79,98. Тираж 5000 экз. Заказ № 2760.
Отпечатано с готовых диапозитивов в ФГУП «Печатный двор» им. А. М. Горького
Министерства РФ по делам печати, телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.

Посвящается моему дорогому Найгелю, чья страсть к науке направляет и вдохновляет меня во всем, что я делаю; с любовью и восхищением

Предисловие к третьему изданию

Java Gently — это книга, которая представляет язык Java как первый изучаемый язык программирования. Она предназначена для студентов, а также для тех, кого серьезно заинтересовали возможности их компьютеров и Internet. Цель данной книги состоит в том, чтобы научить читателя не только писать работающие программы, но и использовать при их создании характерный стиль Java. В процессе изложения материала освещены все фундаментальные структуры языка Java 2, а также большинство библиотек и вспомогательных средств, которые составляют его ядро. Что касается непосредственно технологий программирования, то в книге рассматриваются такие темы: объектно-ориентированное программирование, разработка программного обеспечения, структурное программирование, создание графического пользовательского интерфейса, программирование, управляемое событиями, использование в программах нескольких подпроцессов, работа в сети и введение в структуры данных.

Философия преподавания Java

Преподавание Java в качестве первого языка программирования не является большой редкостью, что учитывается при составлении учебных программ для университетов, институтов и даже старших классов школ. Преподаватели осознали, что Java имеет ряд уникальных характеристик, которые делают его языком, подходящим для подготовки студентов к изучению других курсов и языков. Кроме того, язык Java актуален и пригоден к немедленному практическому применению, и это способствует тому, что студенты хотят его изучать! Java занимателен, интересен, и поэтому вовсе не трудно освоить его основополагающие принципы. Перечислим его отличительные признаки.

1. Перечень возможных конструкций Java невелик, и язык охватывает ограниченное количество характеристик (функций), которые можно успешно применять совместно.
2. Java — это объектно-ориентированный язык, что дает возможность программам отражать реальный мир и уменьшает количество повторений и ошибок.
3. Java предоставляет возможность создавать полноценный графический пользовательский интерфейс (GUI) и использовать средства мультимедиа (звук, изображения и анимацию), а также способствует преобразованию GUI-программ в апплеты, предназначенные для Web-броузеров.
4. Java имеет встроенные характеристики, позволяющие создавать программы с несколькими подпроцессами, а также обладает многочисленными средствами для программирования на сетевом уровне.

Учитывая сказанное, можно предложить два возможных подхода для представления Java как первого языка программирования. Первый, вероятно, наиболее привлекательный — начать сразу с GUI, мультимедиа, апплетов и всех атрибутов сети, демонстрируя тем самым студентам, насколько современен и разнообразен Java. Второй подход заключается в изучении основных концепций языка, после чего можно будет перейти к его новым аспектам.

В данной книге были применены оба подхода. Каждый из них имеет как преимущества, так и недостатки. Начать с апплетов и GUI — перспектива, конечно, захватывающая, но понимание этих тем требует такого объема соответствующих знаний об объектах, что изучение внешней оболочки программ и запоминание того, что нужно сделать для придания им презентабельного вида, не поможет понять реально происходящих процессов. С другой стороны, плавное продвижение в освоении материала с изучением всех основ без наглядных примеров может показаться скучным и отпугивающим занятием.

Подход, выбранный для книги Java Gently

Подход, который был мною разработан, предусматривает продвижение одновременно в трех направлениях и основывается на трех главных козырях Java: сетевом ориентировании, сравнительно малом количестве конструкций и стимулировании повторного использования созданных компонентов. Поэтому мы начнем с программ, которые показывают, как используется Web, но затем сразу же перейдем к освещению основополагающих принципов. Будут рассмотрены не только пакеты, используемые в предыдущем издании (библиотеки классов или интерфейсов прикладных программ), но и программы, написанные специально для данной книги. Это позволит студентам начать освоение языка Java с изучения довольно сложного процесса ввода/вывода и получить представление о принципах объектно-ориентированного программирования, причем материал будет изложен в полном объеме с рассмотрением базисных концепций. В результате студенты, ознакомившись с такими понятиями, как классы, объекты, экземпляры и наследование, научившись работать с новыми элементами языка, смогут быстрее воспринимать всю дальнейшую информацию (в частности, касающуюся апплетов, мультимедиа, графиков и т.д.).

Однако данный подход не может заменить собой все наработанные методы преподавания, поэтому в книге используются следующие уже оправдавшие себя приемы:

- использование диаграмм везде, где только можно;
- объяснение на примерах;
- вопросы для самоконтроля с несколькими вариантами ответа на них, а также задачи по рассмотренным темам.

В книге используются три вида диаграмм.

- **Диаграммы моделирования**, или **схемы** — демонстрируют структуру программы на уровне классов и объектов и одновременно включают в себя базисные компоненты, такие как методы и переменные, а также показывают связи между ними.
- **Описания** — помогают понять синтаксис и семантику каждой новой конструкции Java; представлены в форме, которая отражает применение этих конструкций на практике.
- **Схемы алгоритмов** — объясняют последовательность действий, выполняемых в небольших программах, а также служат для демонстрации новых концепций, таких как обработка исключительных ситуаций и использование нескольких подпроцессов.

В третье издание книги внесен ряд изменений, касающихся диаграмм. Во-первых, обозначения для диаграмм классов упрощены и приведены в соответствие с унифицированным языком моделирования UML (Unified Modeling Language), хотя сам UML и не пригоден для программирования на этом уровне. Кроме того, были расширены описания, и теперь они включают семантику. Книга снабжена справочным приложением, содержащим, в частности, названия всех описаний, по которым можно легко найти нужную конструкцию языка или API. Наконец, многие алгоритмы, недостаточно четко объяснявшие выполняемые действия, были переписаны.

Примеры

В книге имеется более 80 работоспособных примеров, содержащих вводимые данные и полученные результаты, причем в третьем издании 30% примеров — новые. Каждый из таких примеров написан в наилучшем для Java стиле и не только демонстрирует работу рассмотренных функций и свойств, но и решает реальные задачи. Особое внимание было уделено тому, чтобы примеры действительно имели "привкус" реальных проблем, что приучает начинающего программиста понимать нужды и возможности рядового пользователя, а значит, писать программы, работать с которыми было бы легко и удобно. Некоторые программы встречаются в книге неоднократно, каждый раз обновляясь рядом интересных возможностей языка Java. Вот их перечень:

- "Лавка древностей";
- "Доставка пиццы";
- "Разработчик флагов";
- "Конвертатор валют".

Вводимые в программу структуры или характеристики сразу же становятся ее частью и в дальнейшем повторяются. По мере рассмотрения новых тем примеры становятся длиннее и, естественно, интересней.

Изучение программирования с помощью языка Java

В настоящее время на первом курсе института уже недостаточно простого изучения основ языка программирования — больше внимание следует уделять его практическому применению. Помимо объяснения синтаксиса и конструкций и без того хорошо оформленной программы, учебник программирования этого столетия должен включать в себя методики решения различных задач.

Java Gently — действительно современная книга. В ней рассматриваются более 80-ти задач, каждую из которых можно решить несколькими способами. После обсуждения возможных вариантов решения выбирается наиболее удачный из них. Затем разрабатывается алгоритм. Чтобы упростить понимание решения, алгоритмы сопровождаются структурными диаграммами, а наиболее интересные способы обсуждаются и на их основе создаются специальные утилиты. Надлежащее внимание в книге уделено всем важным составляющим программирования, и в частности:

- структурному программированию;
- объектно-ориентированному программированию;
- конструкции класса;
- абстракции;
- повторному использованию программного обеспечения;
- обобщенности.

Более того, примеры подобраны таким образом, чтобы показать, как эти технологии могут способствовать повышению корректности и эффективности программирования. Так, имеют место дискуссии о возможной конструкции класса, о взаимоотношениях между классами и массивами, о том, целесообразно ли использовать в циклах объекты-исключения, а также о различных способах реализации структур данных.

Порядок изложения материала

Последовательность рассмотрения операторов и операций следующая: вывод результатов работы, оператор присваивания, цикл `for`, вызов (метода), ввод данных, структура выбора `if`, исключительные ситуации, циклы `while` и `do-while`, структура с множественным выбором `switch`. Такой порядок не совпадает с порядком изложения материала в справочном руководстве Java или в книгах, предназначенных для тех, кто знаком с программированием. Однако это уже оправдавшая себя на практике методика преподавания основ программирования, следуя которой студенты погружаются все глубже в данную науку, причем процесс обучения проходит без задержек и остановок, т.е. *плавно*, или даже *мягко* (в данном случае это будет наиболее корректным вариантом перевода английского слова *gently*).

Почти во всех главах содержатся сведения, касающиеся классов, данных и управления, а кроме того рассматриваются некоторые особенности непосредственно языка Java. Мой опыт показал, что подача материала именно в таком интегрированном виде значительно облегчает понимание. Студенты достаточно быстро продвигаются вперед и спустя относительно короткое время уже могут решать довольно интересные задачи. Причем это происходит не в ущерб освоению основ программирования и языка Java. Ключевыми моментами в порядке изложения всего материала являются:

- возможность быстрого перехода к реально существующим проблемам программирования;
- использование реалистичных примеров (где с самого начала применяются более чем один класс и более чем два объекта), демонстрирующих силу объектно-ориентированного программирования;
- рассмотрение в первых главах исключительных ситуаций (объектов-исключений) и подача их как естественного способа контролирования потоков между классами;
- уделение должного внимания не только трудным темам (таким, как приведение к типу или клонирование), но и простым характеристикам Java (в частности, структуре класса);
- рассмотрение хеш-таблиц одновременно с массивами в целях демонстрации их эффективности в обеспечении индексации без использования целых чисел;
- присутствие трех специально созданных пакетов ввода/вывода, которые применяются начиная с 4-й главы и подробно рассматриваются в 7-й и 14-й главах;
- достаточно подробный анализ пакета `awt` (abstract windowing toolkit — пакет абстрактных классов для реализации оконного интерфейса), сопровождаемый множеством примеров, в том числе с использованием пакета `Swing`;
- представление апплетов с помощью специальных программ просмотра и броузеров, объяснение различий между апплетами и приложениями при рассмотрении новой, но идентичной темы;

- активное использование в примерах подпроцессов, в результате чего было разработано несколько интересных приложений;
- программирование на уровне сокетов, демонстрирующее простые системы клиент/сервер или одноранговые системы (например, программа-сервер "Chatter"), а также описание принципов использования RMI, показывающего возможности Java в сетевой среде;
- подключение к базам данных посредством моста ODBC-JDBC (открытое подключение к базам данных — подключение к базам данных с помощью API-интерфейса), который позволяет работать с системой баз данных; такой доступ для лучшего его понимания, изучался поэтапно;
- рассмотрение некоторых компонентов JFC (Java Foundation Collection — коллекция классов API Java), относящихся к алгоритмам и структурам данных, в частности алгоритмов сортировки и поиска для работы со стеками, очередями, списками и битовыми множествами.

Средства, способствующие лучшему освоению материала, и информация на Web-узле

Книга *Java Gently* содержит:

- более 80-ти полностью завершенных, работоспособных примеров, а также 11 программ, которые рассматриваются под отдельной рубрикой "Практикум";
- выводы по окончании каждой главы;
- вопросы для самоконтроля, задаваемые в конце каждой главы, с несколькими вариантами ответов (правильные ответы даются в Приложении Г);
- более 100 задач;
- ссылки на Web-узел поддержки;
- специальные классы `Display`, `Stream` и `Graph`, предназначенные для расширения возможностей ввода-вывода в Java (в частности, использование класса `Display` позволяет на ранних этапах создавать GUI-программы);
- более 70-ти алгоритмов и схем моделирования программ;
- уникальный способ представления синтаксиса и семантики Java для начинающих программистов.

Все примеры были проверены и протестированы. Они доступны на Web-узле

`www.booksites.net/bishop`

который активно поддерживается и обновляется. Кроме того, на нем имеются:

- Web-страницы, которые представляют книгу;
- все приведенные в настоящем издании примеры (их можно загрузить как по главам, так и все сразу);
- часто задаваемые вопросы;
- список ошибок;
- стол дискуссий;
- адрес электронной почты для контакта с автором и командой, работавшей над созданием *Java Gently*.

Этой книге посвящен также узел

`www.cs.up.ac.za/javagently`

Для кого предназначена книга

Данная книга предназначена в первую очередь для студентов колледжей и университетов, начинающих изучать программирование на языке Java и имеющих доступ к пакету JDK 1.3 (Java Development Kit) компании Sun, который может работать на платформах PC, Macintosh и Unix. Здесь не упоминается ни одна из огромного числа существующих интегрированных сред разработки. А поскольку для работы не требуется особых знаний по математике, то даже старшеклассники, изучающие в школе компьютерные дисциплины, могут использовать это издание в качестве учебника. В книге подобраны самые разнообразные примеры, которые отражают многие проблемы, часто возникающие в различных сферах деятельности. Поэтому она будет полезна не только для желающих быстро освоить язык Java, но и для опытных программистов.

Изложение материала базируется на большом опыте преподавания языка Java (с 1996 года), а также ряда других языков программирования (с 1980 года) на курсах, предназначенных для студентов-первокурсников научных и инженерных факультетов университетов. Все примеры, вошедшие в книгу, были протестированы на JDK 1.3 на компьютерах Intel PC с установленными операционными системами Windows и Linux, а большинство из них также на iMac и PowerBook G3 с Java 1.16 на Mac OS9 и CodeWarrior 4. Броузером, применявшимся для просмотра апплетов, в основном служил Netscape Communicator с HotJava, но иногда использовался и Internet Explorer.

Поскольку описанные технологии и сам язык Java постоянно усовершенствуются, очень даже возможно, что появление новой версии Java совпадет с выходом этой книги. По поводу последней информации и обновлений, сделанных в программах, вы можете проконсультироваться на нашем Web-узле.

Благодарности

Третье издание *Java Gently* было в значительной степени улучшено благодаря усилиям и стараниям многих людей, которые писали мне письма, делились своим опытом и идеями. Я бы хотела выразить особую признательность за огромный вклад в создание книги Дженсу Каасболлу из Университета в Осло и Уве Кастенсу из Университета в Падеборне за их, безусловно, полезную критику тех глав, которые посвящены объектно-ориентированному программированию.

Я также благодарна техническому персоналу и студентам исследовательского департамента, которые помогали решать многочисленные проблемы, быстро тестируя Java на различных платформах. Отдельное спасибо Базилу Вороллу: — он посвятил много времени поиску ошибок, проверяя программы до тех пор, пока не убеждался в их корректности. Базил взял на себя ответственность и за диаграммы, примеры по базам данных и за обновление нескольких более ранних программ. Я и мечтать не могла о таком способном и преданном ассистенте! Однако все те ошибки, которые остались незамеченными, я считаю своими.

Джонни Ло постоянно занимался обновлением Web-узла *Java Gently*, ежедневно администрируя его и вместе с Базилом отвечая на поступающие вопросы. Нужно отметить вклад таких членов команды *Java Gently*, как Луис Бота, Тони Абботт, Олвин Мулмен, Джон Бота, Греми Пайл и Дениел Эктон. Хочу поблагодарить мой департамент за предоставленное оборудование и за бесперебойную телефонную связь, что давало мне возможность трудиться даже ночью. Моя работа была бы невозможной и без Бетховена, Брамса, Битлз, без веселой и ободряющей электронной почты от моих друзей с западного побережья Тихого океана.

Программное обеспечение было получено от компаний Sun, Microsoft и Apple, и я считаю своим долгом отдать дань уважения этим гигантам компьютерной индустрии за простоту его использования и стабильность в работе, что мы так часто принимаем как нечто само собой разумеющееся. Большинство программ было разработано на рабочей станции с Windows NT, с применением JDK 1.3 и TextPad 4, иметь дело с которыми — одно удовольствие.

Компания Pearson Education предоставила первоклассный профессиональный сервис в подразделениях редактирования и производства, благодаря чему книга была напечатана вовремя. Выражаю благодарность Мартину Сагдену, который сделал даже больше, чем мы договаривались, а также Кейт Бревин, Мартину Клопстоку, Бриджит Оллен и их командам.

А в завершение я хочу поблагодарить Найгеля — за его поддержку и помощь в решении трудных проблем, которые так часто почему-то возникают поздно ночью, моим дорогим сыновьям, Вильяму и Майклу, за их понимание и терпение в течение всех тех месяцев, когда я была полностью занята книгой. Они пока изучают Паскаль, но я надеюсь, что, поступив в университет, освоят язык Java и с удовольствием будут с ним работать.

Джуди М. Бишоп
Претория, Южная Африка
Ноябрь 2000 г.

ГЛАВА 1



Введение

1.1. Добро пожаловать в Java

Язык программирования Java не похож ни на один из существовавших ранее. Он разработан для того, чтобы облегчить работу на компьютерах в WWW посредством броузеров. В настоящее время на каждом компьютере имеется броузер. В свою очередь, каждый броузер — будь то Netscape, Explorer или Mosaic — поддерживает Java. Это означает, что вы можете просматривать документы, хранящиеся в разных точках земного шара, и, щелкнув на ссылке, активизировать Java-программу, которая будет передана через сеть и запущена на вашем компьютере.

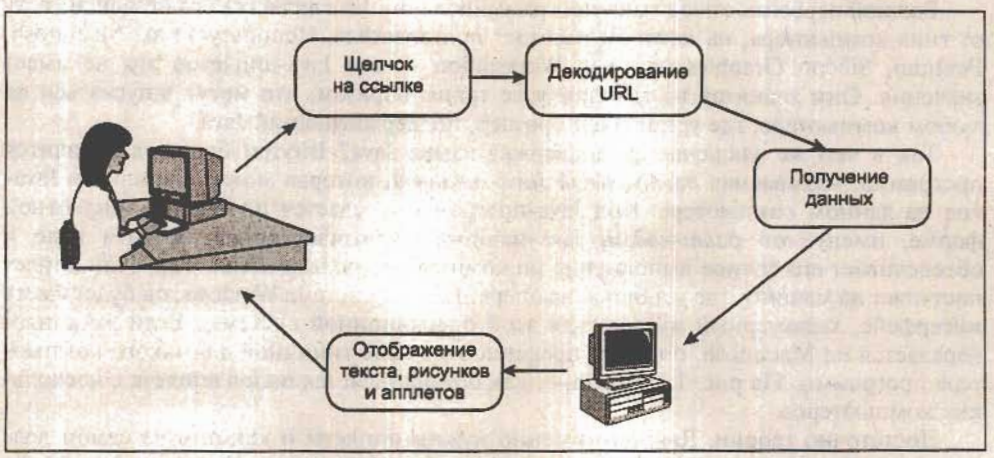


Рис. 1.1. Навигация в WWW с помощью java-апплетов

Одним из основных преимуществ, предоставляемых языком Java, является возможность работать не только со статичными текстами и графикой, но и с различными динамическими объектами. На рис. 1.1 представлена схема цикла работы броузера, поддерживающего Java. Программы, которые запускаются в сети, называются апплетами (от англ. *applet* — little application, или маленькое приложение). В дальнейшем мы будем пользоваться этим термином.

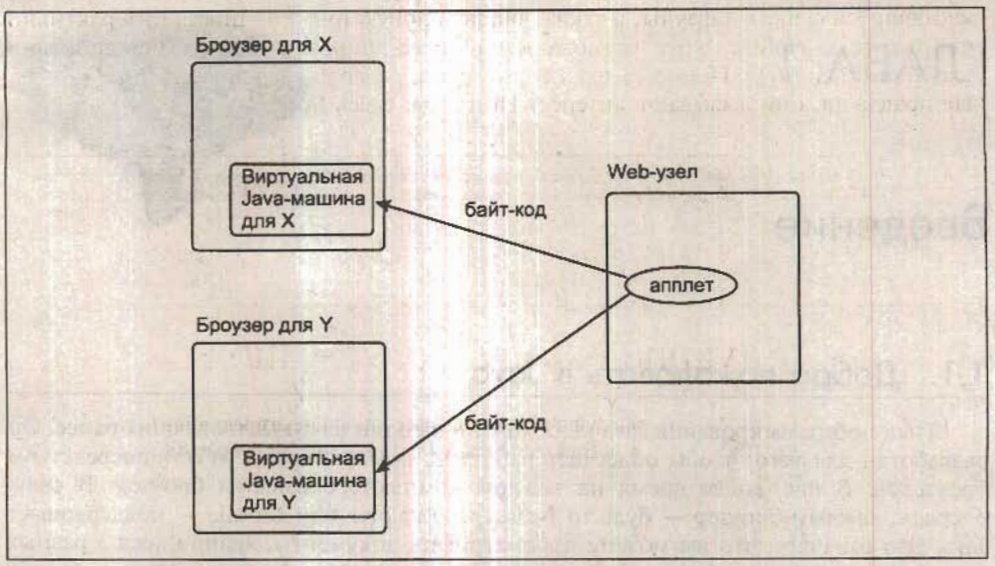


Рис. 1.2. Java-апплет, запрашиваемый двумя разными компьютерами X и Y

Важной особенностью языка программирования Java является его независимость от типа компьютера, на котором работает пользователь. Используете вы Macintosh, Pentium, Silicon Graphics или Sun Workstation — для Java-апплетов это не имеет значения. Они хранятся на главном узле таким образом, что могут запускаться на любом компьютере, где установлен броузер, поддерживающий Java.

Так в чем же заключается поддержка языка Java? Внутри броузера находится программа, называемая *виртуальной Java-машиной*, которая может выполнять Java-код на данном компьютере. Код Java-программ передается по сети в стандартной форме, именуемой *байт-кодом*. Java-машина распознает апплет в этом коде и обеспечивает его точное выполнение на компьютере пользователя. Так, если апплет поступает на машину, где установлена операционная система Windows, он будет иметь интерфейс, характерный именно для этой операционной системы. Если же апплет передается на Macintosh, он будет представлен в виде типичной для такого компьютера программы. На рис. 1.2 показано, как осуществляется вызов апплета с нескольких компьютеров.

Достаточно теории. Для чего именно нужны апплеты и как они на самом деле выглядят? Давайте рассмотрим пример, который представлен на Web-узле *Java Gently*. Вы сможете найти его по адресу* http://www.booksites.net/bishop.

Проект охраны природы

В рамках международной благотворительной программы "Саванна" (Саванна — страна, расположенная в африканских степях) финансируется проект создания в WWW информационной базы данных о природе. Посетители Web-узла могут делать запросы,

* Термин URL, который присутствует в одной из надписей на рис. 1.1, является сокращением от Universal Resource Locator (Универсальная система поиска ресурсов). Это уникальная ссылка на документ в Web, которая обычно состоит из названия протокола, например http, и имени узла или компьютера, где хранится метка, за которой следует информация о поддиректории.

касающиеся флоры и фауны, а также вносить новую информацию в интерактивном режиме (как видите, круг возможностей более широк, чем при использовании электронной книги). Посмотрите на Web-страницы первой версии узла (рис. 1.3—1.5). Не правда ли, они вызывают интерес? Но где же здесь Java?

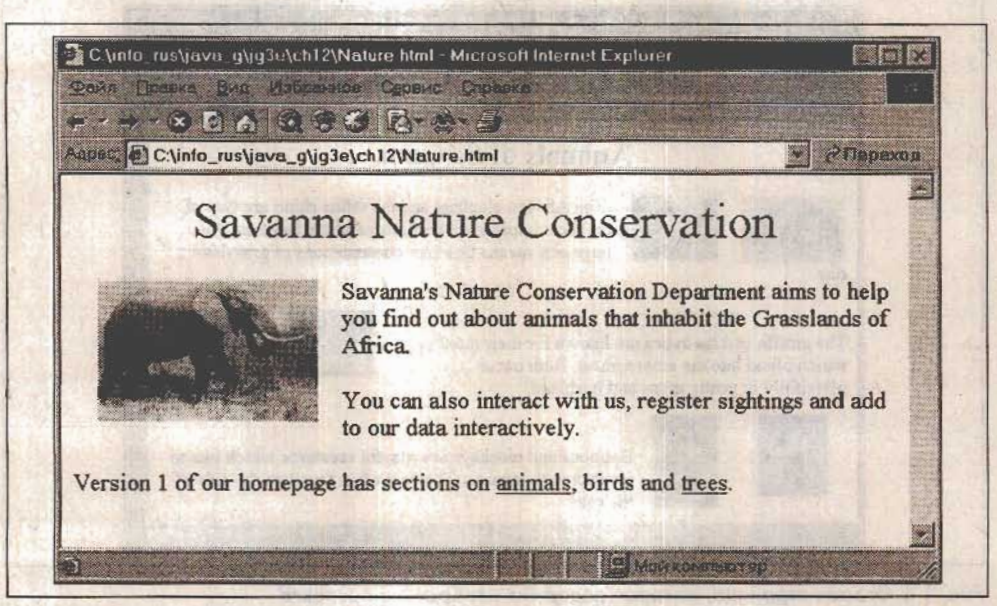


Рис. 1.3. Первая страница Web-узла содержит ссылки на разделы о животных и деревьях

Полезность и возможности данного языка программирования сразу не очевидны. Наша первая Web-страница (рис. 1.3) анонсирует проект и содержит готовые ссылки на разделы, посвященные животным и деревьям. Раздел о птицах еще не готов, поэтому ссылка не активна. Если щелкнуть на ссылке, указывающей на раздел о животных, загрузится страничка, представленная на рис. 1.4. В данный момент здесь можно найти только подборку фотографий, в то время как раздел о растениях (рис. 1.5) уже содержит таблицу, где демонстрируется, как систематизировать растения в зависимости от вида их листьев. Посетителю предлагается ввести три параметра интересующего дерева. Далее начинает работу апплет, ожидающий своей очереди на пользовательской машине. Основываясь на указанных значениях, апплет вычисляет код дерева, а затем дает указание броузеру загрузить с Web-узла информацию о нем. В результате на Web-странице выводится информация об определенном дереве, например фотография баобаба, сведения о его семенах, цветах и месте распространения. Одна из таких страниц показана на рис. 1.6. Более подробную информацию об указанном апплете вы найдете в разделе 12.4.

Если пара значений вид/размер листа не знакома апплету, он предоставит вам возможность задать этот вопрос опытному ботанику, который даст квалифицированный ответ по электронной почте.

Конечно, передача данных по сети — не единственное назначение Java. Это еще и язык программирования, такой же как C++, Pascal, Ada и Modula-3. Научить читателя работать с Java — вот основная задача данной книги, однако она служит еще и общим введением в программирование. Дочитав книгу до конца, вы будете

готовы создавать Java-приложения и апплеты для решения разнообразных задач, таких как инвентаризация, усреднение и сортировка по признакам, разработка игр и проведение викторин, а также путешествие по Web.

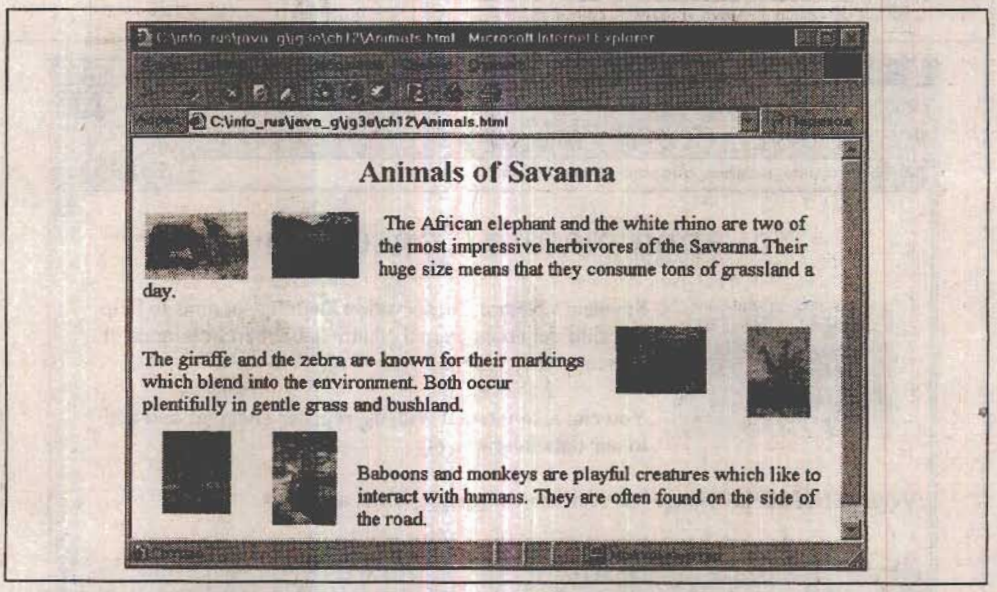


Рис. 1.4. Вторая страница, которая содержит изображения и текст

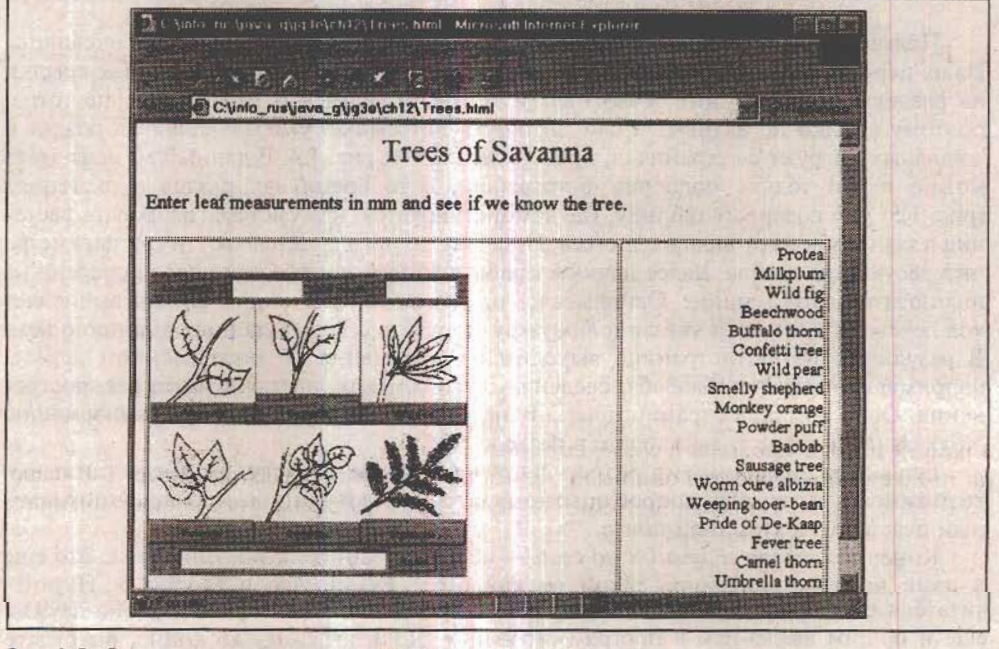


Рис. 1.5. Страница, посвященная деревьям, содержит апплет

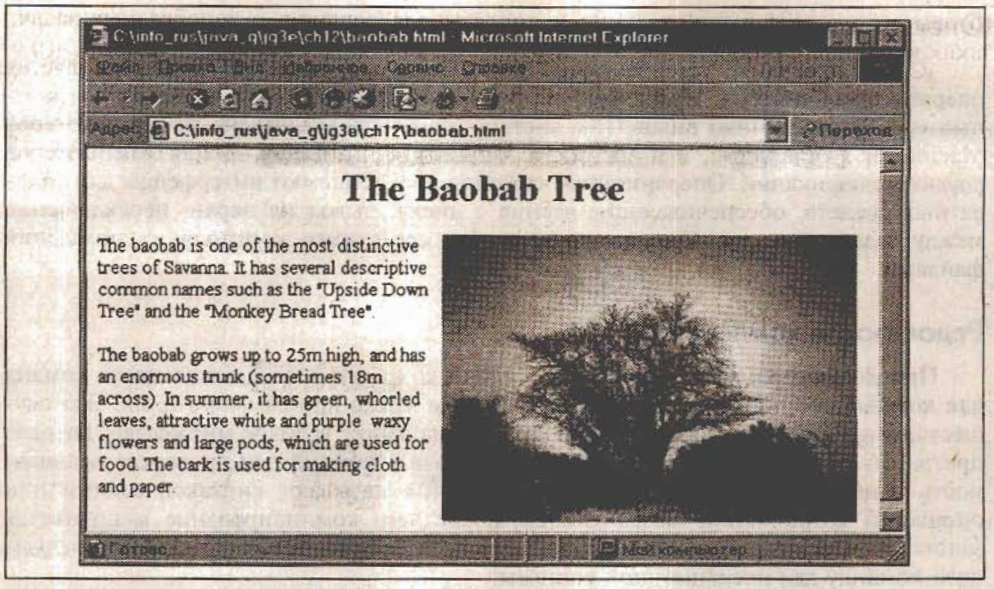


Рис. 1.6. Web-страница, посвященная выбранному дереву

1.2. Разработка программного обеспечения

Компьютер состоит из аппаратных средств и программного обеспечения. Как правило, прежде чем купить компьютер или приступить к его использованию, мы уделяем повышенное внимание именно аппаратному обеспечению: каков объем памяти, какова скорость процессора и т. д. Данные параметры, несомненно, являются существенными, но не самыми важными. Конечно, досадно, когда не хватает памяти на диске, но можно, по крайней мере, часть информации сохранить в другом месте, а часть — удалить, чтобы освободить пространство.

Мы же хотим подчеркнуть, что именно программное обеспечение определяет качество работы компьютера. Далее дан обзор программных средств, которые необходимы для программирования на Java, а также описано их взаимодействие. Этот материал является универсальным, он изложен без привязки к конкретным программным продуктам или поставщикам. В завершение мы вкратце рассмотрим этапы разработки программного обеспечения.

Процесс программирования

Цель процесса программирования — создание программ, которые должны храниться в памяти и быть готовыми для запуска и выполнения. Программы могут выполняться на том же компьютере, где они были написаны, или же передаваться по сети. В наше время для программирования требуется достаточно обширный набор программного обеспечения. Некоторые продукты устанавливаются на компьютере при его покупке, а часть приходится приобретать отдельно. Вот необходимый минимум для разработки программного продукта: операционная система, редактор и компилятор.

Операционные системы

Среди программ, установленных на компьютере, одной из главных является операционная система. Некоторые системы специально разрабатываются для компьютеров определенных видов. Так, система MacOs предназначена для компьютеров Macintosh, а остальные, и в частности Windows, Unix, Linux, — для компьютеров других модификаций. Операционные системы предоставляют интерфейсы для аппаратных средств, обеспечивающие чтение с диска, вывод на экран, переключение между задачами и т.д. Они также являются средствами контроля и управления файлами.

Редакторы и компиляторы

Процесс программирования заключается в создании последовательности команд для компьютера, написанных на определенном языке программирования. Это осуществляется с помощью редакторов, которых на рынке великое множество. Готовая программа обрабатывается компилятором. Сначала компилятор проверяет правильность написанного. Обнаруженные ошибки (их называют синтаксическими или ошибками компиляции) исправляются, после чего компилирование выполняется заново. Компилятор языка Java активизировать несложно, достаточно ввести следующую команду для операционной системы:

```
javac Trees.java
```

Здесь Trees.java — это файл, который создан с помощью редактора. Если в программе нет синтаксических ошибок, компилятор транслирует программу в ту форму, в которой она может быть запущена на компьютере. Для компиляторов большинства других языков такой формой являются команды для определенного процессора. Однако в случае Java цель компиляции состоит в создании виртуальной Java-машины, для которой нет разницы, на каком именно компьютере будет запускаться готовая программа.

Компилятор проверяет грамматику программы и при этом может выявлять достаточно много ошибок, однако он не способен обнаружить ошибку в логике. Например, весьма велика вероятность того, что в апплете, показанном на рис. 1.5, небрежный программист значение ширины ошибочно воспримет как длину или наоборот.

Чтобы избежать логических ошибок, надлежит соблюдать следующие правила:

- тщательно следить за структурой программы;
- пользоваться хорошими руководствами по программированию;
- повторно использовать существующие фрагменты программы (место того чтобы каждый раз вводить их сначала).

Тем не менее иногда единственный способ отследить логическую ошибку заключается в запуске программы и просмотре ее результатов. Это второй этап процесса разработки программного обеспечения. Программа выполняется или запускается путем активизации с помощью операционной системы. В Java для этой цели предназначена такая команда:

```
java Trees
```

Некоторые логические ошибки могут быть обнаружены при внимательном тестировании программы с использованием тщательно отобранных данных. После исправления ошибок тест необходимо повторить, поскольку, устраняя одну ошибку,

легко допустить другую. Все эти темы освещены в данной книге. Действия, которые необходимо выполнить для отладки программы, перечислены на рис. 1.7.

Среды разработки

Так как в процессе программирования цикл редактирование—компиляция повторяется многократно, существует программное обеспечение, которое сочетает эти два этапа и позволяет программисту переключаться с одного процесса на другой нажатием кнопки. Примером таких Java-систем являются TextPad, Kawa, VisualJ++, VisualAge и JBuilder. Интегрированная среда разработки (ИСР) включает также средства для управления большими проектами (даже с привлечением нескольких программистов). Еще один ее отличительный признак — наличие усовершенствованного редактора для создания пользовательских интерфейсов.

Одним из важнейших преимуществ некоторых ИСР для Java является то, что, как и сам язык программирования, они не зависят от платформы, благодаря чему вы можете продолжать работу в одной и той же среде, даже если меняете компьютер. Их недостаток заключается в том, что они не всегда наглядны, поэтому для того, чтобы научиться эффективно в них работать, требуется некоторое время.

Сопровождение

На рис. 1.7 указано только около 30% операций, выполняемых в процессе программирования. Самое трудное для программиста начинается, когда готовый продукт поступает в эксплуатацию. На этом этапе продукт требует сопровождения, которое заключается в исправлении ошибок, а также выполнении необходимых для пользователя изменений и усовершенствований. Даже во время обучения программист должен стремиться сделать подобную работу как можно более легкой.



Рис. 1.7. Действия, выполняемые в процессе программирования на Java

1.3. Методика

В данной книге мы будем учиться программированию на примерах. Здесь вашему вниманию предлагаются проверенные программы, благодаря которым вы получите представление как об основных принципах программирования, так и о сложных конструкциях языка. Примеры тщательно отобраны и классифицированы таким образом, чтобы в каждом из них присутствовали только те характерные элементы, которые рассматривались ранее.

Работа над каждым примером будет включать следующие этапы:

- постановка задачи;
- определение путей ее решения;
- построение алгоритмов решения задачи;
- разработка классов (в случае необходимости);
- анализ нюансов, которые должны быть учтены в программе;
- написание программы;
- тестирование программы.

Задача — это краткое изложение того, что нужно сделать. На втором этапе определяется подход к решению задачи, а также осуществляется выбор программного обеспечения. Построение алгоритма является ключевым моментом решения задачи. Каждая программа — это цельная запускающаяся Java-программа, которую можно найти на нашем Web-узле по адресу http://www.booksites.net/bishop.

Алгоритмы

Алгоритм — это точная, недвусмысленная последовательность действий, которые необходимо выполнить для решения задачи. Существует несколько способов подачи алгоритма, мы будем использовать сочетание русского языка, линий и стрелок. Итоговая диаграмма демонстрирует общую структуру программы, правда она еще в недостаточной степени формализована. Например, в апплете Trees необходимо ввести три значения. Алгоритм, позволяющий вводить эти значения в любом порядке, показан на рис. 1.8.

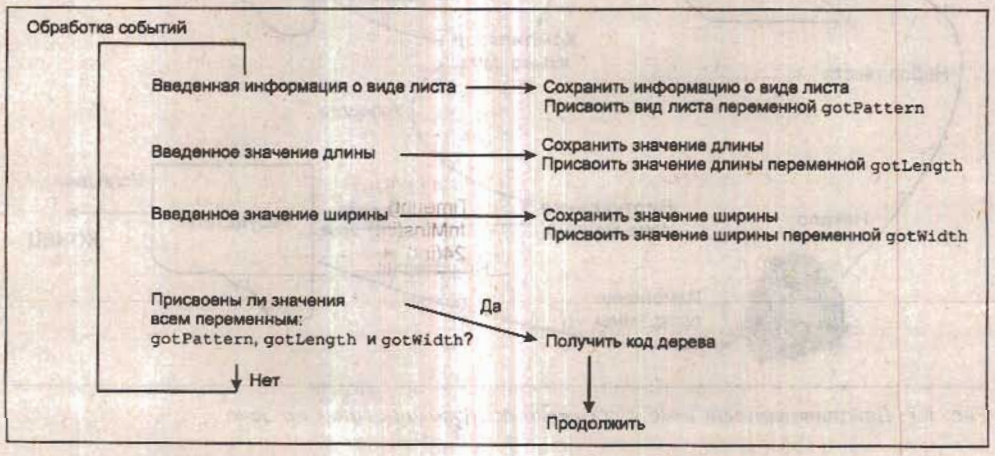


Рис. 1.8. Пример алгоритма

Выполнение алгоритма осуществляется сверху вниз. Боковые стрелки указывают на действие, которое производится, если условие истинно. Часто алгоритм требуется повторить. Тогда следует ориентироваться по линии, которая расположена слева, направлена снизу вверх и возвращается к вершине.

Разработка классов

Java является объектно-ориентированным языком программирования. Это означает, что написанная на нем программа состоит из взаимосвязанных объектов. В этой книге при описании программы демонстрируются ее объекты и указываются способы взаимодействия между ними. Для этих целей применяются стандартные обозначения языка моделирования UML с дополнениями, призванными облегчить его понимание. Они позволяют наглядно показать интересные свойства Java, например провести четкое разграничение между объектами и классами, а также обеспечить описание массивов и хеш-таблиц. Пример диаграммы моделирования, связанной с проектом охраны природы, приведен на рис. 1.9.

`PizzaDelivery1` — это класс (обозначен пунктирными линиями), из которого создается объект (обозначен сплошными линиями). Объект имеет три переменные, которым присвоены соответствующие значения, и семь методов. Более подробно о языке UML рассказывается в главах 2 и 3.

Программа

Для создания программ предназначены специальные языки программирования. Существует большое количество языков, но только малая их часть подходит для компьютеров всех типов. К разряду универсальных можно отнести языки Pascal, Ada, FORTRAN, COBOL, BASIC, C++, Lisp, Prolog, Modula и, конечно, Java. Этот язык разработан в 1993 году. В то время он носил название Oak. Популярность пришла к Java в 1996 году, когда в сети появились бесплатные компиляторы этого языка компании Sun Microsystems.

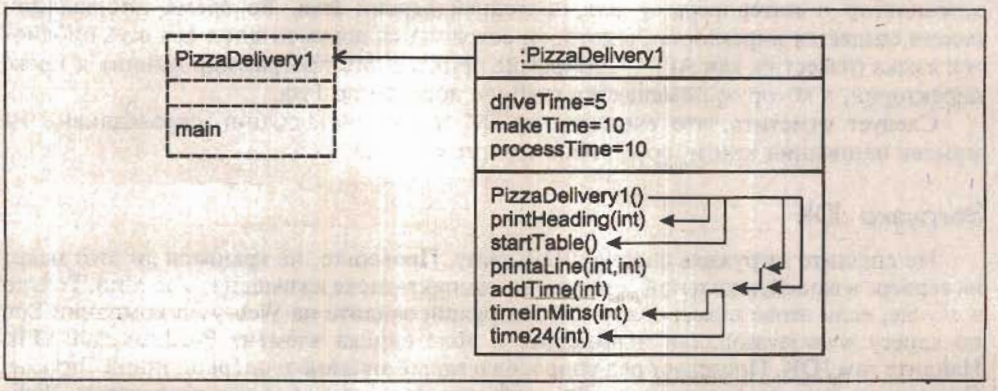


Рис. 1.9. Пример диаграммы моделирования

Синтаксис программы имеет большое значение: любая ошибка может привести к тому, что компилятор сведет на нет ваши усилия. Таким образом, очень важно основательно выучить синтаксис языка. Чтобы помочь вам в этом, автор книги при

вводе новой конструкции особым образом демонстрируют ее синтаксис и семантику, например:

Оператор If

`if (условие) {`
`оператор1;`
`}`
`else {`
`оператор2;`
`}`

Выражает *условие*. Если оно истинно, то выполняется *оператор1*, иначе — *оператор2*. Часть `else` является необязательной: если ее нет, то в случае, когда условие не выполняется, программа переходит к оператору, следующему за `If`.

Значение жирного шрифта и курсива вы поймете, когда мы приступим к изучению операторов языка Java.

Резюме

В конце каждой главы в краткой форме излагаются ее основные положения, приводится список ключевых слов Java, библиотек и элементов, там могут также присутствовать формы и диаграммы моделирования. Список форм включен и в приложения, расположенные в конце книги.

1.4. Начинаем работать с Java

Настало время поговорить об установке средств разработки Java. Система Java Development Kit (JDK) — это свободно распространяемый пакет от фирмы Sun, предназначенный для разработки программ на языке Java. JDK содержит эталонный компилятор и интерпретатор для последней версии Java. Во время инсталляции пакета создается директория, в которой сохраняется документация для всех библиотек языка (известны как API — интерфейс прикладного программирования), а также директория, в которую помещается учебное пособие по Java.

Следует отметить, что сведения о JDK и его инсталляции, справедливые на момент написания книги, со временем могут устареть.

Загрузка JDK

Не спешите загружать систему JDK сразу. Проверьте, не хранится ли этот пакет на сервере в вашей локальной сети или на компакт-диске из вашей коллекции. Только в случае, если этого пакета нет у вас под рукой, зайдите на Web-узел компании Sun по адресу www.javasoft.com и выберите в поле списка элемент Products and APIs. Найдите там JDK. Последует ряд вопросов о вашей машине и операционной системе. Ответив на них, вы сможете начать загрузку. Вам следует также распечатать Web-страницу, связанную с загрузкой, где рассказывается, как инсталлировать и тестировать пакет JDK.

Итак, вы получили в свое распоряжение пакет JDK. Следующий этап — его инсталляция. Тщательно выполняйте требования инструкции, особое внимание уделяйте тем пунктам, которые касаются указания пути и определения полного имени класса.

Тестирование системы

Для того чтобы проверить, правильно ли вы инсталлировали систему, воспользуйтесь простым текстовым редактором. Введите в нем первую программу на языке Java (пример 2.1) и сохраните ее в файле Welcome.java. Не выходя из редактора, откройте окно командной строки (Unix или MS-DOS) и введите команду

```
javac Welcome.java
```

Компилятор JDK загрузит файл и откомпилирует его. Если при вводе не было допущено ошибок, выполнение команды довольно быстро завершится. Если же появились сообщения об ошибках, вернитесь в окно редактора, внесите соответствующие исправления в программу, после чего повторите компиляцию.

Предположим, компиляция прошла успешно. В этом случае компилятор создаст файл Welcome.class. Запустить программу позволит команда

```
java Welcome
```

(Обратите внимание на то, что слово "class" добавлять не нужно). Результаты работы программы будут представлены в окне командной строки.

Пакет Javagently

Когда вы приступите к работе с языком программирования Java, вам придется взаимодействовать со своей программой, что наиболее просто осуществлять с помощью специальных классов пакета Javagently, который написан специально для данной книги. Детальную информацию о нем, а также указания по его инсталляции вы сможете получить в разделах 4.2 и 4.3. В следующих главах будет дан полный перечень классов. Если вы сможете получить этот пакет по сети (см. раздел 4.3), вам не понадобится создавать его вручную.

Документация и учебное пособие

Вместе с JDK поставляется очень ценная дополнительная информация. В документации представлены все библиотеки языка Java с детальным описанием каждого метода и соответствующими примерами. Документация — это справочная система для работающего с Java программиста, доступная через броузер. Учебное пособие содержит подборку статей разных авторов, написанных в разное время и на различных уровнях, а также полезные примеры программ.

Интегрированные среды разработки для языка Java

Хотя система JDK бесплатна и многие используют исключительно ее, на рынке представлены более совершенные системы, которые описаны в разделе 1.2. Некоторые из них располагают собственными компиляторами, но большинство переходит на использование компиляторов фирмы Sun, которые считаются более современными.

Широкое распространение получили такие ИСР: Java Workshop компании Sun, Visual J++ компании Microsoft, Jbuilder компании Borland, Visual Age for Java компании IBM, Kawa компании Tek-Tool и CodeWarrior компании Metrowerks. Все они подходят для PC-платформ, где используется Windows 95 или NT либо более поздние версии этой системы. Кроме того, Java Workshop функционирует под управлением операционной системы Solaris фирмы Sun, а CodeWarrior — под управлением системы Macintosh.

Существует более простой класс ИСР, предназначенных для редактирования и компиляции без применения средств управления проектами. В первую очередь заслуживает внимания среда TextPad, запросить которую можно по адресу sales@textpad.com.

Web-узлы, посвященные Java

Языку Java посвящено немалое количество интересных Web-узлов, где публикуются статьи о нем, а также размещаются различные Java-ресурсы. Рекомендуем обязательно посетить следующие четыре узла:

- www.javasoft.com — узел фирмы Sun, на котором постоянно обновляются тематические статьи, а также публикуются сведения о новых продуктах Java и методах их использования;
- www.gamelan.com — потрясающая коллекция апплетов, ресурсов языка Java и статей о нем;
- www.javaworld.com — ежемесячный сетевой журнал;
- www.booksites.net/bishop — Web-узел этой книги, содержащий справочный материал.

Web-узел книги Java Gently

Первое издание этой книги вышло в свет почти одновременно с появлением Web-узла, на котором изначально размещались описанные в ней программы. С мая 1997 года узел увеличился в объеме и стал интерактивным. Теперь на нашем узле посетители со всего мира могут участвовать в дискуссиях, делиться мнениями и обмениваться идеями, касающимися языка Java и содержания книги. Ежедневно узел посещают более 100 000 человек.

На Web-узле книги *Java Gently* представлены следующие материалы:

- информация о книге;
- краткое содержание глав;
- список учреждений, использующих данную книгу для обучения;
- примеры, описанные в книге;
- ответы на часто задаваемые вопросы;
- дискуссионная страница, управляемая автором и командой JG;
- информация о Java, представляющая интерес для читателей в данный момент;
- материалы о Java от автора книги;
- список допущенных в книге ошибок;
- планы и даты выпуска будущих версий книги;
- ссылка на узел издателя.

Web-узел книги *Java Gently* помимо информационной может выполнять еще две функции. Во-первых, его можно применять во время лекций в классах и аудиториях с доступом к сетевым средствам, при котором программы могут быть просмотрены прямо на узле. Этот метод обучения считается более эффективным, чем копирование. Возможность быстро переключаться между программами, сохранять, компилировать и запускать их способствует созданию среды динамического обучения.

Во-вторых, вы можете связаться по сети с командой JG, которая ответит на вопросы и обсудит любые проблемы, касающиеся языка программирования Java. Сначала в состав команды входили специалисты из учреждения, в котором работает

автор, но в последнее время она расширилась за счет читателей данной книги, имеющих доступ к узлу и готовых поделиться своим опытом.

Web-узел регулярно обновляется, ваши комментарии и мнения всегда будут приняты во внимание. На все вопросы вы получите ответы, а мы, в свою очередь, будем признательны вам за любое содействие.

Резюме

Часто приходится слышать, что программирование — увлекательное занятие. Возможно. В данной книге сделано все возможное, чтобы это суждение оказалось истинным. Читатели смогут получить поддержку профессионалов со всего мира, в частности на Web-узле www.booksites.net/bishop. Независимо от того, насколько вам понравится учиться программированию, надеемся вы запомните следующую "модификацию" бессмертного стихотворения Редьярда Киплинга:

О, если ты покоен, не растерян,
Когда теряют головы вокруг,
И если ты себе остался верен,
Когда в тебя не верит лучший друг,
И если будешь мерить расстоянье
Секундами, пускаясь в долгий с Java в путь,—
Мир Web — твое, мой мальчик, достоянье!
И более того, ты — программист!



Вопросы

1.1. Каковы функции компилятора?

1.2. Чем отличается ошибка компиляции от логической ошибки?

1.3. Какова тактовая частота компьютера, который вы используете?

1.4. Какой объем памяти у вашего компьютера?

1.5. Какова емкость дисков на вашем компьютере?

1.6. Поместится ли этот учебник (включая все пробелы) у вас на диске, если исходить из того, что 1 символ занимает 1 байт памяти?

1.7. Составьте список прикладных программ, которые вы обычно используете в течение недели.

1.8. Сталкивались ли вы когда-либо с компьютерной ошибкой? Если да, то была ли она в программе или возникла из-за неверного ввода данных.

1.9. Узнайте название, версию и дату выпуска компьютера, операционной системы и компилятора, которые вы будете использовать.

1.10. Посетите Web-узел *Java Gently* (http://www.booksites.net/bishop) и ознакомьтесь с базой данных о природе.

ГЛАВА 2



Простые программы

В этой главе мы рассмотрим структуру программ, написанных на языке Java, и поговорим о том, как согласуются между собой их фрагменты. Три тщательно подобранных примера позволят вам познакомиться с основами объектно-ориентированного программирования. Изучив материал главы, вы сможете самостоятельно написать простую программу. Предлагаемые в ниже программы будут использоваться при решении последующих задач.

2.1. Демонстрационные программы

В начале главы представлены три небольшие программы, которые иллюстрируют различные аспекты программирования на языке Java. Первая выводит текстовую информацию на экран, а вторая и третья демонстрируют, какие возможности обеспечивает язык Java при работе с графикой и объектами.

Пример 2.1. Welcom

Наша первая задача — вывод на экран сообщения "Welcome to Java Gently!". Программа, выполняющая данную операцию, имеет следующий вид:

```
class Welcome {
/*Программа Welcome Дж. М. Бишоп, декабрь 1996 г.
* ----------------- Java 2 апрель 2000 г.
* Простая программа, выводящая текстовое сообщение на экран.
*/
Welcome () {
      System.out.println("Welcome to Java Gently!");
}
public static void main (String [] args) {
new Welcome ();
}
}
```

В результате выполнения данной программы на экран будет выведено сообщение*:

```
Welcome to Java Gently!
```

* Далее для оформления Java-кода будет использоваться шрифт Courier, а для выделения результатов выполнения программы — жирное начертание.

На данном этапе следует обратить внимание на следующие моменты. Текст между символами /* и */ называется **комментарием** и используется для того, чтобы пояснить читателю, каковы функции программы. Комментарии не обрабатываются компьютером. За командой `println` в кавычках следует информация, которая должна быть выведена на экран. Остальной код — это основная часть Java-программы. В дальнейшем мы изучим ее более подробно.

Упражнение. Запустите программу Welcome (как это сделать, рассказывается в разделе 1.4). Измените ее таким образом, чтобы выводимое на экран сообщение включало ваше имя, например: "Welcome to Java, Peter!". Поэкспериментируйте: модифицируйте программу так, чтобы приветствие на экране располагалось в нескольких строках или выделялось подчеркиванием.

Пример 2.2. Рисование флага

Одно из достоинств языка Java — наличие встроенных средств, позволяющих работать с графикой и цветом, использовать анимацию, звук и т.д. Следующая программа выводит на экран изображениефлага.

```
import java.awt.*;
import java.awt.event.*;

class FlagMaker1 extends Frame {

/* Программа, создающая изображение флага. Дж.М.Бишоп, апрель 2000 г.
 * ------------------------------------
 * Иллюстрирует вывод цветного графического изображения.
 */

FlagMaker1 () {
Add ("Center", new Flag ());
//Завершает программу при закрытии окна.
    addWindowListener (new WindowAdapter (){
      public void windowClosing(WindowEvent e) {
          System.exit (0);
      }
    });

//Задает заголовок и размер фрейма, активизирует функцию рисования paint.

    setTitle ("A Flag");
   setSize (300, 200);
   setVizible (true);
 }

 public static void main (String [ ] args) {
    new FlagMaker1 ();
 }
}

class Flag extends Canvas {
    public void paint (Graphics g) {
       //Рисует флаг, используя цветные прямоугольники.
```

```
        g.setColor (Color.black);
        g.fillRect (40,40,200,40);
        g.setColor (Color.red);
        g.fillRect (40,80,200,40);
        g.setColor (Color.yellow);
        g.fillRect (40,120,200,40);

       //Подпись под рисунком.
       g.setColor (Color.black);
       g.drawString ("Germany",100,180);
    }
}
```

Результат работы программы показан на рис. 2.1 (к сожалению, в черно-белом варианте).



Рис. 2.1. Графика, выведенная на экран программой FlagMaker1

Программа `FlagMaker1` имеет несколько больший объем, чем программа `Welcome`, и содержит два класса — `FlagMaker1` и `Flag`. Первый выполняет вспомогательные функции при создании нового окна для отображения флага. Основная задача программы решается с по́мощью метода `paint` класса `Flag`. Каждая из строк наподобие

```
g.fillRect (40, 80, 200, 40);
```

рисует один из цветных прямоугольников. Числа в скобках — это размер прямоугольника в пикселях (экран может иметь ширину до 600 пикселей). Элементы списка, расположенного после метода `fillrect`, имеют следующие значения:

- координата X левого верхнего угла прямоугольника;
- координата Y левого верхнего угла прямоугольника;
- ширина прямоугольника;
- высота прямоугольника.

Изменяться будет только значение координаты Y, поскольку элементы рисунка создаются сверху вниз. Более подробно возможности Java по работе с графикой рассматриваются в главе 10.

Оставшаяся часть программы содержит инструкции Java, обеспечивающие доступ к графическим средствам этого языка программирования, которые не имеют

непосредственного отношения к нашей задаче. Вот краткий перечень предоставляемых ими возможностей:

- передача в программу графических пакетов (`awt` и `awt.event`);
- закрытие окна;
- расположение флага в центре окна (или *фрейма*);
- создание графического окна определенного размера и с определенным заголовком.

Пример 2.2 иллюстрирует употребление комментариев второго типа — однострочных, вводимых между символами //. Java позволяет использовать любой из описанных нами типов комментариев. Преимущество однострочного комментария заключается в том, что он автоматически завершается по достижении конца строки. Однако многие программисты считают, что у комментариев первого типа более элегантный вид. В таких комментариях принято располагать звездочку и косую */, которые закрывают комментарий, под символами /*, которые его открывают.

Умение работать в графической среде является очень важным для программиста, а для этого необходимо изучить множество функций (информацию о большинстве из них вы при необходимости найдете в справочнике). Поэтому мы пока не будем останавливаться на возможностях Java по обработке графики (этот материал изложен во второй части книги), а сосредоточим внимание на написании обычных объектно-ориентированных программ.

Упражнение. Программа `FlagMaker1` выводит германский флаг с черной, красной и желтой* полосами. Модифицируйте ее так, чтобы она выводила другой подобный флаг, например Нидерландов (красный, белый, синий) или Эфиопии (зеленый, желтый, оранжевый). Измените размер окна и флага таким образом, чтобы увеличился размер результирующего рисунка.

Пример 2.3. Лавка древностей

Цель нашего третьего примера — продемонстрировать, что представляет собой объектно-ориентированная программа, написанная на языке Java. Объекты хранят информацию о предметах из реального мира (антикварных вещах), а каждый класс объектов определяет, какая функция должна быть выполнена по отношению к ним. Предположим, нам нужно составить перечень товаров, предлагаемых в лавке древностей. В очень простой программе, которая позволяет решить данную задачу, мы создадим три объекта класса `Curio` (древность) и осуществим вывод их значений (каковыми является стоимость различных древностей). Поскольку в дальнейшем эта программа будет совершенствоваться, назовем ее `CurioStore1`.

```
class CurioStore1 {

/* Curio Store No. 1 Дж М Бишоп, апрель  2000
 *-----------------
 * Иллюстрирует структуру объектно-ориентированной программы.
 */

//Объявляет три объектные переменные.
Curio mugs, tshirts, carvings;
```

* Желтый цвет должен иметь золотистый оттенок, однако, к сожалению, некоторые оттенки не могут быть нами использованы в данной программе без значительного ее усложнения.

```
//Конструктор - это та часть программы, с которой начинается
//ее исполняемая часть.
CurioStore1 () {
  //Создает три объекта с различными начальными значениями.
  mugs = new Curio("Traditional mugs", 6, "beaded in Ndebele style");
  tshirts = new Curio("T-shirts", 30, "sizes M to XL");
  carvings = new Curio("Masks", 80, "carved in wood");

  //Выводит на экран заголовок.
  System.out.println("The Polelo Curio Store sells\n");

  //Выводит на экран значения, содержащиеся в каждом объекте.
  mugs.write();
  tshirts.write();
  carvings.write();
}

//Каждая программа должна содержать метод main.
public static void main (String [ ] args) {
   //Выполнение программы начинается с конструктора.
   new CurioStore1 ();
 }
}

class Curio {

  //Объявляет переменные, соответствующие древностям.
  String name;
  int price;
  String description;

  //Конструктор копирует начальные значения в объектные переменные.
  Curio (String n, int p, String d) {
    name = n;
    price = p;
    description = d;
}

  //Метод, который осуществляет вывод значений объектных переменных на экран.
  void write () {
   System.out.println(name + " " + description + " for G" + price);
  }
}
```

В результате выполнения программы мы получим следующие данные:

```
The Polelo Curio Store sells

Traditional mugs beaded in Ndebele style for G6
T-shirts sizes M to XL for G30
Masks carved in wood for G80
```

Программа `CurioStore1` по структуре подобна `FlagMaker1`. Она содержит два класса — `CurioStore1` и `Curio`. Метод `main` в обоих случаях служит "начальной точкой" конструктора программы: с этого места начинается ее реальная работа.

В программе объявляются и создаются три объекта, а затем их значения выводятся на экран.

Каждой древности присваиваются три значения — `name`, `price` и `description` (имя, цена и описание). В классе `Curio` эти значения копируются в объект, так что они будут доступны, когда нам потребуется вывести их на экран. Обратите внимание на то, что порядок значений при выводе отличается от того, который был установлен при создании объектов (т.е. имя, описание, цена, а не имя, цена, описание).

Как мы вскоре увидим, классы, подобные `Curio`, способны описывать объекты реального мира. Внутри программы, такой как `CurioStore1` (определена как класс), создаются объекты класса `Curio`.

Упражнение. Пользуясь образцом, который дан в программе, добавьте четвертую древность в базу данных магазина, например *wovenBaskets* (плетеные корзины). Установите для нее цену, введите описание и выполните все действия, которые нужны для создания объекта и его вывода на экран.

2.2. Структура программы

Базовые элементы

Приступая к изучению языка Java, мы в первую очередь рассмотрим, как записывать инструкции, из которых состоит программа. Для этого нам необходимо ознакомиться с рядом элементов языка. Некоторые уже известны вам из примеров предыдущего раздела.

- Термин **программа** используется для описания совокупности классов, которая может быть откомпилирована и запущена для выполнения.
- **Класс** представляет собой совокупность переменных и методов.
- **Объект** создается на основе класса посредством оператора `new`. Процесс его создания называется *инициализацией*.
- **Переменная** содержит значение, с которым может работать метод. Переменные могут иметь как простые значения (`price`), так и целые объекты (`carvings`).
- **Идентификатор** — это имя единицы языка Java, такой, например, как класс. Примерами могут служить имена `CurioStore1` и `carvings`.
- **Ключевым** называется слово, которое имеет особое значение в языке Java и не может быть использовано как идентификатор (например, `class` или `new`).
- Работа программы заключается в выполнении **операторов**, из которых она состоит. Оператор вызывает определенное действие: инициализацию объекта, активизацию метода, вывод сообщения и т.д.
- **Метод** определяется идентификатором и состоит из переменных и операторов. Метод активизируется путем вызова соответствующего идентификатора. Примерами являются `main`, `write` и `paint`.
- Каждый класс имеет специальный метод, называемый **конструктором**, который активизируется, когда инициализируются объекты или классы.
- Метод может иметь различные варианты, в зависимости от значений, которые определяются в круглых скобках, — **параметров**.

На рис. 2.2 показана стандартная структура программы. На данном этапе изучения Java нет необходимости немедленно вникать во все, что касается элементов этого языка программирования: мы детально рассмотрим каждый их в ходе изучения.

Диаграммы моделирования

На рис. 2.2 отсутствует объект: где же он? Объект, на котором основывается программа, инициализируется в методе `main` посредством оператора

```
new Welcome ();
```

Использование объекта в Java-программе демонстрируется с помощью диаграмм моделирования. Вы не раз встретите их на страницах нашей книги. На рис. 2.3 показана диаграмма моделирования для программы `Welcome`. Условные обозначения заимствованы из универсального языка моделирования, а употребляемые дополнения предназначены для иллюстрации большего количества элементов на элементарном уровне.

```
Class Welcome {
/**Программа Welcome to Java! Дж. М. Бишоп, декабрь 1996 г.
 * -------------------------- Java 2 апрель 2000 г.
 * Простая программа, выводящая текстовое сообщение на экран.
 **/
Welcome () {
      System.out.println("Welcome to Java!");
}
public static void main (String [] args) {
new Welcome ();
```

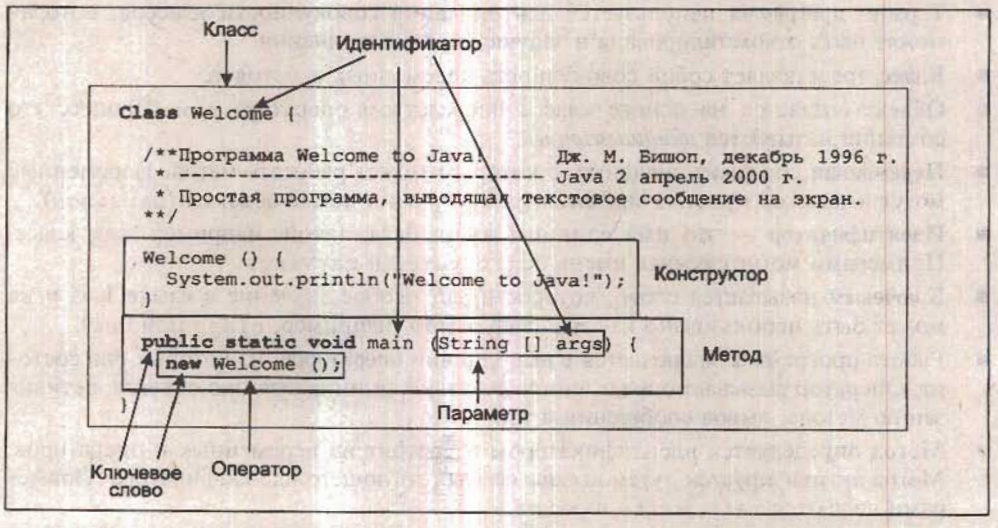


Рис. 2.2. Структура программы

В диаграммах моделирования классы и объекты представлены с помощью прямоугольников, которые разделены на три части — для идентификаторов, переменных и методов. Класс, имеющий описания, обозначается пунктирной линией, а объект, который ссылается на содержимое области памяти, — сплошной линией. Сначала

рассмотрим блок, относящийся к классу: класс называется `Welcome`, не содержит переменных и имеет только один метод — `main`. Справа изображен объект данного класса. Поскольку у него нет собственного идентификатора, ему присваивается идентификатор класса, который предваряется символом ":". Сам объект не содержит переменных, а включает только конструктор, который имеет то же имя, что и класс.

Для демонстрации работы классов на диаграммах моделирования применяются прямоугольники с загнутыми углами (рис. 2.4). В качестве содержимого этих прямоугольников могут выступать Java-команды, сокращенные Java-команды и комментарии.

Синтаксические формы языка Java

Java является языком и, следовательно, имеет свой синтаксис. В книге для описания синтаксиса широко применяются таблицы и примеры. Ниже показано, какой формат надлежит использовать для обычной программы.

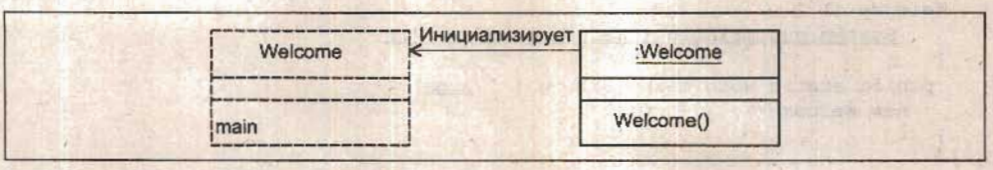


Рис. 2.3. Диаграмма моделирования для программы Welcome



Рис. 2.4. Диаграмма моделирования для программы Welcome с описанием функций

В книге вы неоднократно встретите такие элементы:

Простая программа

```
class Classid {
  Classid () {
    data and control
  }

public static void main (String [ ] argname) {
  new Classid ();
 }
}
```

Поговорим об особенностях данного способа подачи материала. В верхней строке указывается название рассматриваемого элемента языка Java. Далее демонстрируется его синтаксис. Шрифтовые выделения несут такую смысловую нагрузку:

- *Курсивом* выделяются идентификаторы или части программы более высокого уровня, вместо которых впоследствии подставляются соответствующие имена (так, на месте `Classid` может оказаться `CurioStore1`).
- **Жирным начертанием** выделяются ключевые слова языка Java (например, `class` или `new`). Такой способ выделения не используется в основном тексте.
- Обычное начертание назначается идентификаторам и символам, которые должны присутствовать в программе, однако не являются ключевыми словами (`main` или `[]`). Вместо них нельзя подставить другие идентификаторы.

Вернемся к программе `Welcome` из примера 2.1:

```
class Welcome {

   Welcome () {
       System.out.println ("Welcome to Java!");
   }
    public static void main (String [ ] args) {
      new Welcome();
   }
}
```

Мы видим, что в данной программе подстановка была выполнена в трех местах: `Classid` был заменен именем `Welcome`, `data and control` — методом `println`, а `argname` (аргументы метода `main`) — `args`. Для наглядности данные элементы были подчеркнуты. Остальные элементы либо являются постоянными, либо зависят от имени класса.

Семантика языка Java

Синтаксис языка описывает, какую структуру должна иметь программа, какие элементы должны в ней использоваться и в каком порядке. Каждая комбинация объектов имеет свое значение, на основе которого мы определяем, какую операцию выполняет данный фрагмент программы. По этой причине наряду с синтаксисом мы изучим и семантику.

Семантический анализ будет выполняться при рассмотрении синтаксиса отдельных элементов программы:

Простая программа

```
class Classid {
  Classid () {
    data and control
  }

  public static void main (String [ ] argname) {
    new Classid ();
  }
}
```

В Java инициализация программы осуществляется в конструкторе класса `Classid` посредством оператора `new`. Выполнение программы начинается с конструктора и заканчивается по достижении последнего оператора в последовательности.

В подобных табличках значения элементов определяются в краткой форме. Более подробные сведения вы сможете почерпнуть из последующего текста.

Идентификаторы и ключевые слова

Имена в языке программирования Java называются идентификаторами. Идентификатор может включать буквы, символы подчеркивания и цифры, но начинаться он должен с буквы. В идентификаторе нельзя использовать символы пробела. Строчные и прописные литеры считаются в нем разными символами, поэтому `CurioStore1` и `Curiostore1` будут восприниматься как разные идентификаторы. Если идентификатор состоит из нескольких слов, то для облегчения восприятия рекомендуется начинать каждое слово с заглавной буквы.

В программе `Welcome` идентификаторами являются следующие элементы:

```
Welcome     System     out     println
main        String     args
```

Принято соглашение о том, что все идентификаторы, за исключением идентификаторов класса, должны начинаться с маленькой буквы. Благодаря этому мы сможем мгновенно определить, что в предыдущем списке идентификаторы `Welcome`, `System` и `String` относятся к классам.

Ключевые слова — это идентификаторы, зарезервированные для использования языком Java, которые не могут применяться программистом для других целей. Так, мы не можем присвоить программе имя `"class"`, поскольку это слово зарезервировано. В программе `Welcome` употребляются следующие ключевые слова:

```
class      public      static    void      new
```

Ниже приведен полный список ключевых слов языка Java. Слова разделены на три основные категории и упорядочены по алфавиту. Нет необходимости запоминать данный перечень: новые ключевые слова вводятся по мере необходимости, а их краткое описание дается в конце каждой главы.

Таблица 2.1. Ключевые слова Java

Данные	Управление	Объекты
Boolean	break	abstract
byte	case	class
char	catch	extends
double	default	implements
false	do	import
final	else	instanceof
float	finally	interface
int	for	native
long	if	new
short	return	null
static	switch	package
transient	synchronized	private
true	throw	protected
void	throws	public
	try	super
	while	this
		volatile

Символы языка Java

Наряду с идентификаторами и ключевыми словами в Java-программе могут использоваться разнообразные символы. Наиболее часто употребляются следующие:

- **Фигурные скобки** применяются для объединения группы операторов в класс, метод и т. д.
- **Круглые скобки** используются для объединения группы параметров в список. Их применение допускается даже в том случае, если параметры отсутствуют. Примерами могут служить параметры в программе `Welcome`.
- **Точка с запятой** ставится в после каждого оператора.
- **Запятая** предназначена для разделения элементов списка независимо от того, заключены они в скобки или нет.
- **Квадратные скобки** являются отличительным признаком идентификаторов особого вида (более подробное описание будет приведено ниже).

Фигурные скобки определяют в программе границы каждого класса или метода. Внешние элементы относятся к классу, а внутренние — к конструктору и методу `main`. Поскольку операторы отделены друг от друга точкой с запятой, они могут быть расположены в одной строке, однако чаще программисты на языке Java располагают их один под другим. Более подробную информацию об элементах Java-программы вы найдете в разделе 2.4.

Создание и запуск программы

Продолжая развивать тему, затронутую в данной главе, мы можем раскрыть значение каждой строки программы `Welcome`. Нам уже известно, как задавать имя класса. Мы используем также в программе конструктор и метод `main`. Этому методу предшествуют три ключевых слова — `public static void`, которые обязательно должны находиться в этой позиции. Материал о них изложен далее. Здесь достаточно сказать, что, когда виртуальная Java-машина активизируется при выполнении команды языка Java, как описано в главе 1, имя класса должно быть указано. Затем она ищет метод с именем `main` и характеристиками `public static void` и задает запуск программы с этого места. Параметры синтаксиса таковы, что некоторые данные могут быть переданы программе при ее запуске. Данная особенность языка Java рассматривается в главе 4.

Метод `main` создает объект класса программы и вызывает его конструктор. В нашем случае вызывается конструктор `Welcome` и выполняется оператор с именем `println`. Конструктор завершается символом `}` и передает управление в то место программы, из которого был вызван, т.е. методу `main`. Затем заканчивается выполнение этого метода и программа успешно завершает свою работу. На рис. 2.3 показана диаграмма моделирования для программы `Welcome`.

2.3. Основы объектного ориентирования

Классы

Итак, Java-программы состоят из одного или нескольких взаимозависимых классов. Классы являются средствами описания свойств и возможностей реальных объектов, которые указаны в программе. Например, программа `CurioStore1` содержит класс для описания древностей, а программа, предназначенная для создания

изображения флага, — класс для флагов. Мы можем создать множество объектов на основе класса. Кроме того, классы могут образовывать иерархию (см. главу 9).

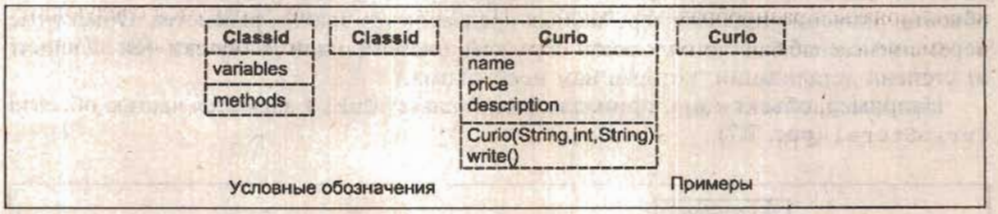


Рис. 2.5. Условные обозначения для класса

Способ представления класса на диаграмме (рис. 2.5) зависит от необходимого уровня детализации. `"id"` — это сокращение английского слова `"identifier"` (идентификатор). Таким образом, `Classid` является обозначением идентификатора класса. Запомните: употребление в начале прописной буквы является обязательным.

В конструкторе `Curio`, который также является методом, объявляются типы параметров, которые необходимы при инициализации того или иного предмета старины.

Объекты

Объект предназначен для точной реализации того, что задано в определении класса. Объекты обычно связаны с существительными, обозначающими такие предметы, как дерево, книга, флаг и т.д. Поэтому они обладают разнообразными свойствами и возможностями. Процесс создания новых объектов для программы заключается в создании нового экземпляра класса. Когда класс инициализируется, описываемые им свойства становятся **переменными** объекта, а возможности — **методами** объекта. В нашем примере объект `Curio` имеет три переменные — `name`, `price` и `description` — и два метода — конструктор и `write`. Переменные и методы класса называются его **элементами**. Пример объекта приведен на рис. 2.6.



Рис. 2.6. Условные обозначения для объекта

Переменные

Для хранения значений переменной резервируется место в памяти компьютера. Значения эти впоследствии могут быть изменены. Например, простые переменные

хранят числовые значения, а объектные переменные ссылаются на область памяти, где хранится объект. Объектные переменные не содержат символов, но выступают в качестве идентификаторов в сегменте переменных соответствующего класса или объекта. Значения простых переменных предваряются знаком равенства. Объектные переменные снабжаются сплошной стрелкой, указывающей на объект. (Все зависит от степени детализации, которая нам необходима.)

Например, объект `mugs`, принадлежащий классу `Curio`, является частью объекта `CurioStore1` (рис. 2.7).

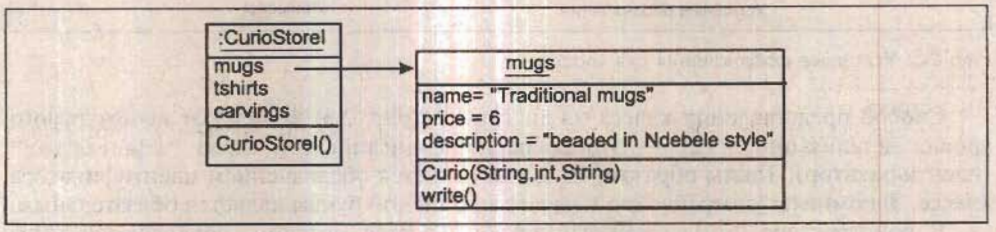


Рис. 2.7. Объект и объектная переменная

На рис. 2.7 показаны шесть переменных. Сможете ли вы указать их все?

Мы подчеркиваем идентификаторы объекта, чтобы показать, что они не существуют в нем самом. Ни один из объектов языка Java не имеет собственного идентификатора: идентификатор относится к переменной. Иногда не известен даже идентификатор объектной переменной, как в том случае, когда переменная создана в методах `main` и применяется обозначение `:Classid`.

Наглядное представление

Часто возникает необходимость представить объект и класс, из которого он создается. Для демонстрации этой зависимости мы использовали пунктир и стрелки, избрав компактную форму представления объектов и классов (рис. 2.8).

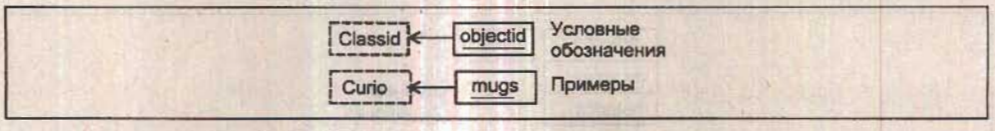


Рис. 2.8. Связь между объектом и его классом

Создание объектов и объявление объектных переменных

Объектная переменная объявляется следующим образом:

Объявление объектной переменной
`Classid objectid;` `Classid objectid1, objectid2, . . . objectid n;`
Объектная переменная объявляется с помощью идентификатора `objectid`, ей присваивается нулевое значение. Внутри класса идентификатор объектной переменной не может дублироваться.

На данном этапе самого объекта еще не существует, поэтому любая попытка получить доступ к какой-либо переменной или функции, которые описаны классом, окажется безуспешной. Например, если мы возьмем содержимое памяти программы в момент вызова конструктора, полученный результат будет таким, как на рис. 2.9: три объектные переменные, каждой из которых присвоено нулевое значение. Создание самого объекта происходит следующим образом:

Инициализация объекта
`objectid = new Classid (parameterlist);`
Новый объект создается для объектной переменной `objectid` в соответствии с описанием класса `Classid`. Конструктор класса `Classid` вызывается с параметрами, данными в списке `parameterlist` (если они существуют). Он может использовать список значений по своему усмотрению (подробная информация дана в конце раздела).

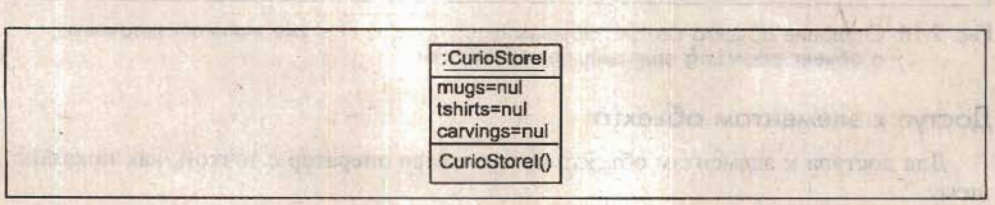


Рис. 2.9. Объявление объектных переменных

В качестве примера снова рассмотрим программу `CurioStore1`. В ней мы объявляем три объектные переменные, четко указывая, что идентификаторы будут использоваться для объектов заданного класса.

```
Curio mugs, tshirts, carvings;
```

Затем следует такая группа операторов:

```
mugs = new Curio("Traditional mugs", 6, "beaded in Ndebele style");
tshirts = new Curio("T-shirts", 30, "sizes M to XL");
carvings = new Curio("Masks", 80, "carved in wood");
```

Здесь каждый объект инициализируется с помощью сочетания "`= new`", которое означает, что нужно создать конкретную реализацию класса, резервируя область памяти для данных, необходимых объекту `Curio`. Для каждого объекта вызывается конструктор `Curio`, вследствие чего начальные значения, определенные в списке параметров, вводятся в объект. На рис. 2.10 смоделирована ситуация, которая имеет место, когда элементы `mugs` и `tshirts` инициализированы как объекты, а объект `carvings` ждет своей обработки. На этой диаграмме не представлены какие-либо классы, т.к. нас интересует только процесс инициализации.

Давайте определим различие между объектной переменной и самим объектом. Когда начинается работа программы, выполняется метод `main`, который создает объект класса `Curiostore1`. Он содержит три объектные переменные, каждая из которых ссылается на объект `Curio`. До того как объект будет создан (путем выполнения операторов "`= new`"), переменная содержит нулевое значение. Таким образом, переменная `carvings` имеет значение `null`, тогда как две другие объектные

переменные ссылаются на созданные объекты, которые содержат начальные значения.

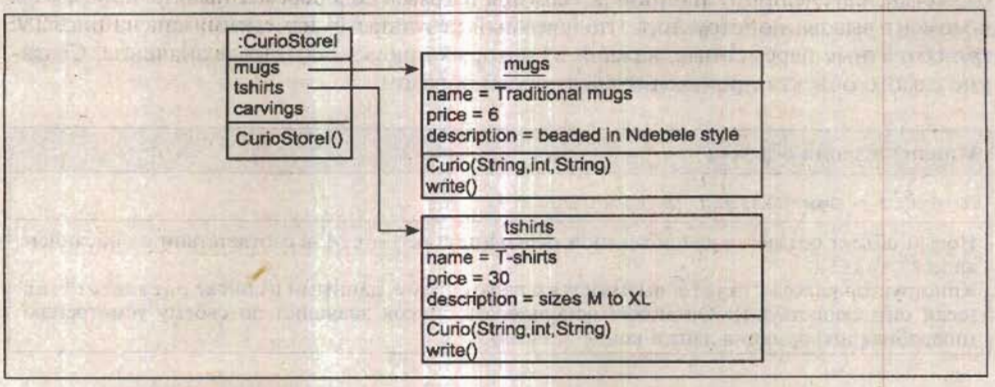


Рис. 2.10. Создание объекта `Curio`; объекты `mugs` и `tshirts` уже инициализированы, а объект `curving` еще ожидает обработки

Доступ к элементам объекта

Для доступа к элементам объекта используется оператор с точкой, как показано ниже:

Доступ к элементам объекта
objectid.variableid *objectid.methodid ()* *objectid.methodid(parameters)*

Вот примеры использования операторов этого вида в программах из раздела 2.1:

```
g.drawString ("Germany", 100,180);

mugs.write();
```

Объектами являются элементы `g` и `mugs`, а методами — `drawString` и `write`. Мы можем также получить доступ к переменным объекта следующим образом:

```
mugs.price
tshirts.description
```

Управление переменными объекта

Давая определение переменным, мы упомянули, что их значения можно изменять. Наиболее простым способом сделать это является выполнение операции присваивания с использованием оператора присваивания, как можно увидеть в конструкторе `Curio`:

```
Curio (String n, int p, String d);
{
   name = n;
   price = p;
```

```
    description = d;
}
```

Здесь значения `n`, `p` и `d` присваиваются и хранятся в переменных `name`, `price` и `description` соответственно. Что происходит во время изменения значения переменной объекта? Переменная может ссылаться на объект, что задается в процессе инициализации. В результате присвоения другого значения она изменится таким образом, что будет ссылаться на другой объект. Если существует намерение сделать это, следует сначала создать переменную объекта с произвольным именем, например:

```
Curio checked;
```

Затем можно взять следующую последовательность операторов, которая в качестве значения последовательно присваивает переменным стоимость каждой древности:

```
checked = tshirts;
checked.write();
checked = carvings;
checked.write();
checked = mugs;
checked.write();
```

Данная последовательность выглядит несколько неестественно, но зато наглядно демонстрирует использование объектных переменных. Первый шаг этого процесса иллюстрирует рис. 2.11. Объекты на диаграмме представлены в виде прямоугольников. Их элементы не указаны. На этом этапе при вызове метода `checked.write()` мы будем ссылаться на метод `write` объекта `tshirts`. При следующем присваивании `checked` будет ссылаться на объект `carvings`.



Рис. 2.11. Управление объектными переменными

Конструкторы и их параметры

Конструктор выполняет три основные функции:

- Если класс содержит метод `main`, он выполняет работу программы (как в примерах 2.1 и 2.3).
- Если в программе используется графика, конструктор отвечает за создание фрейма, определение его размера и т.д.

- Если класс является основой для создания нескольких объектов, конструктор применяется для копирования начальных значений в объектные переменные (как в случае класса `Curio` в примере 2.3).

Для понимания третьей функции необходимо получить представление о параметрах. Рассмотрим начало класса `Curio`:

```
class Curio {
//Объявляет переменные, описывающие древность
String name;
int price;
String description;

Curio (String n, int p, String d) {
name = n;
price = p;
description = d;
}
... and so on
```

Здесь в классе `Curio` объявлены три переменные, а конструктор содержит список соответствующих параметров — две строки и одно целое число, которое в языке Java имеет тип `int`. В следующей главе мы рассмотрим данные этого типа подробнее, а пока отметим лишь, что они являются числами и могут быть скопированы в переменные того же типа с помощью операции присваивания. Таким образом, при создании нового объекта

```
mugs = new Curio("Traditional mugs", 6 "beaded in Ndebele style");
```

Строка `"Traditional mugs"` будет соответствовать параметру `n` и будет скопирована в переменную `name`, `6` будет соответствовать параметру `p` и будет скопировано в переменную `price`, а `"beaded in Ndebele style"` будет соответствовать параметру `d` и будет скопирована в переменную `description`. Java не позволит перепутать параметры: должно соблюдаться соответствие типов, т. е. строки должны соответствовать строкам, а целые числа — целым числам.

2.4. Знакомимся с методами вывода

Мы рассмотрели целую серию примеров, иллюстрирующих структуру Java-программы. Теперь мы можем перейти к написанию простой программы, которая отображает что-либо на экране. В отличие от многих других языков программирования, в Java не существует оператора для вывода данных. Вывод осуществляется с помощью специальных методов классов. Таким образом, чтобы вывести какую-либо информацию, нужно знать, какой метод следует вызвать.

Необходимые нам методы называются `println` и `print`. Они находятся в специальном встроенном классе `PrintStream`. Внутри общедоступного класса `System` уже существует объект `PrintStream` с именем `out`, который связан с экраном вашего компьютера. Итак, мы имеем три элемента:

- класс `System`;
- объект `out`;
- методы `print` и `println`.

Из них можно скомпоновать операторы, которые вызывают метод для вывода информации на экран. По причине частых обращений к данному методу мы будем называть их для краткости операторами вывода.

Вот наиболее простая форма операторов языка Java, с помощью которых фрагмент текста отображается на экране:

Операторы вывода

```
System.out.println (string);
System.out.println (string1 + string2 + . . . + stringn);
System.out.println ();
System.out.print (string);
System.out.print (string1 + string2 + . . . + stringn);
```

Метод `Println` выводит строку или список строк и заканчивает вывод с переводом курсора на новую строку.
Метод `Print` не завершает строку вывода.
Метод `Println`, указанный без параметров, завершает строку вывода.

Как видите, Java выполняет вывод только строк. Но если числа преобразовать в строки, их также можно будет вывести на экран. В случае необходимости объединить несколько строк в одну надлежит использовать знак "+". (К этому мы еще вернемся.)

Вывод строк

Символьной строкой называется произвольная последовательность символов, заключенная в кавычки, например:

```
"sizes M to XL"
"Licence No.CFD678GT"
"€ 5.95 per kg"
```

В книге употребляется краткий вариант данного термина — строка. Если кавычка является частью строки, ее необходимо предварить знаком \. Обратная косая не является элементом строки, а служит указанием, что следующий символ принадлежит строке. Например, запись

```
"He said \ "No\""
```

представляет собою строку "He said "No"". В результате вызова метода `println` отображается строка, указанная в качестве параметра, и завершается строка печати (часть "`-ln`" в операторе `println` — это сокращение от слова `line`, строка). Таким образом, выполнение операторов

```
System.out.println("€5.95 per kg");
System.out.println("He said \ "No\"");
```

приведет к выводу на экран следующих строк:

```
€5.95 per kg
He said "No"
```

(символ € — это обозначение денежной единицы *евро*, общеевропейской валюты).

Метод `print` работает аналогично методу `println` с той разницей, что он не переводит курсор в новую строку, поэтому следующий оператор `print` *или* `println` продолжает вывод информации с той позиции, где завершил печать предыдущий. Для метода `println` не обязятельно указывать строку в качестве параметра. Если она не

указана, курсор переводится в другую строку. Это свойство также может быть использовано, если при выводе данных требуется ввести пустую строку, например:

```
System.out.println("€5.95 per kg");
System.out.println();
System.out.println("He said \ "No\"");
```

В результате на экран будут выведены следующие данные:

```
€5.95 per kg

He said "No"
```

Обратите внимание на то, что, хотя у метода языка Java могут отсутствовать параметры (элементы в скобках), сами скобки ставятся в любом случае. Для создания нескольких пустых строк надлежит вызвать метод `println` несколько раз подряд:

```
System.out.println();
System.out.println();
System.out.println();
```

Кроме того, мы можем использовать другой специальный символ, `\n`, который позволяет перевести курсор в другую строку, как показано в следующем примере:

```
System.out.println ("\n\n\n");
```

Пример 2.4. Вывод предупреждения

Задача. Выведите на экран предупреждение о том, что компьютер может быть заражен вирусом*.

Программа. Эта программа очень проста и аналогична той, что была рассмотрена в примере 2.1. Она содержит класс, конструктор и метод `main`, за которым следует ряд операторов вывода, в результате выполнения которых отображается окно, показанное ниже.

```
class DisplayWarning {
   /*Программа вывода на экран предупреждения  Дж.М. Бишоп.  август 1996 г.
    * ---------------------------------------  Java 1.1 октябрь 1997 г.
    *                                          обновлена - апрель 2000 г.
    *                                          Java 2
    * Иллюстрирует использование метода println.
    */

DisplayWarning () {
    System.out.println("----------------------------");
    System.out.println("|                          |");
    System.out.println("|       W A R N I N G      |");
    System.out.println("| Possible virus detected  |");
    System.out.println("|   Reboot and run virus   |");
    System.out.println("|      remover software    |");
    System.out.println("|                          |");
    System.out.println("----------------------------");
 }
public static void main (String [] args) {
   new DisplayWarning ();
 }
}
```

* Вирус — это программа, которая может инфицировать ваш компьютер, повредить данные на жестком диске и т.д. Создаваемая программа не может обнаруживть вирус, а только выводит сообщение о его наличии.

Тестирование. В результате выполнения программы на экране отобразилось следующее сообщение:

```
---------------------------
|                         |
|       W A R N I N G     |
| Possible virus detected |
|   Reboot and run virus  |
|      remover software   |
|                         |
---------------------------
```

Конкатенация

В завершение разговора о строках и выводе данных на экран мы рассмотрим, что произойдет, если строка программы окажется длиннее экранной строки. Мы не можем перейти в следующую строку, нам необходимо корректно завершить одну строку и начать новую, соединив их с помощью оператора +. Эта операция называется *конкатенацией*. Ниже показано, как использовать в программе оператор конкатенации:

```
System.out.println ("Your tree is not known to us. Would " +
          "you like to submit details of by WWW or " +
          "courier?");
```

Мы еще не раз столкнемся с длинными строками при анализе программ из практической части книги.

Пример 2.5. Прейскурант лавки древностей

Задача. Перечень товаров, предлагаемых в лавке древностей, необходимо представить в виде таблицы и отсортировать в определенном порядке.

Решение. Вместо того чтобы использовать метод `write`, мы будем для каждого объекта осуществлять доступ к каждому полю в отдельности, используя запись с точкой в качестве разделителя. Такой оператор, как

```
mugs.write();
```

будет заменен на

```
System.out.println(mugs.item+"\t"+mugs.description+"\t"+mugs.price);
```

где ключ `\t` задает перевод курсора в следующую ячейку таблицы.

```
T-shirts    sizes M to XL     30
```

Если применить эти операторы к двум другим древностям одновременно, то при выводе мы получим следующее:

```
The Polelo Curio Store
======================

Items               Descriptions                Price per item
-------------------------------------------------------------
Traditional mugs    beaded in Ndebele style     6
T-shirts            sizes M to XL               30
Masks               carved in wood              80
```

Примечания:

- Длинная строка символов "=" может быть разбита на две строки, хотя должна быть разделена на две части и объединена оператором конкатенации.
- Размер шага табуляции при выводе может быть разным: на нашей машине он составлял четыре символа, так что нам потребовалось подгонять число шагов для каждого объекта, чтобы расположить все значения одно под другим.

Формат

Вид, который имеет написанная программа с учетом строк и пробелов, не влияет на данные, получаемые в результате ее выполнения. Когда работа программы завершена, на экран выводится только то, что заключено в кавычки. Между строками на экране не будет пробелов, если в программе они разделены на части, объединенные оператором +. На вывод данных не влияют также пустые строки и комментарии. Таким образом, при выполнении следующих операторов информация, выведенная на экран, будет иметь тот же вид, что и в предыдущем примере:

```
System.out.println("€5.95 " + "per kg");

    System.out.println();
    System.out.println("He " +
"said \"No\"");
```

Инструкции, которые передаются компьютеру, не должны иметь какой-либо особый формат. Обычно их пишут одну под другой. Для выделения группы операторов можно создать отступы от левого края, величиной в несколько символов, но это не является правилом, которому нужно беспрекословно следовать. Важно запомнить такие моменты, касающиеся формата программы:

- Одна строка может содержать один или несколько операторов.
- Операторы могут быть разбиты на несколько строк.
- В конце операторов ставятся точки с запятой.

2.5. Простые вычисления

В этом разделе мы рассмотрим, как выполнять простые вычисления на языке Java и как присваивать результирующие значения переменным. В следующих главах описываются другие виды вычислений.

Типы данных

Мы уже обсуждали, как объекты хранят свои данные в переменных. В языке Java для каждой переменной характерен свой тип данных, и только те переменные, которые имеют одинаковые типы, могут использоваться вместе. Другими словами, мы не можем смешивать строки и числа. Переменные делятся на объектные (например, `mug`) и базовые. В языке Java существует восемь встроенных типов данных, которые рассматриваются в главе 3. В настоящей главе описан только один, предназначенный для определения целых чисел и называемый в данном языке `int`.

Объявление переменной

Переменные могут быть объявлены в любом месте класса или метода языка Java, а также внутри любой группы операторов, заключенной в фигурные скобки. Такую группу операторов называют **блоком**. О правилах использования блоков мы поговорим позже. Обычно объявление располагают в начале или в конце блока:

Простейшее объявление переменной
type id; *type id1, id2, … idn ;* *type id = value;*
Объявляет переменную согласно типу и присваивает ей идентификатор. В третьей строке переменной присваивается начальное значение. Если оно не задано, то присваивается то значение, которое определено по умолчанию (например, 0 для чисел). Идентификатор не может дублироваться внутри блока операторов в фигурных скобках {}.

Объявление позволяет инициализировать одну или несколько переменных данного типа. Предложенный выше способ может быть использован для инициализации переменной одновременно с ее объявлением. Ниже приведены примеры объявления переменных типа `int`:

```
int temperature;              //in degrees Celsius
int oldWeight, newWeight;     //in kilograms
int price;                    //in graz
int maxMark = 100;
```

Напомним, что идентификаторы не должны дублироваться внутри блока.

Объявления должны быть четкими и аккуратными, идентификаторы и типы необходимо размещать в одной строке. Желательно выработать привычку указывать, для чего предназначены переменные, если это не следует из их идентификаторов, а также вводить данные о единицах измерения, как в первых трех примерах.

Когда переменная объявлена, ей присваивается значение по умолчанию, если только пользователь не установил для нее значение самостоятельно. Для числовых типов таким значением является ноль. Вы можете смело использовать его, хотя многие программисты предпочитают задавать значения для переменных явно.

Выражения

Выражение — это термин, который в языках программирования употребляется для обозначения формул. Выражение состоит из операндов — чисел или переменных — и операторов, которыми для типа `int`, например, являются скобки, знаки умножения, деления, сложения и вычитания. В языках программирования способ написания выражений несколько отличается от общепринятого. Основные отличия заключаются в следующем:

- знаком умножения является звездочка (*);
- деление обозначается косой чертой (/);
- знаменатель следует за числителем и пишется в той же строке.

Следствием записи в одну строку является то, что в Java-выражениях используется больше скобок, чем в математических выражениях. Выражение деления, которое в обычном случае было бы представлено в виде дроби, должно быть разбито косой

чертой на две части, каждая из которых заключена в скобки. В качестве примера запишем несколько простых формул в соответствии с правилами языка Java.

Формула	Выражения на языке Java
$\frac{euros}{rate}$	`euros/rate`
$\frac{30test + 20project + 50exam}{100}$	`(30*test+20*project+50*exam)/100`

Важен тот факт, что в Java оператор / употребляется и с целыми, и с вещественными числами. Если оба значения целочисленные, выполняется целочисленное деление, иначе — вещественное. Хотя первый вид деления не всегда подходит (например, при делении 1 на 2 мы в результате получим 0), он активно используется, что отражено в примере 2.6. Вещественное деление рассматривается в разделе 3.1.

Наряду с выполнением пяти операций, производимых посредством четырех указанных операторов (оператор / выполняет две функции), часто требуется найти остаток от деления целых чисел. Для этого применяется оператор деления по модулю — %. Рассмотрим его использование на примере:

```
23 % 2     дает  1
6 % 6      дает  0
81 % 11    дает  4
```

В Java, как и в других современных языках, предусмотрены операторы, которые служат для прибавления или вычитания единицы из целого числа. Это операторы ++ и --. Они полезны при выполнении операций присваивания, которым посвящен следующий раздел.

Присваивание

Операция **присваивания** выполняется после объявления переменной с тем, чтобы присвоить ей значение выражения:

Оператор присваивания

```
variable = expression;
variable ++;
variable --:
variable op = expression;
```

Значения, полученные в результате вычисления выражения, присваиваются переменной. Этот оператор применим к переменным всех видов — как к простым, так и к объектным.
Операторы ++ и -- увеличивают и уменьшают на единицу значение переменной (применяются только по отношению к числам).
В выражении `op=` оператор применяется к существующему значению переменной и выражению. Полученный результат присваивается данной переменной. Этот вид записи подходит только для числовых переменных.

Функцией оператора присваивания является вычисление значения выражения и присвоение его переменной, указанной идентификатором. Как надлежит читать выражения, содержащие знак равенства? Символу операции присваивания соответствует слово "равняется". Таким образом, выражение

```
temperature = 24;
```

читается как "температура равняется 24". Значение выражения — 24, именное оно присваивается переменной "temperature". В конструкции

```
oldWeight = 65;
newWeight = oldWeight - 5;
```

переменной oldWeight присваивается значение 65, которое затем используется для вычисления значения переменной newWeight (равно 60).

Необходимое условие выполнения операции присваивания — совпадение типов выражения и переменной. Поэтому мы не можем присвоить объекту Curio значение, имеющее тип int. Эта операция также может использоваться для создания нового экземпляра объекта, в чем можно убедиться при рассмотрении следующего оператора:

```
mugs = new Curios("Traditional mugs", 6, "beaded in Ndebele style");
```

В данном выражении генерируется значение, которое ссылается на новый объект и присваивается объектной переменной mugs.

Обратимся к распространенной операции прибавления единицы к значению счетчика. Решить данную задачу позволяют следующие выражения:

```
in total;
total = total + 1;
```

Поскольку такой вид присваивания используется довольно часто, целесообразно сэкономить время, которое ушло бы, если бы мы каждый раз набирали имя переменной дважды. По этой причине в языке Java предусмотрены специальные виды присваивания для сокращенной записи инкремента и декремента: i++ или i+=1. Их следует интерпретировать так: "Прибавить 1 к переменной i". Выражение i+=2 означает "прибавить 2 к переменной i" и т. д. Таким образом, рассмотренное выше выражение примет вид:

```
total++;
```

Если каждый раз прибавляется 5, выражение будет следующим:

```
total +=5;
```

Чтобы запомнить, что оператор инкремента имеет вид +=, а не =+, следует понять, что во втором случае программа может расценить выражение как присваивание положительного значения, а не его прибавление. Так, в выражении

```
total =+5;
```

значение +5 будет воспринято как константа и присвоено переменной total.

Краткие формы предусмотрены и для других операторов, таких как вычитание и умножение. Например, если каждый раз приходится удваивать значение переменной, можно прибегнуть к следующей конструкции:

```
int power;
power *= 2;
```

Приоритет операций

В языке Java применяются обычные правила приоритета, согласно которым в первую очередь вычисляются значения выражений в скобках, а операции умножения и деления производятся прежде, чем сложения и вычитания. Ниже показаны группы

приоритета для арифметических операторов (вычисление в пределах группы выполняется слева направо):

Последовательность выполнения операций (арифметических)
group 0; ()
group 1; ++ -- + (unary) - (unary)
group 2; * / %
group 3; + - + (concatenation)
Группа операторов в скобках имеет самый высокий приоритет. Приоритет операторов внутри группы увеличивается слева направо.

Правила приоритета в языке Java (которые во многом подобны правилам языка C) таковы, что вам не придется часто использовать круглые скобки для того, чтобы сделать выражение более легким для понимания.

Вывод выражений

В примерах мы обращались к методам `println` и `print` для вывода строк на экран. В Java можно также выводить числа и, конечно, результаты вычислений. Например, чтобы отобразить на экране результирующее значение выражения "4 умножить на 5 минус 2", необходимо ввести:

```
System.out.println (4 * 5 - 2);
```

В результате на экране отобразится число

```
18
```

Идентичным образом мы можем вывести значения выражений, содержащих переменные. Например, вследствие применения операторов

```
int hoursPerDay = 24;
System.out.println (7 * hoursPerDay);
```

на экране отобразится значение

```
168
```

Выводимое число может содержать столько цифр, сколько необходимо. Если требуется особое форматирование, следует задействовать дополнительные утилиты языка Java, обзор которых дан в главах 4 и 7.

Метод toString

Как следует поступить в случае, когда необходимо вывести более одного значения или цифры вместе со строками? В предыдущем разделе указывалось, что для вывода нескольких строк надлежит применять операцию конкатенации. Такой же прием следует использовать и для решения данной задачи. Однако сейчас мы выполним более глубокий анализ происходящего.

Оператор конкатенации объединяет строки. Обнаружив что-либо иное слева или справа (например, число), он обращается к методу `toString`, определенному классом, который может преобразовать это значение в строку. Все встроенные типы имеют соответствующие методы `toString`, а классы, такие как `Curio`, могут с

легкостью определять их. Оператор + выполнит конкатенацию, после чего полученная строка будет выведена на экран. Например, выражение

```
System.out.println (7 + "days is"
                    + (7 + hoursPerDay) + "hours");
```

позволяет вывести на экран такую запись:

```
7 days is 168 hours
```

Число и выражение были преобразованы в строки, после чего в результате выполнения операции объединения мы получили полное сообщение. Необходимо включать в строку, примыкающую к числу, пробелы, поскольку они не вставляются автоматически.

Пример 2.6. Расписание маршрутов автопарка

Задача. Компания Savanna Deliveries Inc. закупила новые транспортные средства, которые способны развивать более высокую скорость, чем старые. Сотрудники компании хотят узнать, как в этой связи следует изменить расписание маршрутов.

Решение. Мы будем использовать 24-часовую временную шкалу и такие значения, как 0930 и 1755. Нам необходимо определить, как изменится время прибытия с учетом того, что средняя продолжительность поездки сократилась на 15%. Для этого надлежит найти разницу между временем прибытия и отправления, а затем, учитывая сокращение времени поездки, умножить ее на 0,85. Вопрос состоит в том, как выполнять арифметические операции над значениями времени.

Алгоритм. Значения времени состоят из двух частей, представляющих часы и минуты. Чтобы произвести над ними операции вычитания и умножения, мы должны перевести их в минуты, выполнить вычисления, а затем осуществить обратное преобразование. Последовательность действий такова:

- Задайте время отправления и прибытия.
- Вычислите продолжительность поездки в минутах.
- Уменьшите ее на 15%.
- Прибавьте новое значение продолжительности поездки к времени отправления.
- Выведите на экран новые значения продолжительности поездки и времени прибытия.

Во время первого преобразования целое число разделяется на две части. Чтобы выполнить это, мы должны разделить число на 100, вследствие чего получим количество часов, а затем, вычислив разность, найти число минут. Рассмотрим пример:

Преобразование в минуты

Значение времени	0715
Находим количество часов, разделив число на 100	7
Определяем количество минут, взяв остаток	15
Переводим значение в минуты, умножив количество часов на 60 и прибавив количество минут	435

Обратное преобразование выполняется аналогичным образом. Его алгоритм следующий:

Обратное преобразование в часы и минуты

Значение времени в минутах	435
Находим количество часов, разделив значение в минутах на 60	7
Определяем остаток от предыдущего деления, чтобы вычислить число минут	15
Вычисляем значение времени, умножив количество часов на 100 и прибавив число минут	715

Программа. Программа обработки информации об одной поездке довольно проста. Время отправления и прибытия конкретного транспортного средства инициализируется посредством переменных. В главе 4 мы увидим, как вводить их в программу. Помните, что деление целых чисел дает в результате целое число, поэтому алгоритмы, приведенные выше, можно сразу записать на языке Java.

```
class FleetCalculator {
/* Программа обработки информации о поездке  Дж. М. Бишоп, август 1996 г.
 * ----------------------------------------  Java 1.1 октябрь 1997 г.
 *                                           обновление - апрель 2000, Java 2
 * Определяет новое время прибытия для транспортных средств,
 * которые совершают поездку на 15% быстрее.
 *
 * Иллюстрирует применение выражений и операции присваивания.
 * Включает оператор деления по модулю %.
 */
FleetCalculator () {

    int reduction = 85;
    int depart24 = 900;
    int arrive24 = 1020;

   int newArrive24,   //показатель времени для 24-часового цикла
       departMins,    //минут в день
       arriveMins,
       oldJourneyMins,
       newJourneyMins,
       newArriveMins;

//Преобразование начального значения времени в минуты.
  departMins = depart24 / 100 * 60 + depart24 % 100;
  arriveMins = arrive24 / 100 * 60 + arrive24 % 100;

//Вычисление продолжительности поездки до и после
//приобретения новых транспортных средств.
  oldJourneyMins = arriveMins - departMins;
  newJourneyMins = oldJourneyMins * reduction / 100;

//Вычисляет новое время прибытия в минутах, а затем
//преобразует его в соответствии с 24-часовой шкалой.
  newArriveMins = departMins + newJourneyMins;
  newArrive24 = newArriveMins / 60 * 100 + newArriveMins % 60;
```

```
//Вывод результатов на экран.
  System.out.println("Departure time is "+depart24);
  System.out.println("Old arrival time is "+arrival24);
  System.out.println("Old Journey time is "
                     +oldJourneyMins+"minutes");
  System.out.println("New journey time is "
                     +newJourneyMins+" minutes");
  System.out.println("New arrival time is "+newArrive24);
 }

 public static void main (String[] args) {
   new FleetCalculator ();
 }
}
```

Обратите внимание на то, что мы используем шесть разных переменных, чтобы хранить в памяти сразу все значения.

Тестирование. Чтобы протестировать программу, мы подбираем подходящие значения времени отправления и прибытия и задаем их в программе. Выполнение программы даст следующие результаты:

```
Departure time is 900
Old arrival time is 1020
Old journey time is 80 minutes
New journey time is 68 minutes
New arrival time is 1008
```

Более подробно о тестировании

Это наша первая программа, в которой результаты вычислений могут различаться от запуска к запуску в зависимости от исходных значений. Поэтому мы должны убедиться, что все запланированные вычисления будут выполняться корректно для всех возможных значений. Конечно, проверка каждого набора данных займет очень много времени при использовании тех немногочисленных средств языка Java, которые мы успели изучить, но можно выбрать такие наборы данных, которые охватывают все возможные варианты, и протестировать программу с их использованием.

Ошибки в программе обычно возникают при вводе граничных значений, таких как ноль или последнее возможное значение. Также следует проверить, как будет вести себя программа при вводе "неожиданного" значения, например, если присвоить отрицательное число переменной, которая соответствует процентному соотношению, или сделать время прибытия меньше времени отправления. Нам нужно подобрать такой набор тестовых значений, чтобы мы могли изменять только одно значение, оставляя другие неизменными, после чего задать новые "неизменные" параметры и продолжить тестирование. Ниже показан выбранный нами набор данных для проверки:

процент	отправление	прибытие	
0	900	1020	
100	900	1020	
115	900	1020	проверяет, что получится, если новые машины будут медленнее
50	900	1020	

процент	отправление	прибытие	
1	900	1020	проверяет, что получится при малых процентных соотношениях
85	0	1020	
85	1019	1020	проверяет, что получится при небольших значениях времени поездки
85	1100	1020	проверяет, что получится, если ввести неправильное время отправления
85	2400	1020	проверяет, что получится при использовании последнего возможного значения
85	900	2359	

Этот перечень значений не является исчерпывающим, но дает представление о том, что же следует проверять. Конечно, изменять программу каждый раз при проверке новых значений — очень утомительное занятие. Вскоре вы узнаете, как написать программу так, чтобы она оставалась неизменной, а новые значения вводились вручную для последующего присвоения переменным.

Резюме

Во время прочтения данной главы мы изучили классы и объекты, выяснили, как они определяются и инициализируются, а также рассмотрели некоторые дополнительные элементы, такие как конструкторы. В качестве примеров применялись три программы, иллюстрирующие использование на практике базовых элементов, необходимых при объектно-ориентированном программировании: классов, объектов, конструкторов, методов, параметров и операторов. Работа с графическими средствами языка Java частично продемонстрирована в программе `FlagMaker1`. В главе изложены также сведения о доступе к элементам объекта и присваивании значений объектным переменным.

Значительное внимание уделено синтаксису для идентификаторов, ключевых слов, а также присваивания и вывода данных. Завершают главу базовые данные о принципах использования выражений и операторов присваивания для проведения элементарных вычислений.

Мы пришли к заключению, что конкатенация является наиболее полезным средством объединения нескольких значений для одновременного вывода на экран. В этой главе мы определили простые типы данных `int` и `String`. Это позволило нам написать несколько новых программ.

Рассмотрены следующие ключевые слова:

```
class     new
int       null
```

Ниже представлен список употреблявшихся ключевых слов, значение которых более полно будет раскрыто позже:

```
import    void
public    extends
static
```

Широко использовались следующие классы, объекты и методы:

```
System.out.println    String
System.out.print      toString
```

Графическое изображение создавалось с помощью таких средств (программа `FlagMaker1`):

```
java.awt            System.exit
java.awt.event      setTitle
Frame               setSyze
add                 setVisible
addWindowListener   g.setColor
WindowAdapter       g.fillRect
WindowClosing       g.drawString
WindowEvent         Color. … a colour name
```

Подробные описания этих средств вы найдете в главах 10 и 11.

Вопросы синтаксиса и семантики обсуждались для:

- простой программы;
- объявления объектной переменной;
- инициализации объекта;
- доступа к элементам объекта;
- операторов вывода;
- объявления простых переменных;
- оператора присваивания;
- последовательности выполнения операторов (арифметических).

Диаграммы моделирования были разработаны для:

- класса;
- объекта:
- переменной объекта;
- инициализации.

Вопросы



2.1. При запуске программы `Welcome` первым оператором, который должен быть выполнен, является:

а) конструктор

б) `System.out.println("Welcome to Java");`

в) `new Welcome();`

г) метод `main`

2.2. Чтобы изменить цвет рисунка и сделать его зеленым, мы используем:

а) `setcolour(green)`

б) `setcolor(green)`

в) `g.setColor(Color.green)`

г) `setColor(Color.green)`

2.3. Комментарий, который начинается символом `//`, должен заканчиваться так:

а) `//`

б) символ конца строки

в) `*/`

г) `}`

2.4. Какая последовательность может использоваться в качестве идентификатора?

а) `out`, `println` и `args`

б) только `out` и `println`

в) только `println` и `args`

г) ни одна из них

2.5. Какое из приведенных ниже выражений позволяет вставить новую строку между двумя значениями, выводимыми на экран оператором `println`?

а) `+ "n" +`

б) `newLine()`

в) `"\n"`

г) `+ "\n" +`

2.6. Какое выражение нужно использовать, чтобы прибавить 10 к значению переменной `points`?

а) `points + 10`

б) `points += 10`

в) `points =+ 10`

г) `points + +10`

2.7. Конструктор предназначен для:

а) создания пространства для объектов

б) создания пространства и инициализации объекта

в) инициализации объекта

г) запуска программы

2.8. Попытка доступа к элементу `Curio.name` приведет к возникновению ошибки компиляции, потому что

а) элемент `name` не определен в объекте `Curio`

б) доступ к `name` должен осуществляться через объект `Curio`

в) после элемента `name` должны идти круглые скобки

г) `Curio` следует писать с маленькой буквы

2.9. Если переменная `newArriveMins` имеет значение 310, каким будет результат использования формулы из примера 2.6?

а) 510

б) 610

в) 505

г) 516

2.10. Для класса `Curio` инициализация

```
Baskets = new Curios ("Masks", 30, 80);
```

приведет к ошибке компиляции, потому что:

а) первым параметром должен быть `"Baskets"`

б) `Baskets` начинается с большой буквы

в) `Bask ets` не является классом

г) параметры не соответствуют классу `Curio`

Задачи

2.1. **Диаграмма флага.** Создайте диаграмму моделирования для программы `FlagMaker1` (пример 2.2).

2.2. **Олимпийские кольца.** Необходимо вывести на экран цветное изображение олимпийского логотипа в виде пяти пересекающихся колец. Используйте класс `Ring`, который позволяет задать цвет и позицию кольца в качестве параметров конструктора, а затем напишите другой класс, создающий пять объектов `Ring`, чтобы завершить логотип. Используйте программу `FlagMaker1` в качестве модели. (Кольца должны быть синего, желтого, черного, зеленого и красного цветов, причем желтое и зеленое находятся внизу).

2.3. **Прейскурант лавки древностей.** Модифицируйте программу `CurioStore` (пример 2.3) так, чтобы она, во-первых, сохраняла данные о количестве каждой древности, а во-вторых, выводила сведения о каждой древности в таком порядке:

```
25 Traditional mugs beaded in Ndebele style for G6
```

(Программа контроля продаж рассматривается в примере 3.5)

2.4. **Флаги с вертикальными полосами.** В программе `FlagMaker1` измените функцию `paint` так, чтобы она рисовала флаги с вертикальными полосами. Попробуйте изобразить флаги Италии, Франции или Мали (зеленый, желтый, красный).

2.5. **Среднее значение.** Курс компьютерных наук предусматривает тестирование, практическое задание и экзамен — 20%, 30% и 50% соответственно. Напишите программу, которая выставляет оценки на основе полученных результатов по 100-балльной шкале, и вычислите окончательную оценку с учетом указанных процентных соотношений.

2.6. **Расход топлива.** Расход топлива составляет 8 литров на каждые 100 км при езде по ровной дороге и на 15% больше по ухабистой. Напишите программу, которая отображает на экране информацию о том, какое расстояние может пройти автомобиль с полным 40-литровым баком по нормальной и по ухабистой дороге.

2.7. Рабочие смены на фабрике. На фабрике 24-часовой день разделен на три смены следующим образом:

1-я смена	с 0000 до 0759
2-я смена	с 0800 до 1559
3-я смена	с 1600 до 2359

Напишите программу, которая вычисляет, сколько времени осталось работать текущей смене, а также выводит на экран номер этой смены. (Подсказка: используйте операторы % и /, как в примере 2.6.)

2.8. **Детский сад.** В детском саду о каждом ребенке фиксируется такая информация: имя, фамилия, название группы (младшая, средняя, старшая или подготовительная), а также год, когда ребенок пойдет в школу (через один-четыре года, начиная с текущего). Создайте класс `Child` (ребенок) для хранения этой информации и протестируйте его с помощью программы, которая инициализирует шесть объектов этого класса, содержащих данные о трех различных детях, и вызывает метод `write` для вывода их содержимого. Переменная, содержащая информацию о годе, — это целое число, остальные — строки. Метод `write` также должен вычислять, через сколько лет ребенок пойдет в школу.

2.9. **Диаграмма программы "Детский сад".** Создайте диаграмму моделирования для программы "Детский сад", которая соответствует ситуации после создания двух первых объектов, хранящих информацию о детях.

2.10. **Книга.** Мы разработаем программу, которая будет сохранять информацию об учебнике. Начнем с определения класса `Chapter` (глава), который содержит переменные для названия главы и количества страниц в ней. Используя данные о настоящей книге в качестве тестовых, инициализируем объекты для первых четырех глав. Впоследствии мы будем работать с этими объектами, вычисляя общее количество страниц в книге и среднее количество страниц в главе. Страница, написанная вручную, содержит примерно 80% информации печатной страницы. С помощью наших четырех объектов вычислим количество рукописных страниц для каждой главы. Например, если первая глава содержит 20 страниц, то рукописная будет содержать 20/0.8=25 страниц, поэтому вторая глава начнется со страницы 26, а не 21.

2.11. **Диаграмма программы "Книга".** Создайте диаграмму моделирования для программы `Книга`, которая соответствует ситуации после создания двух первых объектов, хранящих информацию о главах.

2.12. **Автобус, осуществляющий внутренние перевозки.** Автобус курсирует между двумя терминалами аэропорта, отправляясь через каждые полчаса. Напишите программу, которая на основе заданного времени путем инициализации соответствующей переменной вычисляет, сколько времени осталось до очередного отправления автобуса.

ГЛАВА 3



Типы и методы

3.1. Поговорим о типах

В разделе 2.4 мы рассмотрели целые числа, а сейчас перейдем к изучению других типов данных. В языке Java насчитывается шесть типов данных для определения чисел — `int`, `long`, `double`, `byte`, `short`, `float` — и два специальных типа — `boolean` и `char`. Анализируя численные типы данных, мы поднимали вопрос о том, как преобразовывать один тип в другой, чтобы переменной можно было присваивать значения разных типов.

В продолжение разговора рассмотрим свойства данных целочисленных типов (табл. 3.1). Тип `byte` обычно используется в программах, которые предназначены для обработки данных на машинном уровне. Благодаря типу `short` можно зарезервировать место в памяти, однако в Java он не используется широко. Для объявления целых чисел обычно используется тип `int`, хотя часто применяется и тип `long`, поскольку он по умолчанию устанавливается для результатов целочисленных выражений. Для объявления вещественных чисел естественным был бы выбор типа `float`, хотя по умолчанию результат выражения с вещественными числами представляется с использованием типа данных `double`. Таким образом, в программах чаще всего применяются типы данных `int`, `long` и `double`.

Таблица 3.1. Описание целочисленных типов

Тип	Форма представления	Значение по умолчанию	Длина	Максимальное значение
`byte`	Целое число со знаком	0	8 бит	127
`short`	Целое число со знаком	0	16 бит	32767
`int`	Целое число со знаком	0	32 бит	2147483647
`long`	Целое число со знаком	0	64 бит	порядка 10^{18}
`float`	Число с плавающей точкой	0	32 бит	порядка 10^{38}
`double`	Число с плавающей точкой	0	64 бит	порядка 10^{308}

Со значениями первых четырех численных типов можно выполнять все вышеупомянутые операции. Как осуществляется их вывод на экран, рассказывается в разделе 2.4. `Float` и `double` являются вещественными типами данных. Для них характерны специфические свойства, по отношению к ним можно выполнять специфические операции и, кроме того, они имеют собственный формат вывода.

Вещественное число записывается с использованием десятичной точки, а не запятой, например:

```
3.1415962
-80.5
0.000007
```

Если значение становится очень большим, его можно записать в логарифмическом формате, или E-формате. Согласно данному формату перед точкой ставится одна цифра, а после точки — число, состоящее максимум из 16-ти цифр, затем следуют символ E и показатель степени с основанием 10. Три числа, представленные выше, в E-формате имеют такой вид:

```
3.141592E0
-8.05E1
7E-6
```

Над вещественными числами могут выполняться операции сложения, умножения и деления. Операция деления по модулю недопустима. Деление вещественных чисел производится иначе, чем целых, о чем упоминалось в разделе 2.5. Чтобы выполнить деление вещественных чисел, мы должны убедиться, что одна из переменных или констант в выражении является вещественной. В качестве примера рассмотрим следующие выражения:

```
int roughLength = 180;                //в сантиметрах
double preciseLength = 185.3427;      //в сантиметрах
double materialNeeded;                //в метрах

materialNeeded = roughLength/100;     //результат – 1 метр
materialNeeded = roughlength/100.0;   //результат – 1.8 метра

materialNeeded = preciselength/100;   //результат – 1.85327 метров
```

Присваивание в пределах каждого типа данных является допустимым оператором. Эта операция требует, чтобы типы выражения и переменной совпадали. Таким образом, мы не можем присвоить какое-либо значение типа `int` объекту `Curio` или переменную типа `double` переменной типа `int`. Однако помня о том, что даже в математике класс вещественных чисел включает в себя класс целых чисел, мы *можем* присваивать целые значения вещественным. Таким образом, следующее присваивание является допустимым:

```
double rate;
rate = 10;
```

Вывод целых и вещественных значений

Как уже отмечалось, когда необходимо вывести число, Java вызывает свой метод `toString`, который возвращает строку, передаваемую затем методу `print`. Метод `toString` не содержит информации о том, сколько разрядов он должен получить в результате, поэтому передает их все. Целые числа типа `long` могут иметь 10 разрядов, а вещественные числа типа `double` — до 17. Если размер вещественного числа превышает 17 значащих разрядов, Java выводит число в E-формате. А как в нашем языке программирования будут выведены следующие выражения?

Число	Вид после вывода
10	10
650	650
2001	2001
14.75	14.75
1.6213	1.6213
1/1.6213	0.6167889964843027
123456789	123456789
0.000000001	1.0E-9
1.0/2	0.5
1/2	0

Удивление у вас может вызвать лишь последний пример. Дело в том, что Java воспринимает знак "/" как целочисленный оператор, если оба значения в выражении являются целыми. Таким образом, деление 1 на 2 дает в результате 0.

При использовании описанной выше схемы проблему могут составить длинные вещественные числа. Во избежание ошибок нужно либо использовать специально созданный пакет ввода/вывода (ВВ), либо разыскать его среди множества форматирующих классов пакета `java.text`. Решением этих вопросов мы займемся в главах 4 и 7.

Класс Math

Выполнение стандартных арифметических и тригонометрических функций, которые поддерживает любой хороший калькулятор, в языке Java обеспечивается с помощью специального класса `Math`. Он принадлежит к пакету `java.lang`, который автоматически импортируется в каждую программу. Поэтому мы можем использовать функции класса `Math` когда угодно. Ниже представлены наиболее важные из них.

Некоторые методы класса `Math`

```
final double PI
double pow (double, double)
double sqrt (double)
double atan (double)
double cos (double)          // в радианах
double sin (double)          // в радианах
double tan (double)          // в радианах
double toDegrees (double)
double toRadians (double)
double random ();
int round (float)
long round (double)
value abs (value)          // значение может иметь тип int, long, float, double
value max (value, value)  // значение может иметь тип int, long, float, double
value min (value, value)  // значение может иметь тип int, long, float, double
```

`PI` является константой, значение которой совпадает с первыми 10 разрядами числа π.
Метод `pow` возводит первый параметр в степень с показателем, равным второму параметру. Если второй параметр имеет дробное значение, `pow` выполняет роль функции извлечения корня (см. пример ниже).

Четыре тригонометрических метода работают со значениями параметров, выраженными в радианах. Таким образом, 360° – это 2*PI, а 90° — это PI/2.
Допускаются взаимные преобразования значений и выражений в градусах или радианах.
Метод random возвращает вещественное значение в диапазоне от 0 до 1, который впоследствии может быть расширен (см. примеры ниже).
Методы round округляют значение параметра до ближайшего целого числа (см. примеры ниже).
Метод abs возвращает значение абсолютной величины (модуля) параметра.
Метод max из двух значений параметров выбирает максимальное, а метод min — минимальное.
Методы abs, max и min имеют по четыре версии для каждого из перечисленных типов данных. Результат имеет тот же тип, что и данные параметры.

Вы, наверное, заметили, что в качестве аргументов большинства функций используются значения типа double, а результат формируется, как правило, в виде переменной, которая тоже имеет тип double. Класс Math интересен с нескольких точек зрения:

- это первый встроенный класс языка Java, который мы изучаем подробно;
- все его методы известны как методы класса; это новое для нас понятие, которое сейчас будет раскрыто;
- все его методы имеют свой тип, отличный от void, т. е. являются *типизированными*; это еще одно новое понятие, которое будет введено.

Сначала рассмотрим несколько примеров, чтобы получить представление о том, как используется класс Math, а затем дадим определения новым терминам. Вот типичные способы применения класса Math:

```
double squareRoot, cubeRoot, sin0f90, x;
int player1, player2, winner, diceRoll;

squareRoot = Math.sqrt(x);
cubeRoot = Math.pow(x,3);
sin0f09 = Math.sin(Math.toRadians(90));
winner = Math.max(player1, player2);
diceRoll = Math.round(Math.random()*6);
```

Пример 3.1. Исследование класса Math

Как используется класс Math, мы продемонстрируем на примере простой программы, которая генерирует два случайных числа и выводит большее из них с помощью метода max.

```
class RandomInvestigation {

 RandomInvestigation () {
   double random1, random2;
   random1 = Math.random();
   random2 = Math.random();
   System.out.println("The numbers are:" + random1 + "and" + andom2);
   System.out.println("Max is" + Math.max (random1, random2));
 }
 public static void main (String [] args) {
   new RandomInvestigation ();
 }
}
```

```
The numbers are: 0.8084638874747664 and 0.8945058642673429
Max is 0.8945058642673429
```

При выводе выражения "в одну строку" для того, чтобы возвести в степень, необходимо вызвать процедуру, показанную в следующем примере:

x^2 `Math.pow(x,2)`

Когда мы вводим формулу, тригонометрические функции вызываются следующим образом:

```
sin x  Math.sin(x)
```

Еще один полезный метод класса `Math` выполняет функцию округления вещественного числа к ближайшему целому. Рассмотрим примеры:

```
Math.round(6.6)   7
Math.round(6.3)   6
```

Здесь Java поступает не совсем обычным образом. Предполагается, что любое выражение с вещественными числами дает в результате значение типа `double`. Поэтому, если процедура `round` вызывается для значений типа `double`, она в результате даст значение типа `long`, а не `int`. Чтобы получить число типа `int`, необходимо выполнить явное преобразование, которое описано ниже. А сейчас мы поговорим о том, что такое метод класса и функции.

Методы класса

Когда создан объект какого-либо класса, в нем есть методы, определенные внутри этого класса. Доступ к этим методам осуществляется с помощью команды `objectid.metodid()`. Исключением является только метод `main`, который мы не показываем в диаграмме моделирования как часть объекта: он остается внутри класса. Чтобы убедиться в этом, взгляните на рис. 2.3. Причина в том, что метод `main` был объявлен как `static`, поэтому он является единственным и принадлежит самому классу. Когда виртуальная машина Java ищет метод `main` для запуска, она использует идентификатор `Classid.main()`.

С классом `Math` дело обстоит точно так же. Все его методы объявлены как `static`, так что для их использования объект не создается: доступ к нему осуществляется прямо через имя класса.

Типизированные методы

О методах данного вида подробно мы поговорим в одном из следующих разделов данной главы. Сейчас мы лишь отметим, что методы, которые определяются как имеющие типы (как все методы класса `Math`) могут быть использованы в выражениях. Метод выполняет необходимые вычисления с заданными параметрами и возвращает значение определенного типа.

Преобразование типов

В каких случаях может понадобиться преобразование одного численного типа в другой? Например тогда, когда результатом вычислений является вещественное значение, которое для облегчения понимания должно быть представлено в виде целого числа. Пока мы рассмотрели два варианта преобразования:

- Округление чисел типа `double` до чисел типа `long` с использованием функции `Math.round`.
- Присвоение значения типа `int` числу типа `double`.

Второй вариант преобразования широко применяется к данным численных типов при увеличении порядка чисел. Как следует из табл. 3.1, мы можем присвоить значение типа `byte` переменной любого другого типа, `short` — любой переменной, кроме `byte`, и т. д. А что произойдет, если выполнить присвоение наоборот? С нашей точки зрения, нужно будет "укоротить" значение типа `double`, преобразовав его во `float`. Для этой цели следует выполнить операцию приведения типа:

Приведение типа
(type) expression
Выражение `expression` преобразовывается в значение типа `type`, если его значение совместимо с этим типом. В противном случае на экран выводится сообщение об ошибке.

Рассмотрим пример:

```
float kilograms;
double estimate = 45;
kilograms = (float)(estimate * 1.2);
```

Это условие означает, что значение выражения должно быть совместимо со значениями заданного типа. Мы не можем значению 1000 типа `short` присвоить тип `byte`, поскольку оно для этого не подходит. Java выдаст сообщение об ошибке компиляции или ошибке рабочего цикла.

В случае приведения вещественных типов к целочисленным отбрасывается любая дробная часть и осуществляется проверка на совместимость численной части выражения с типом данных, к которому ее пытаются привести, после чего приведение данных продолжается. Например:

```
(int) 6.3      дает в результате 6
(int) 6.8      дает в результате 6
```

Тем не менее, приведение значения 1000.678 к типу `byte` по-прежнему невозможно.

3.2. Повторение с помощью цикла for

В программе повторения одних и тех же действий могут быть определены как **циклы**. В языке Java, как и в большинстве языков программирования, существует два вида циклов — **циклы `for`** и **условные циклы**. В этом разделе мы рассмотрим циклы первого вида, а об условных циклах поговорим в главе 5. В Java циклы обоих видов зависят от условий. Цикл `for` использует их главным образом для отсчета числа повторений.

Простые условия

Условия предназначены для определения того, какая из альтернативных частей алгоритма будет выбрана. Они могут принимать значение **true** (истина) или **false** (ложь). Другое название условия — **булево***, или **логическое, выражение**. Выражения этого типа будут подробно рассмотрены в главе 4. Между тем, мы можем использовать шесть операторов сравнения, чтобы сопоставлять результаты численных выражений и контролировать выполнение цикла `for`.

---

* Названо в честь Джорджа Буля — математика, жившего в девятнадцатом столетии.

Такими операторами являются:

==	равно
=!	не равно
>	больше чем
<	меньше чем
>=	больше или равно
<=	меньше или равно

Вот примеры условий:

```
i < n
temperature > 0
mark >= 50
```

Цикл for

Рассмотрим основной и самый простой цикл `for` языка Java:

Простой оператор `for`

```
for (int var = start; check; update) {
    body
 }
```

Переменная `var` инициализируется значением выражения *`start`*, параметр *`check`* содержит условие, зависящее от значения `var`. Если проверка показала, что условие истинно, программа переходит к выполнению тела цикла *`body`*. Каждый раз при прохождении цикла значение `var` изменяется в соответствии с параметром *`update`*, после чего снова выполняется проверка условия. Цикл повторяется, пока условие не станет ложным. Следует обратить внимание на то, что проверка осуществляется только в том случае, если цикл возвращается к оператору `for`, т.е. она не производится постоянно. Тело цикла *`Body`* состоит из нуля и более операторов.

Например, самый простой цикл имеет следующий вид:

```
for (int i = 0; i < n; i++){
  doSomething(i);
}
```

В качестве начального значения переменной цикла `i` задается 0. В цикле выполняется проверка на истинность условия `i < n` и вызывается метод `DoSomething`. По возвращении в начальную точку цикл прибавляет единицу к значению переменной и проверяет, осталось ли условие истинным. Если это так, снова выполняется тело цикла, прибавляется единица к переменной и проверяется условие. Когда, в конце концов, условие становится ложным (значение `i` становится равным `n`), управление передается оператору, следующему за `for`.

Следующая конструкция выводит на экран пять рядов звездочек.

```
for (int i = 0; i < 5; i++) {
  System.out.println("**********");
}
```

Где гарантия, что будет выведено пять, а не шесть рядов звездочек? После завершения инициализации (которая осуществляется только один раз) цикл выполняет следующую последовательность действий:

- проверка;

- выполнение тела цикла;
- изменение значения переменной цикла.

Вот более подробное описание шагов. Сначала переменной `i` присваивается значение 0, которое сравнивается со значением 5, затем выводится ряд звездочек и к значению `i` прибавляется 1. Так переменная `i` последовательно приобретает значения 3, 4 и 5. Когда значение переменной становится равным 5, проверка показывает, что условие ложно. Цикл завершает свою работу, после чего управление передается операторам, которые следуют за блоком. Теперь мы можем убедиться, что на экран выведено пять рядов звездочек:

```
**********
**********
******** **
**********
**********
```

Важно научиться правильно применять стандартный способ записи простого цикла `for`. Так, чтобы цикл выполнился *n* раз, можно выбрать один из двух вариантов:

```
for (int i=1; i<=n; i++) …
```

или

```
for (int i=0; i<n; i++) …
```

Первый вариант более легок для понимания. Однако в Java существует множество связанных последовательностей (например, массивов, о которых речь пойдет в главе 6), где 0 используется в качестве начального значения. Поэтому принято начинать цикл с нулевого значения. Таким образом, предпочтительнее второй вариант.

Пример 3.2. Печать набора этикеток

Задача. Мисс Мэри Браун, проживающая по адресу Бруклин, Чарльз Стрит, 33, хочет нанести на клейкие этикетки свой адрес.

Решение. По желанию мисс Браун клейкие этикетки должны быть напечатаны одна под другой (позже вы узнаете, как заполнить ими всю страницу). Чтобы решить данную задачу, мы возьмем тело программы `DisplayWarning` из примера 2.4 и вставим его в цикл.

Программа. Программа не является сложной. Обратите внимание на то, что пары этикеток отделяются одна от другой двумя пустыми строками. В данном примере количество этикеток задано в программе, но позже мы рассмотрим, как сделать так, чтобы этот параметр варьировался от запуска к запуску. Еще одна особенность рассматриваемого примера заключается в том, что в этой программе тело цикла состоит из нескольких операторов.

```
class LotsaLabels {

/* Программа Создание этикеток Дж. М. Бишоп, август 1996 г.
 * ------------------------- Java 1.1, октябрь 1997 г.
 *                           изменение, Java 2, апрель 2000 г.
 * Печатает набор этикеток
 * Иллюстрирует простой цикл for
 */

LotsaLabels () {
```

```
  for (int i = 0; i < 8; i++) {
    System.out.println("---------------------------");
    System.out.println("|                         |");
    System.out.println("| Ms Mary Brown           |");
    System.out.println("| 33  Charles Street      |");
    System.out.println("| Brooklyn                |");
    System.out.println("|                         |");
    System.out.println("---------------------------");
  }
}

public static void main(String[] args) {
  new LotsaLabels ();
}
      }
```

Тестирование. Запуск программы осуществляется очень просто, т.к. в ней нет дополнительных частей, которые необходимо тестировать. В результате ее выполнения будет напечатано 8 этикеток.

Использование переменной цикла

Переменная цикла служит для записи текущих итераций цикла. Ее значения могут использоваться по-разному, например:

- в операторе вывода;
- при выполнении простых арифметических действий;
- в качестве граничных значений других циклов.

Остановимся на первом способе. Мы можем подсчитать количество звездочек в строках:

```
for (int i=0; i<5; i++) {
  System.out.print(i*10);
  System.out.println(" **********");
}
```

Результат будет следующим:

```
0 **********
10 **********
20 **********
30 **********
40 **********
```

Как видите, первая строка немного сдвинута. Теперь рассмотрим, как вывести первые десять четных чисел. Существует два способа решения данной задачи. Мы можем организовать цикл от 0 до 10, не включая граничное значение, и выводить на экран переменную цикла, умноженную на два, как в следующем примере:

```
for (int number = 0; number < 10; number++) {
  System.out.print((number * 2)+" ");
}
System.out.println();
```

вследствие чего получим:

```
0 2 4 6 8 10 12 14 16 18
```

Согласно другому способу, выполняется удваивание переменной в той части цикла, где происходит изменение параметров, как в следующем примере:

```
for (int number = 0; number < 20; number+=2) {
  System.out.print((number * 2)+" ");
}
System.out.println();
```

А вот о том, как вывести первые десять нечетных чисел, придется немного поразмыслить. Если `number * 2` является четным числом, то, следовательно, выражения `number * 2 + 1` и `number * 2 – 1` примут нечетные значения. Выбрав одно из этих выражений, получим следующий цикл:

```
for (int number = 0; number < 10; number++) {
  System.out.print((number+" ");
}
System.out.println();
```

В результате будет выведена такая последовательность:

```
1 3 5 7 9 11 13 15 17 19
```

Пример 3.3. Таблица преобразований

Задача. Океанографическая лаборатория Саванны измеряет температуру озер в градусах Цельсия, однако некоторым сотрудникам требуются значения в градусах Фаренгейта. Менеджеры приняли решение сформировать таблицы преобразований. Для этого необходимо написать программу, которая генерирует такую таблицу.

Решение. Для того чтобы перебрать все значения в диапазоне требуемых температур, используйте цикл, который сначала выводит значение в градусах Цельсия, а затем преобразует его в соответствии со шкалой Фаренгейта.

Алгоритм. Сначала выводится заголовок, а затем организуется цикл, выводящий на экран требуемую нам таблицу. Каким должен быть диапазон значений температур в градусах Цельсия? Поскольку это температура воды, то граничными значениями могут быть 5 и 20.

Программа. Программа несложная. В ней используется функция `Math.round`, чтобы полученные результаты являлись целыми числами. Для того чтобы выводимые данные имели приемлемый формат, применяется сочетание символов `\t` для выполнения табуляции. Мы следим за тем, чтобы операнды для операции деления были вещественными числами и данная операция выполнялась корректно.

```
class TemperatureTable {

/* Программа преобразования температур Дж. М. Бишоп, август 1996 г.
 * ---------------------------------- Java 1.1 изменена, апрель 2000 г.
 * Выводит таблицу преобразований значений температуры в градусах Цельсия
 * в соответствии со шкалой Фаренгейта в пределах заданного диапазона
 * Иллюстрирует использование переменной цикла
 * в выражениях внутри цикла.
 */

TemperatureTable () {
  System.out.println("Temperature Conversion Table");
  System.out.println("===========================");
  System.out.println();
  System.out.println("C\tF");
```

```
  for (int c = 5; c <= 20; c++) {
    System.out.println(c+"\t"+ Math.round(c*9.0/5 + 32));
  }
}

public static void main (String [] args) {
  new TemperatureTable ();
}
}
```

Тестирование. Программой выполняется вывод данных на экран. В командной строке мы можем задать перенаправление потока данных в файл, который впоследствии будет напечатан:

```
java TemperatureTable > temptable.out
```

```
Temperature Convertion Table
============================

C   F
5   41
6   42
7   44
8   46
9   48
10  50
11  51
12  53
13  55
14  57
15  59
16  60
17  62
18  64
19  66
20  68
```

Другие виды циклов

- **Обратный отсчет**. Мы уже отмечали, что цикл может выполнять обратный отсчет с помощью оператора --. Например, вывести последовательность

  ```
  10 9 8 7 6 5 4 3 2 1 0 -1 -2 - 3 -4 -5 -6
  ```

  позволяет следующий цикл:

  ```
  for (int n = 10; n >=-6; n--) {
    System.out.print(n + " ");
  }
  System.out.println();
  ```

- **Пустой цикл**. Возможны ситуации, когда начальное условие превосходит конечное. В таких случаях тело цикла совсем не выполняется. Так, если в цикле

  ```
  for (int n = 0; n < finish; n++) {
    System.out.print(n + " ");
  }
  System.out.println();
  ```

  переменная `finish` имеет отрицательное значение, сложится описанная ситуация и будет выведена только пустая строка.

- **Вложенные циклы.** К этой категории относятся циклы, расположенные внутри других циклов. В таких случаях переменные циклов должны быть разными.
- **Бесконечный цикл.** Иногда необходимо использовать другой способ выхода из цикла, чем отсчет вперед или назад. Как мы узнаем из следующей главы, благодаря графическому пользовательскому интерфейсу программу можно остановить, всего лишь нажав кнопку на экране. Чтобы цикл `for` выполнялся без остановки, мы пропускаем часть, в которой проверяется условие:

  ```
  for (int i = 0; ; i++)
  ```

  Здесь при каждой итерации значение `i` увеличивается на единицу. Если нет необходимости вводить переменную `i`, мы можем просто написать:

  ```
  for (; ;)
  ```

  Это не является наилучшим способом. Правильное решение — использование условного цикла. Эти вопросы рассмотрены в главе 5, а пока нам придется применять бесконечный цикл `for`.
- **Прерванный цикл.** В главе 4 описаны ситуации, при которых требуется остановить цикл в процессе итерации. Такую возможность предоставляет оператор `break`, передающий управление в конец цикла. Он используется вместе с условным оператором `if`, который также будет рассмотрен в главе 4.

3.3. Создание методов

Вы уже получили общее представление о методах в разделе 2.2 и даже поработали с некоторыми из них (`println` и `write`). Кроме того, мы часто ссылались на метод `main` и конструкторы. Сейчас речь пойдет о том, почему методы играют такую важную роль в программировании и как объявлять необходимые методы.

Объект состоит из переменных и методов. Другими словами, объект **заключает в себе** данные и информацию о том, какие операции должны быть выполнены по отношению к этим данным. Для обеспечения эффективного взаимодействия с объектами мы выбираем методы так, чтобы как можно реже обращаться к переменным напрямую. Конечно, такая возможность существует, но совмещение переменных и методов позволяет сэкономить время и создавать программы с невысокой вероятностью возникновения ошибок.

Объявление метода

Метод — это группа переменных и операторов, которым присваивается определенное имя и по которому она может быть вызвана для выполнения определенного действия. Существует несколько типов методов. Мы начнем изложение с описания самых простых методов — *void-методов*. Такие методы объявляются следующим образом:

Объявление простого void-метода

```
void methodid () {
    variables and statements
}
```

Объявляет метод *methodid*, позволяющий определить блок данных, переменных и выражений. Метод может использовать переменные, которые объявляются в нем самом или в его объектах.

Префикс `void` указывает на то, что метод может быть вызван для выполнения автономного действия, например, `println` является void-методом. Ниже приведен пример простого метода этого типа:

```
void report () {
  System.out.println("The Polelo Curio Store sells\n");
  mugs.write();
  tshirts.write();
  carvings.write();
}
```

Этот метод может быть включен в программу `CurioStore1`. Создание подобного метода называется **объявлением**. Идентификатор `report` используется для того, чтобы в дальнейшем можно было ссылаться данный метод.

Метод может использовать переменные, которые объявляются в нем самом или в его объекте. Таким образом, метод `report` может свободно использовать три объектные переменные — `mugs`, `tshirts` и `carvings`. Точно так же метод `write` объекта `Curio` имеет доступ к своим трем переменным:

```
void write () {
System.out.println(item + " " + description+" for G" + price);
}
```

Вызов метода

Если метод объявляется как `void`, то он вызывается через его идентификатор.

Вызов простого void-метода
methodid ();
Вводится только название метода, после чего его операторы выполняются от начала до конца.

Например, чтобы вывести отчет, достаточно указать внутри конструктора `CurioStore1` следующее:

```
report();
```

Мы получим тот же результат, что и ранее:

```
The Polelo Curio Store sells

Traditional mugs beaded in Ndebele style for G6
T-shirts sizes M to XL for G30
Masks carved in wood for G80
```

Использование void-методов для структурирования

Чаще всего `void`-методы применяются для разбиения функции на фрагменты. Возьмем, к примеру, программу `FleetCalculator` (пример 2.6), которая занимает целую страницу. Как следует из комментариев, программа аккуратно разделена на секции, и одним из способов сделать это является использование методов:

```
FleetCalculator () {
  convertInitialTimes();
  calculateJorneyTimes();
```

```
  convertNewTimes();
  reportFindings();
}
```

Каждый из названных методов содержит именно те операторы, которые имела бы программа `FleetCalculator`. Переменные, которые ранее были объявлены внутри конструктора, перемещены на уровень объекта, чтобы все методы имели к ним доступ. Например, первым методом может быть:

```
void convertInitialTimes () {
  departMins = depart24 / 100 * 60 + depart24 % 100;
  arriveMins = arrive24 / 100 * 60 + arrive24 % 100;
}
```

3.4. Передача параметров и возвращение результатов

Метод можно сделать более эффективным, если задать, чтобы он действовал по-разному при каждом вызове. Объявляя какой-либо метод, мы составляем список нужных параметров с указанием их типов. В дальнейшем при вызове метода значения этих параметров должны передаваться ему в правильном порядке. Рассмотрим пример, в котором используется один из хорошо знакомых нам конструкторов

```
Curios (String n, int p, String d)
```

и вызов

```
mugs = new Curios("Traditional mugs", 6, "beaded in Ndebele style");
```

От нас требуется передать в качестве параметров строку, целое число и еще одну строку, строго соблюдая указанный порядок, поскольку любая другая расстановка будет отклонена. Идентификаторы, данные формальным параметрам, являются локальными только для того метода, где они определены, и на них нельзя ссылаться извне. Вот почему именно от конструкторов зависит копирование их параметров в переменные, определенные классом для объектов. Предположим, у нас есть следующий код:

```
class Curios {
  String item;
  int price;
  String description;
Curios (String n, int p, String d) {
  item = n;
  price = p;
  description = d;
}

и т.д.
```

Мы ссылаемся на элементы в обоих случаях — при объявлении метода и при вызове их в качестве параметров. Однако следует учитывать, что при объявлении вводятся **формальные** параметры, а при вызове — **действительные**. Формальные параметры можно сравнить с переменными, а действительные потенциально являются выражениями, которые генерируются до передачи методу.

Формальные параметры объявляются следующим образом:

Объявление формальных параметров
`(type field, type field ...)`

А вот объявление действительных параметров:

Объявление действительных параметров
`(expression, expression, ...)`

Число, тип и порядок следования формальных и действительных параметров должны полностью совпадать, как показано в следующем примере. Кроме того, формальные параметры входят в состав переменных метода, следовательно, каждый идентификатор должен быть уникальным. Например, если существует формальный параметр с именем p, мы не можем объявить в методе переменную с таким же идентификатором. Очень удобно (в первую очередь это касается конструкторов) использовать для формальных параметров идентификаторы, состоящие из одной буквы, поскольку это позволяет предотвратить конфликт с переменными, которые имеют значимые имена.

Объявление void-метода

`void methodid (formalparameters) {`
`  variables and statements`
`}`

Объявляет *methodid*, чтобы обозначить блок данных переменных и операторов. Формальные параметры должны иметь идентификаторы, отличные от идентификаторов переменных. Метод может использовать переменные, которые объявлены внутри него или его объекта.

Вызов метода осуществляется следующим образом:

Вызов void-метода
`methodid (actualparameters);`
Каждый из действительных параметров соответствует формальному параметру, при этом значения первых присваиваются вторым. Вводится только название метода, после чего операторы выполняются от начала до конца.

Пример 3.4. Гистограмма уровня осадков

Задача. Метеорологическая служба Саванны хотела бы составить график, демонстрирующий уровень осадков в каждом месяце года.

Решение. Какой тип графика избрать? Для демонстрации уровней осадков хорошо подходят гистограммы. На оси Y будут указаны месяцы, а на оси X — значения в сантиметрах, показывающие объем осадков.

Вот нужный нам результат:

```
Количество осадков в Саванне
============================
за 2000 год
1   |*************************25
2   |****************16
3   |********************************32
4   |********8
5   |*****5
6   |********************20
7   |*******************19
8   |************************23
9   |********8
10  |***********************************35
11  |*****************************28
12  |*************************25
    ================================================
    +         +         +         +         +         +
    0         10        20        30        40        50
                          см
```

При написании программы мы будем использовать все изученные возможности языка Java. План действий таков:

- Решите, как напечатать простую полосу гистограммы для данного значения h, включая, если необходимо, проверку соответствия длины этой полосы шкале. Данная задача может быть решена с использованием метода `bar`, который имеет дополнительный параметр для метки на оси Y (месяц).
- Сгенерируйте случайное число от 0 до 40, которое послужит значением уровня осадков.
- Вызовите метод `bar` с формулой в качестве параметра.
- Заключите упомянутый вызов метода в цикл, параметром которого является количество месяцев.
- Используйте операторы `println` для формирования заголовка и оси X, используя цикл.

Программа. Программа получится довольно длинной, но мы упростим ее восприятие, введя комментарии. Обратите внимание на то, что сначала указан главный метод, а затем — вызываемые им методы `bar` и `axis`. В отличие от некоторых других языков, для Java не имеет значения, в каком порядке объявляются методы.

```
class RainfallHistogram {

/*
 * Программа для построения гистограммы, демонстрирующей уровни осадков
 * ---------------------------------------------------------------------
 *                                          Дж. М. Бишоп, апрель 2000 г.
 *                                          адаптирована из Плана развития
 *                                          в JG2E, октябрь 1997 г.
 * Выводит гистограмму, показывающую уровень осадков, для каждого месяца,
 * количество осадков генерируется случайным образом.
 *
 * Иллюстрирует передачу параметров для переменных и циклов for.
 *
 */

RainfallHistogram () {
```

```
//Заголовки
  System.out.println ("Savanna Rainfall");
  System.out.println ("================");
  System.out.println ("for 2000");

//Отображает полосу, соответствующую каждому месяцу, и ось.
  for (int month = 1; month <= 12; month++) {
      bar(month, Math.random()*40);
  }
  axis ();
}

void bar(int label, double h) {
//Создает из заданного числа звездочек одну полосу
//гистограммы с меткой, соответствующей определенному месяцу.
  System.out.print(label+"\t|");
  int stop = (int) Math.round(h);
  for (int star = 0; star < stop; star++)
      System.out.print('*');
  System.out.println(" " + (long) h);
}

void axis () {
//Рисует горизонтальную ось с соответствующим количеством
//делений, равным количеству меток + 1, и шагом 9.
//Расстояние между символами равно 9 символам.

   int ticks = 5;

//Вывод строки.
   System.out.print('\t');
   for (int line = 0; line < ticks*10; line++)
       System.out.print("=");
   System.out.println("=");

//Вывод меток делений.
   System.out.print('\t');
   for (int n = 0; n < ticks; n++)
       System.out.print("+         ");
   System.out.println('+');

//Вывод цены деления.
   System.out.print('\t');
   for (int n = 0; n <= ticks; n++)
       System.out.print(n*10 + "         ");
   System.out.println();

//Вывод единицы измерения для оси.
   System.out.println("\t\t\t\tсм");
}

public static void main (String [] args) {
  new RainfallHistogram ();
}
}
```

Тестирование. Выходные данные уже были показаны. Работа с данным примером будет продолжена при решении следующих задач, в которых требуется построить графики с учетом различных данных.

Диаграмма моделирования для методов

Методы представлены на диаграмме моделирования и перечислены в соответствующем разделе объектов, созданных на основе классов. Обычно на диаграммах не дается полная информация о методах, но если эти детали имеют большое значение, мы предоставляем их, используя прямоугольник с загнутым углом (рис. 2.4). Указать внутри класса, какими методами вызываются другие методы, можно с помощью стрелок. На рис. 3.1 программа из предыдущего примера представлена в виде диаграммы моделирования.

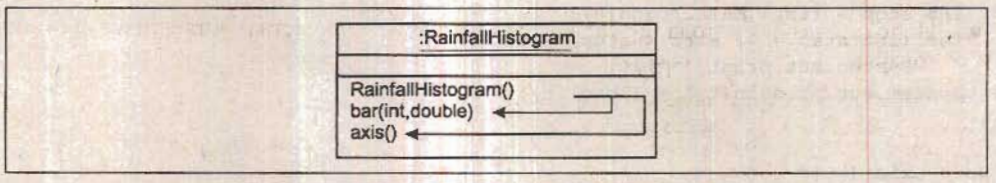


Рис. 3.1. Диаграмма моделирования для программы RainfallHistogram

Для каждого метода указан тип его параметров, информация о которых является очень полезной. Как мы уже видели, подробные сведения о методе можно дать, используя прямоугольники с загнутым углом. Эти символы могут также применяться для отображения самих методов (рис. 3.6). На рис. 3.2 представлена программа, которая строит гистограмму уровней осадков с подробным описанием методов. Текст взят из комментариев программы.

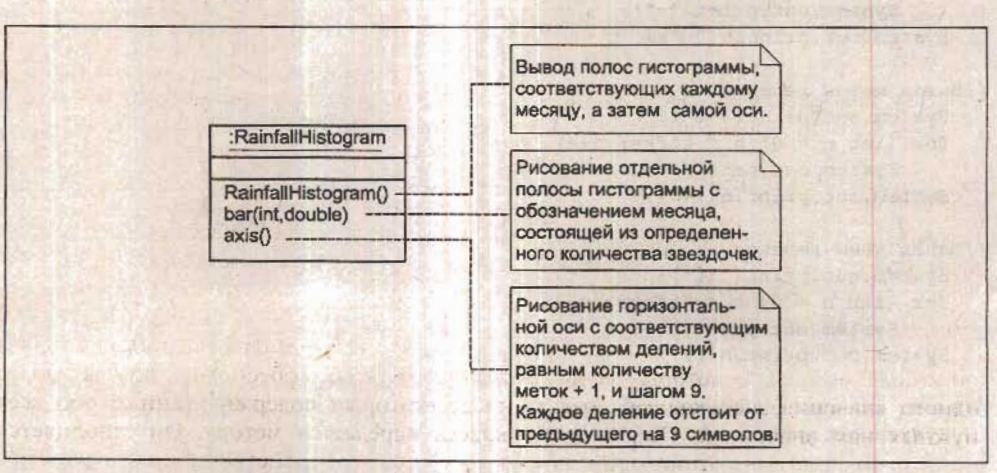


Рис. 3.2. Диаграмма моделирования для программы RainfallHistogram, где поясняется значение каждого метода

Объекты в качестве параметров

Рассмотрим, как передаются параметры, на примере переменной `inStock` объекта `Curio`. Предположим, что мы занимаемся торговыми операциями и нам необходимо, чтобы значение упомянутой переменной изменялось в соответствии с количеством проданных единиц товара. Существует три способа решения данной задачи:

- Использование метода `update` для подсчета количества проданных единиц товара:

```
Void update (int sold) {
    inStock -= sold;
}
```

Таким образом, если продано три кружки, надлежит вызвать такой метод:

```
mugs.update (3);
```

- Использование метода `update` для подсчета количества оставшихся единиц товара:

```
void update (int newStock) {
    intStock = newStock;
}
```

Опять-таки, если продано три кружки, вызывается метод:

```
mugs.update (mugs.inStock-3);
```

Вычисление выполняется при вызове. Преимущество этого варианта заключается в том, что мы можем обнулить весь ассортимент (допустим, после пожара), вызвав такой метод:

```
mugs.update (50);
```

- Мы можем использовать прямой доступ к переменной `inStock`, не обращаясь к методу

```
mugs.inStock -= 3;
```

Это наиболее удачное решение, поскольку в таком случае можно работать с переменной любым подходящим для вас способом.

Результаты работы метода

Когда мы вызываем метод объекта, мы воздействуем на его переменные. Результаты такого воздействия в некотором смысле можно считать результатами вызова. Теперь новые значения переменных объекта доступны тому, кто вызывает метод, и могут быть проверены.

Если метод должен возвратить всего один результат, идеально подойдут типизированные методы, о которых речь пойдет ниже. Чтобы обеспечить возврат более одного значения, необходимо создать класс, который содержит данные обо всех нужных нам значениях. Объект этого класса передается методу. Он заполняется новыми значениями, которые затем будут доступны через действительные параметры при возврате. Мы вернемся к анализу этого процесса в конце главы при решении практической задачи.

3.5. Типизированные методы

Типизированные методы часто используются в программировании на языке Java, поскольку их применение является простым способом получения результата, который возвращает метод. Кроме того, их легко использовать в выражениях. Методы этого вида называют также **функциями**. Существует целое направление в вычислительной технике, посвященное функциональному программированию. Рассмотрим пример практического применения типизированного метода: преобразуем значения времени, представленные по 24-часовой системе, в минуты (см. пример 2.6). Поскольку эта операция выполняется в программе, по крайней мере, дважды, создадим такой метод:

```
int timeInMins (int time24) {
  return time24 / 100 * 60 + time24 % 100;
}
```

Вызов типизированного метода представляет собой выражение, которое может использоваться в любой части кода, где только это позволяет тип метода. Как правило, такие методы могут быть вызваны в операторах вывода или в выражениях присваивания. Метод преобразования, определенный выше, может быть вызван следующим образом:

```
departMins = timeInMins(depart24);
arriveMins = timeInMins(arrive24);
```

Здесь метод вызывается дважды с разными переменными в качестве действительного параметра в каждом случае. Внутри метода значения переменных доступны по имени `time24`.

Объявление методов — представляют они собой малую часть программ или подразделы управления классами — позволяет сократить код программы, выявить ее структуру и сделать удобной для восприятия.

Объявление типизированных методов

```
type metodid (formalparameters) {
     variables and statements
     return value;
 }
```

Метод *metodid* объявляется для выделения блока данных переменных и операторов.
Формальные параметры, если таковые существуют, должны иметь идентификаторы, отличающиеся от идентификаторов переменных.
Метод может использовать переменные, объявленные в нем самом или его объекте.
Оператор возврата `return` может располагаться в любом месте блока. Он устанавливает значение нужного типа, которое становится значением метода в точке вызова.

Ниже показано, как осуществляется вызов типизированных методов:

Вызов типизированных методов

```
methodid (actualparameters);
```

Действительные параметры полностью соответствуют формальным параметрам. Значения первых присваиваются вторым.
Вводится только название метода, затем операторы выполняются от начала до конца, после чего значение, возвращаемое методом, становится результатом вызова.

Пример 3.5. Объем продаж в лавке древностей

Задача. Владелец лавки древностей Сэм Нельсон хотел бы использовать в работе программу, фиксирующую объем продаж в его магазине.

Решение. Для того чтобы найти решение задачи, нужно исследовать, какими характеристиками должен обладать класс для древностей. Затем мы переделаем класс `Curio` и проверим его, добавив новые методы к его методам из программы `CurioStore2`.

Разработка. Для переписи товаров должна быть объявлена переменная. Мы воспользуемся конструктором, чтобы упорядочить записи о товарах, как и при работе с другими переменными. Но в данном случае применение конструктора будет связано с некоторыми особенностями. Он может обновляться в течение продолжительного периода времени. Таким образом, нам необходим метод, позволяющий пополнять список товаров, и конструктор, в котором изначально количество товаров равно нулю. Надлежит обеспечить также возможность просмотра списков проданных и оставшихся товаров. Кроме того, мы будем использовать метод `sell`, который возвращает строку, содержащую информацию о том, что было продано. Эти сведения могут быть выведены в случае необходимости.

На рис. 3.3 показана диаграмма моделирования для программы CurioStore2. При построении диаграммы учитывалась информация о типах переменных и параметрах двух классов. Она содержит объект CurioStore2, а также один из объектов Curio. В диаграмму не включены классы, поскольку они уже хорошо вам известны. Результирующая диаграмма идентична представленной на рис. 2.10, с той лишь разницей, что в ней присутствуют дополнительные методы.

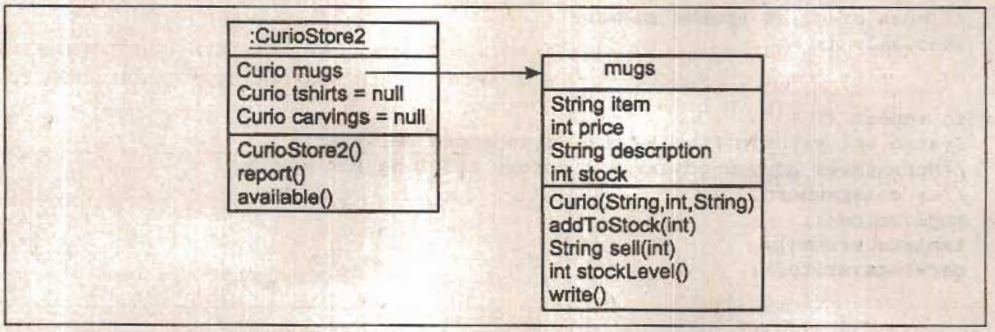


Рис. 3.3. Диаграмма моделирования для программы CurioStore2 (показан этап после инициализации объекта `mugs`)

Программа. Откорректированная программа имеет следующий вид:

```
class CurioStore2 {

/* Лавка древностей № 2, программа  Дж.М. Бишоп, апрель 2000 г.
 * --------------------
 * Магазин, имеющий запас товаров и осуществляющий их продажу.
 * Иллюстрирует применение методов в объектно-ориентированной программе.
 */

public static void main (String [ ] args) {
   //создает объект класса
   new CurioStore2 ()
}
```

```
  //Объявляет три объекта.
  Curio mugs, tshirts, carvings;

  //Конструктор программы,
  //где выполняется инициализация и основная часть работы программы.
  CurioStore2 () {
  //Инициализирует объекты с различными начальными значениями.
  mugs = new Curio("Traditional mugs", 6, "beaded in Mdebele style");
  tshirts = new Curio("T-shirts", 30, "sizes M to XL");
  carvings = new Curio ("Masks", 80, "carved in wood");

   //Вывод отчета.
   report ();

   //Добавляет в запасы магазина определенное количество единиц
   //каждой древности.
   mugs.addToStock(10);
   tshirts.addToStock(50);
   carvings.addToStock(5);

   //Отчет об уровне запасов.
   available();

   //Что-либо продано. Метод sell возвращает строку с квитанцией,
   //которую мы можем вывести.
   System.out.println(tshirts.sell(8));

   //Снова отчет об уровне запасов
   available ()
  }

  void report () {
    System.out.println("The Polelo Curio Store sells\n");
    //Использует доступ объектов к методу toString для вывода
    //их содержимого.
    mugs.write();
    tshirts.write();
    carvings.write();
  }

  void available () {
    System.out.println("Available are "+mugs.stockLevel()+" "+
       tshirts.stockLevel()+" "+carvings.stockLevel()+
       " curios respectively\n");
   }
  }

  class Curio {
      //Внутренний класс,который содержит информацию о каждой древности.
      //Будет расширен в следующих главах в ходе работы над проектом.

      //Объявляет поля, включая одно новое – переменную stock.
      String item;
      int price;
      String description;
      int stock = 0;
```

```
    //Конструктор копирует инициализируемые данные в поля.
    Curio (String n, int p, String d) {
      item = n;
      price = p;
      description = d;
    }

    //Теперь по отношению к элементам списка
    //древностей можно выполнять новые операции.
    void addToStock (int n) {
      stock += n;
    }

    String sell (int n) {
      stock -= n;
      return "Sold " + n + " " + item;
    }

    int stockLevel () {
      return stock;
    }

    void write () {
      System.out.println(item + " " + description + " for G" + price);
    }
  }
```

Тестирование. В результате выполнения программы мы получим следующие данные:

```
The Polelo Curio Store sells

Traditional mugs beaded in Ndebele style for G6
T-shirts sizes M to XL for G30
Masks carved in wood for G80
Available are 10 50 5 curios respectively

Sold 8 T-shirts
Available are 10 42 5 curios respectively
```

Мы можем усовершенствовать процедуру вывода данных, что будет сделано в конце главы. Многие элементы программы следовало бы протестировать, однако мы не будем этим заниматься, пока не научимся осуществлять чтение данных.

Пример 3.6. Большая таблица преобразований

Задача. Сотрудники океанографической лаборатории были так довольны подготовленной нами таблицей (пример 3.3), что попросили нас создать еще одну для большего диапазона температур и придать ей такой формат, чтобы таблица полностью помещалась на экране или странице.

Решение. Мы знаем, как осуществить преобразование, так что сейчас должны подумать о том, как разместить таблицу. Как правило, длина строки на экране составляет 80 символов. Каждая пара температур занимает около 16 символов. Таким образом, мы можем создать пять столбцов. Для диапазона температур от 0 до 99°C понадобится 20 строк, что как раз подойдет для экрана или страницы. В своей работе мы будем использовать вложенные циклы.

Алгоритм. Мы можем заимствовать решение, описанное выше. Нам известно, как вывести одну линию, поскольку это выполнялось в примере 3.3 с помощью следующей команды:

```
System.out.println(c+"\t"+ Math.round(c*9.0/5) + 32);
```

Таким образом, мы можем организовать цикл, который будет выводить 20 строк. Для решения данной задачи можно воспользоваться void-методом.

Программа. Определим следующие значения:

```
int colsPerLine = 5;
int maxLineNo = 20;
```

Строки будут пронумерованы в цикле от 0 до 19 (не включая 20), а в роли переменной цикла выступит `line`. Вывод одной строки может осуществляться с помощью метода `outLine`, который содержит собственный цикл со своей переменной цикла, например `col`. Таким образом, выражение для значения в градусах Цельсия в данной строке и в данном столбце будет следующим:

```
thisline * colsPerLine + col
```

Мы несколько усовершенствуем те части программы, где появляется указанное выражение. Сначала будет создани метод, который преобразует значения, выраженные в градусах Цельсия, в соответствии со шкалой Фаренгейта. Поскольку значение в градусах Цельсия появляется дважды — при выводе и во время преобразования, мы присвоим его локальной переменной `c`.

Наконец, следует выбрать такой шаг табуляции, чтобы в столбцах можно было выполнить выравнивание по вертикали как заголовков, так и всех значений. Обратите внимание на то, что выравнивание будет осуществляться по левому краю, а не по правому, поскольку такая таблица выглядит лучше и легче воспринимается. Следующий код снабжен полным набором комментариев, объясняющих каждый шаг программы.

```
class LargeTemperatureTable {
  /* Вывод большой таблицы температур Дж.М. Бишоп, сентябрь, 1996 г.
   * ------------------------------- Java 1.1
   *                                 изменена, апрель, 2000 г.
   * Составляет таблицу преобразования значений, выраженных в градусах
   * Цельсия, в соответствии со шкалой Фаренгейта в диапазоне от 0 до 99.
   *
   * Иллюстрирует использование цикла for, методов для структурирования
   * программы, типизированных методов и параметров.
   */

  int colsPerLine = 5;
  int maxLineNo = 20;
  String gap = "\t\t";

  LargeTemperatureTable () {

    // Сначала выводится заголовок.
    System.out.println("\t\tTemperature Conversion Table");
    System.out.println("\t\t============================");
    System.out.println();

    for (int col = 0; col < colsPerLine; col++)
      System.out.print("C F" + gap);
    System.out.println();
```

```
    // Затем выводится таблица:
    // для каждой строки, которая должна быть выведена, вызывается метод
    // outLine, которому в качестве параметра передается номер строки.
    for (int r = 0; r < maxLineNo; r++)
      outLine(r);
  }

  void outLine(int thisline) {
    /* Используя переданную в качестве параметра информацию о строке,
     * метод вычисляет значения в градусах Цельсия для этой строки
     * и отображает их вместе с их эквивалентами в градусах Фаренгейта.
     */

    for (int col = 0; col < colsPerLine; col++) {
      int c = thisline * colsPerLine + col;
      System.out.println(c + " ");
      System.out.print(fahrenheit(c) + gap);
    }
    System.out.println();
  }

  // Простая функция преобразования.
  long fahrenheit(int celsius) {
    return Math.round(celsius*9.0/5 + 32);
  }

  public static void main(String[] args) {
    new LargeTemperatureTable ();
  }
}
```

Тестирование. В процессе выполнения программа формирует необходимую таблицу.

```
Temperature Convertion Table
============================

C   F     C   F     C   F     C   F     C   F
0   32    1   34    2   36    3   37    4   39
5   41    6   43    7   45    8   46    9   48
10  50    11  52    12  54    13  55    14  57
15  59    16  61    17  63    18  64    19  66
20  68    21  70    22  72    23  73    24  75
25  77    26  79    27  81    28  82    29  84
30  86    31  88    32  90    33  91    34  93
35  95    36  97    37  99    38  100   39  102
40  104   41  106   42  108   43  109   44  111
45  113   46  115   47  117   48  118   49  120
50  122   51  124   52  126   53  127   54  129
55  131   56  133   57  135   58  136   59  138
60  140   61  142   62  144   63  145   64  147
65  149   66  151   67  153   68  154   69  156
70  158   71  160   72  162   73  163   74  165
75  167   76  169   77  171   78  172   79  174
80  176   81  178   82  180   83  181   84  183
85  185   86  187   87  189   88  190   89  192
90  194   91  196   92  198   93  199   94  201
95  203   96  205   97  207   98  208   99  210
```

Пример 3.6 демонстрирует, как использовать в программах методы и параметры. Обратите внимание на то, что в рассмотренной нами программе каждый раз при вызове метода `outLine` использовалась одна и та же переменная — `r`. Работа с методами и параметрами станет более интересной, когда мы перейдем к рассмотрению программ, в которых действительное выражение, передаваемое формальному параметру, вычисляется с использованием нескольких значений.

3.6. Логический тип boolean

Прежде чем приступить к изучению существенно важных вещей, познакомимся с седьмым простым типом данных языка Java — `boolean`.

Условия

Как указывалось в разделе 3.2, от условий зависит, какая из альтернативных частей алгоритма будет выбрана. Условие принимает значение `true` или `false`. Другое название условия — булево выражение. Выражения этого типа используют шесть операторов сравнения для сопоставления результатов численных выражений. Такими операторами являются:

=	равно;
=!	не равно;
>	больше чем;
<	меньше чем;
>=	больше или равно;
<=	меньше или равно.

Результат такого выражения может храниться в булевой переменной. Например, задав переменные следующим образом:

```
boolean isaMinor, isaPensioner;
int age;
```

мы можем хранить данные о возрасте людей:

```
isaMinor = age < 18;
isaPensioner = age >=65;
```

Отобразить эти сведения на экране позволяет такой метод:

```
System.out.println ("Driver's licence denied:" + isaMinor);
```

Если вам 15 лет, `isaMinor` примет значение `true`, и оператор задаст вывод:

```
Driver's licence denied: true
```

Рассмотрим еще один пример:

```
System.out.println ("It is " + isaPensioner + "that you can
                    ride the bus for free.");
```

Для человека 15-летнего возраста на экран будет выведено следующее сообщение:

```
It is false that you can ride bus for free.
```

Булевы выражения

В математике существуют так называемые булевы операторы, которые могут применяться к булевым переменным и выражениям:

`&`	и;
`|`	или;
`^`	исключающее или;
`!`	не.

Для того чтобы выражение `(x & y)` было истинным, необходимо, чтобы переменные `x` и `y` были истинными. Выражение `(x | y)` является истинным, если хотя бы одна из переменных `x` и `y` принимает истинное значение. Булевы операторы подчиняются принципу иерархии, поэтому при отсутствии скобок оператор `&` всегда выполняется перед оператором `|`. Правила применения операторов с учетом значений булевых переменных даны в следующих таблицах:

and &	false	true
false	false	false
true	false	true

xor ^	false	true
false	false	true
true	true	false

or |	false	true
false	false	true
true	true	true

not !	
false	true
true	false

Рис. 3.4. Применение логических операторов к булевым переменным

Вернемся к рассмотренному нами примеру о бесплатном проезде в транспорте несовершеннолетних и пенсионеров. Мы объявим следующие переменные*:

```
boolean isEmployed, isaYoungWorker, isaVoter, isaTaxpayer;
```

После этого следующие идентификаторы могут быть определены таким образом:

```
isaYoungWorker = isaMinor & isEmployed;
isaVoter = ! isaMinor;
isaTaxpayer = isaVoter | isEmployed;
```

Сочетания булевых операторов используются для формирования более сложных условий. Например, если из того, что несовершеннолетние и пенсионеры могут не оплачивать проезд в автобусе, вытекает, что они не работают, то мы получим:

```
boolean freeBus = (isaPensioner | isaMinor) & !isEmployed;
```

Скобки обеспечивают выполнение оператора `|` в первую очередь.

Булевы операторы особенно полезны в сочетании с операторами сравнения при постановке четких условий. В качестве примера рассмотрим выражение, которое

---

* Рекомендуется начинать идентификатор булевой переменной с префикса 'is' или 'isa'. В этом случае он легче читается.

помогает установить, следует ли закрыть школу в связи с тем, что в ней либо слишком жарко, либо слишком холодно:

```
goHome = temperature > 40 | temperature < 0;
```

В Java операторы сравнения имеют приоритет перед булевыми операторами: первые всегда выполняются перед вторыми (т. е. имеют более высокий приоритет). В других языках все наоборот, поэтому многие программисты заключают рассматриваемые условия в скобки. Выражение, определяющее, должна ли быть закрыта школа, в этом случае выглядит так:

```
goHome = (temperature > 40) | (temperature < 0);
```

С другой стороны, оператор конкатенации имеет преимущество перед операторами `&` и `|`. Мы не можем вывести на экран значения `a&b` и `a|b` с помощью записи

```
println(a&b + a|b);
```

поскольку оператор `+` будет использоваться для вычисления выражения `b + a`, вследствие чего возникнет ошибка. Для исправления ошибки следует поставить скобки:

```
println((a&b) + (a|b));
```

В Java также имеется операция "короткого замыкания" для операторов "и" и "или". Так, в выражении

```
c && d
```

значение `d` учитывается, если значение `c` является истинным. То же относится к оператору `||`: если `c` имеет истинное значение, `d` игнорируется. Хорошим примером употребления таких операторов служит процедура выяснения, какая дата (состоящая из трех целых чисел) является более ранней. Оператор, решающий данную задачу, имеет следующий вид:

```
boolean earlier = y1 < y2 ||
                  (y1 == y2 && m1 < m2) ||
                  (y1 == y2 && m1 == m2 && d1 < d2);
```

С использованием операторов короткого замыкания вычисление значения выражения прекращается, как только становится известным ответ. Этот прием позволит сэкономить немного времени, но существуют случаи, когда важно, чтобы одно условие было защищено другим. Определим, количество наименований товаров в лавке древностей (объект с названием `checked`) меньше 5 или нет. Нам не известно, ссылается ли переменная `checked` на уже созданный объект. Мы можем воспользоваться такой конструкцией:

```
boolean lowStock = checked != null && checked.stock < 5;
```

Далее показано, какой приоритет имеют булевы операторы и операторы сравнения.

Приоритет операторов (для переменных типа `boolean`)

```
group 0: ()
group 1: !
group 2: &
group 3: |
group 4: &&
group 5: ||
```

Сравнение объектов

Булевы операторы и операторы сравнения можно использовать только с переменными простых типов. Для сравнения объектов мы должны сами определить соответствующие методы. Предположим, мы хотим сравнить цены на предметы старины. Добавим следующий метод, возвращающий значение типа `boolean`:

```
boolean cheaper (Curio c) {
  return price < c.price;
}
```

Метод вызывается с помощью первого операнда, который представляет собой объект, затем следуют оператор с точкой и параметр, представляющий собой второй операнд. Например, при использовании значений, заданных в главе 2, где стоимость `mugs` — G6, а `tshirts` — G30, выражение

```
mugs.cheaper (tshirts)
```

не будет истинным. Методы для других операторов, особенно для оператора равенства, также могут быть определены этим способом. Чтобы сделать это для класса древностей, потребуется сравнить две строки и целое число. Поскольку строки сами являются объектами, мы должны сначала узнать, как их сравнивать.

Сравнение строк осуществляется с помощью методов. Список таких методов из класса `String` представлен ниже. Более подробное их описание вы найдете в главе 7.

Методы сравнения строк
int compareTo (*String*) **int** compareToIgnoreCase (*String*) **boolean** equals (*String*) **boolean** equalsIgnoreCase (*String*)

Методы `compareTo` возвращают целое число:

если первый операнд меньше второго — отрицательное число;
если операнды равны — ноль;
если первый операнд больше второго — положительное число.

Это означает, что сравнение строк можно осуществлять следующим образом:

```
name1.compareTo(name2) < 0
```

Данное выражение выполняет то же действие, что и выражение `name1 < name2`.

Перечисленными методами можно воспользоваться, чтобы определить количество древностей. Метод, который мы добавим в класс `Curio`, будет иметь следующий вид:

```
boolean equals (Curio c) {
  return name.equals(c.name) && price == c.price
         && description.equals(c.description);
```

Одинаковые строки могут быть выведены на экран следующим образом:

```
System.out.println("Matching curios found is " + checked.equals(mugs));
```

3.7. Практикум 1. Доставка пиццы 2U

Решая эту задачу, мы пройдем все этапы разработки класса и постараемся обеспечить его широкое применение в дальнейшем. Сначала мы четко сформулируем задачу и наметим путь решения, а затем разработаем программный продукт. В процессе создания программы будет подробно рассказано, как в циклах `for` используются объекты и параметры.

Задача. Компания `Pizza Delivery 2U` заинтересована в своевременной поставке своей продукции клиентам. Важно учитывать следующие факторы: величина очереди к духовке (т. е. количество поступивших заказов) и место проживания заказчика. Клиенты, которые живут ближе, получат свои заказы раньше, чем те, которые живут дальше. Данная компания обслуживает четыре района. Ей необходима поисковая таблица с временной шкалой от 9:00 до 23:00 и значениями очереди от 0 до 5 для каждого из четырех районов.

Решение. Задача кажется довольно сложной, однако благодаря полученным ранее знаниям мы облегчим ее решение. Итак, мы воспользуемся следующими приемами:

- распознавание и управление повторением посредством циклов и методов;
- вывод таблицы (см. пример 3.3);
- преобразование значений времени и их сложение (см. пример 2.6).

Мы можем свести решение задачи к выводу одной таблицы для района x раз (x изменяется в диапазоне от 1 до 4). Таблица должна содержать информацию о длине очереди в верхней строке и значения времени в левом столбце. Если установить для значений времени интервал в 15 минут, мы получим примерно такую таблицу:

```
Pizza 2U Delivery estimates for Area 1
=======================================
Time now                Queue length
         0       1       2       3       4       5
900      915     920     925     930     935     940
915      930     935     940     945     950     955
930      945     950     955     1000    1005    1010
945      1000    1005    1010    1015    1020    1025
...
```

Начальная разработка

Назначение нашей программы — создание таблицы, состоящей из заглавий и определенного числа строк с информацией по каждому из четырех районов. Каждая строка формируется с помощью метода `printaLine`, который еще не определен. Единственное, о чем мы должны подумать, — это о цикле, в котором данный метод выполняется. Так как меткой каждой строки таблицы должно быть значение времени от 9:00 до 23:00 с интервалом в 15 минут, это не будет слишком сложно. Вот конструктор для первой версии программы:

```
PizzaDelivery1 () {
  for (int area = 1; area<= 4; area++) {
  printHeading(area);
  startTable();
  for (int time = 900; time < 2300; time = addTime(time,15))
    printaLine(time, area);
  }
}
```

Методы, которые нужно определить, подчеркнуты. Однако прежде мы рассмотрим метод `printaLine`. Он имеет более простую структуру, чем аналогичный метод `outLine` в программе, выводящей таблицу температур, поскольку в данном случае необходимо вывести только одно значение в столбце, а каждая строка начинается с конкретного значения времени. Цикл будет запускаться для определения длины очереди (от 0 до 5). Нам остается вычислить значения, которые должны быть выведены в столбце. Чтобы получить результирующую величину, мы должны сложить значения времени поступления заказа и времени, которое затрачивается на процесс приготовления:

- время ожидания — 5 минут на каждый заказ в очереди;
- приготовление пиццы — 5 минут;
- обработка заказа — 10 минут;
- доставка — 5 минут для близлежащих районов.

Ниже представлена формула для вычисления значений в таблице:

```
addTime (orderTime, queue*makeTime + processTime + a*driveTime)
```

Другими словами, мы определяем метод `addTime`, который будет иметь тип, поскольку должен возвращать значение времени.

Поработаем над этим методом. Он берет значение времени, которое представляет собой целое число (по 24-часовой шкале), и добавляет к нему минуты. Таким образом, заголовок метода должен иметь следующий вид:

```
int addTime (int t, int m) {
```

Мы специально используем `t` в качестве идентификатора формального параметра, чтобы метод имел как можно более общий вид. Сейчас настало время воспользоваться теми наработками, которые были сделаны в примере 2.6. Тогда мы применяли два оператора, которые преобразовывали значения времени по 24-часовой шкале в минуты и наоборот:

```
arriveMins = arrive24 / 100 * 60 + arrive24 % 100;
newArrive24 = newArriveMins / 60 * 100 + newArriveMins % 60;
```

Если представить эти выражения в виде методов, результат будет следующим:

```
int timeInMins (int t) {
  return t / 100 * 60 + t % 100;
}

int time24 (int t) {
  return t / 60 * 100 + t % 60;
}
```

Данный случай применения метода `addTime` более легкий. Этот метод вызывается для преобразования заданного значения времени в минуты, затем он прибавляет к нему определенное количество минут, после чего новое значение опять выводится в формате часы и минуты. Перечисленные операции выполняются с помощью таких операторов:

```
int addTime (int t, int m) {
  return time24(timeInMins(t) + m);
}
```

При необходимости мы можем заменить используемый метод

```
addTime (time,15)
```

методом

```
time24(timeInMins(time) + 15)
```

Следует отметить, что благодаря методу `addTime` программа становится удобочитаемой. Поэтому мы и применяем именно его.

Диаграмма моделирования

Диаграмма моделирования программы представлена на рис. 3.5. Стрелки в блоке методов показывают, какой метод чем вызывается. В отличие от программ, которые обрабатывают информацию о лавке древностей, данная программа не содержит дополнительных объектов. Диаграмма моделирования, указывающая на вызывающую структуру, действительно может дать нам много информации, не говоря уже о том, что благодаря ей мы получаем четкое представление о структуре методов и их параметрах. Если такая структура усложнится, мы можем создать отдельную диаграмму, как показано на рис. 3.6.

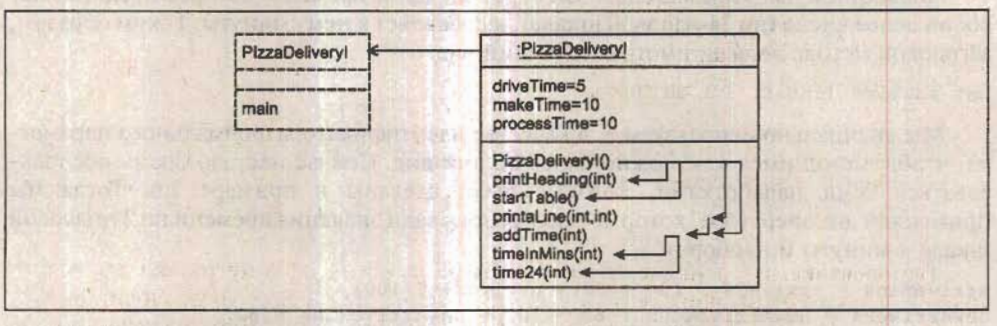


Рис. 3.5. Диаграмма моделирования для программы `PizzaDelivery1`

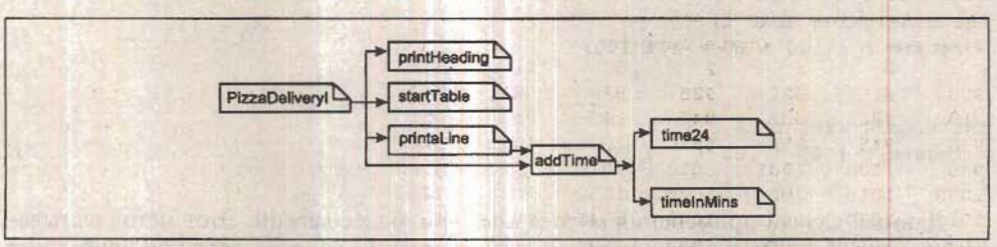


Рис. 3.6. Диаграмма моделирования вызова методов программы `PizzaDelivery1`

Наконец-то мы можем представить программу полностью.

Программа 1

```
class PizzaDelivery1 {

/* Программа доставки пиццы, версия 1, Дж.М. Бишоп, май, 2000 год
 * ================================
```

```
 *
 * Обрабатывает значения времени доставки пиццы, основываясь на
 * таких данных, как длина очереди и район доставки.
 * Иллюстрирует повторное использование методов, их вызов, циклы.
 * Тестовая версия для одного района и значений времени от 9:00 до 13:00.
 */

int driveTime = 5;     //для района
int makeTime = 5;      //в очереди
int processTime = 10; //для заказа

PizzaDelivery1 () {
  for (int area = 1; area<= 1; area++) {
    printHeading(area);
    startTable();
    for (int time = 900; time < 1300; time=addTime(time,15))
      printaLine(time, area);
}
    }

      void printaLine (int t; int a) {
        System.out.print(t+"\t");
        for (int queue = 0; queue <= 5; queue++)
          System.out.print(addTime(t,
            queue*makeTime+processTime+a*driveTime) + "\t" )
        System.out.println();
      }

      public static void main (String [] args) {
        new PizzaDelivery1 ();
      }
     }
```

Тестирование. На данном этапе программа выводит таблицу только для одного района, а также ограничивает шкалу времени значением 1300, чтобы таблицей было легко управлять. Выведенная в результате информация будет иметь следующий вид:

```
Pizza 2U Delivery estimates for Area 1
=======================================
Time now          Queue length
        0     1     2     3     4     5
900     915   920   925   930   935   940
915     930   935   940   945   950   955
930     945   950   955   1000  1005  1010
945     1000  1005  1010  1015  1020  1025
1000    1015  1020  1025  1030  1035  1040
1015    1030  1035  1040  1045  1050  1055
1030    1045  1050  1055  1100  1105  1110
1045    1100  1105  1110  1115  1120  1125
1100    1115  1120  1125  1130  1135  1140
1115    1130  1135  1140  1145  1150  1155
1130    1145  1150  1155  1200  1205  1210
1145    1200  1205  1210  1215  1220  1225
1200    1215  1220  1225  1230  1235  1240
1215    1230  1235  1240  1245  1250  1255
1230    1245  1250  1255  1300  1305  1310
1245    1300  1305  1310  1315  1320  1325
```

Как проверить результаты? Это можно сделать двумя способами. Первый — анализ операторов программы. Второй — выполнение вычислений вручную. Выберем случайным образом пару элементов таблицы, например время 1100 и очередь 2. Время доставки может быть рассчитано следующим образом:

1100 + (2*5 + 10 + 1*5) = 1100 + 25 = 1125

Результат совпадет с соответствующим значением в таблице. Более тщательная проверка должна быть произведена для граничных элементов таблицы, т.е. для последних элементов строк, а также в тех случаях, когда изменяется значение часа (как в строке с начальным значением 1145).

Еще одна проблема

Проводя подобные исследования, мы бы хотели установить, будет ли полезным создание отдельного класса для значений времени. Хотя использование целых чисел для указания времени дало хороший результат, данное решение было некорректным, поскольку 915 означает не "девятьсот пятнадцать секунд", а "девять часов пятнадцать минут". Правильное решение — представить время в виде двух целых чисел (часы и минуты). А для этого мы должны создать класс для значений времени.

Таким образом, нашей задачей является создание такого класса и использование его в программе `PizzaDelivery`, чтобы проверить, как он работает. После этого мы можем сравнить выходные данные предыдущей и новой версий программы `PizzaDelivery`, чтобы убедиться в том, что они не изменились. Нам представится также возможность оценить, насколько использование такого класса усложняет процесс программирования. С нашей точки зрения, новая версия программы немного сложнее, но в тоже время она имеет более четкую структуру, является более гибкой и в ней легче исправлять ошибки.

Классы Date и Calendar языка Java

Прежде чем приступить к разработке нового класса для значений времени, проверим, не предоставляет ли сам язык Java какое-либо средство для работы с этими значениями. Не хотелось бы создавать то, что уже существует. Если мы обратимся к опубликованной в сети документации по языку Java и поищем слово "time", то увидим, что оно занимает одну из первых позиций в списке, который выводится для класса `Date`. Данный класс включает немного методов, перечень которых представлен ниже. Он представляет время в миллисекундах и позволяет работать с датами, начиная с 1 января 1970 года. Однако класс `Date` нам не подойдет, поскольку он работает лишь со значениями времени в миллисекундах.

Класс `Date` языка Java

```
Date ()
Date (long Date)
long getTime ()
void setTime (long time)
boolean after (Date when)
boolean before (Date when)
int compareTo (Date anotherDate)
// плюс обычные методы класса toString и т.д.
```

Кроме того, в языке Java существует класс `Calendar`, который включает большое количество методов для управления датами и значениями времени, а также для выполнения арифметических операций над ними. К сожалению, он не позволяет работать только со значениями времени, так как использует формат год/месяц/день/время. Следовательно, данный класс является более сложным, чем необходимо. Поэтому мы и займемся разработкой класса `Time`.

Создание класса Time

При разработке класса мы будем учитывать следующие элементы:

- переменные, которые включены в его объект;
- его конструкторы;
- методы, требующиеся программам, которые могут его вызывать.

Для решения стоящих перед нами задач нет необходимости включать в программу дополнительные переменные, кроме используемых для хранения значений часов и минут. Конечно, класс можно расширить и для работы со значениями времени, выраженными в секундах, но это приведет к возникновению дополнительных проблем.

Вызывают интерес конструкторы. Нам необходимо передать значения часов и минут в класс. Сам класс может, если есть необходимость, вывести значения часов и минут из данных любого формата. Это могут быть:

- два целых числа — часы и минуты (11, 30);
- одно четырехразрядное целое число, у которого первые два разряда интерпретируются как значение часов (1130);
- вещественное число, у которого дробная часть интерпретируется как значение минут (11.30).

Конструкторы могут быть созданы для любого из указанных форматов значений времени. Одно значение времени может быть получено на основе другого, что даст нам четвертый из показанных ниже конструкторов. Рассмотрим конструкторы:

```
class Time {
  int hour;
  int min;

Time (int h, int m) {
  hour = h;
  min = m;
}

Time (int mins) {
  hour = mins / 60;
  min = mins % 60;
}

Time (double t) {
  hour = (int) t;
  min = (int) t*100 - hour*100;
}

Time (Time t) {
  hour = t.hour;
  min = t.min;
}
... и т. д.
```

На рис. 3.7 показана диаграмма моделирования для класса `Time` вместе с некоторыми объектами, созданными с использованием различных конструкторов.

В первую очередь должен быть создан метод `addTime`. Нам предстоит определить, как будет выглядеть этот метод в классе и как он будет вызываться. По-видимому, это будет делаться совершенно новым способом, не похожим на те, которые мы рассматривали ранее. Мы уже видели, что когда класс инициализируется при создании объекта, метод вызывает этот объект, используя оператор с точкой. В нашем примере (мы прибавляем минуты) данным объектом является `time`. Поэтому вызов

```
time = addTime(time,15)
```

примет следующий вид:

```
time.addTime(15)
```

Изменение данных объекта производится путем добавления к ним 15 минут. Исходя из этого, мы можем сделать вывод о том, что метод `addTime` должен иметь тип `void`. Добавление минут упрощается за счет того, что мы имеем две отдельные части значения времени:

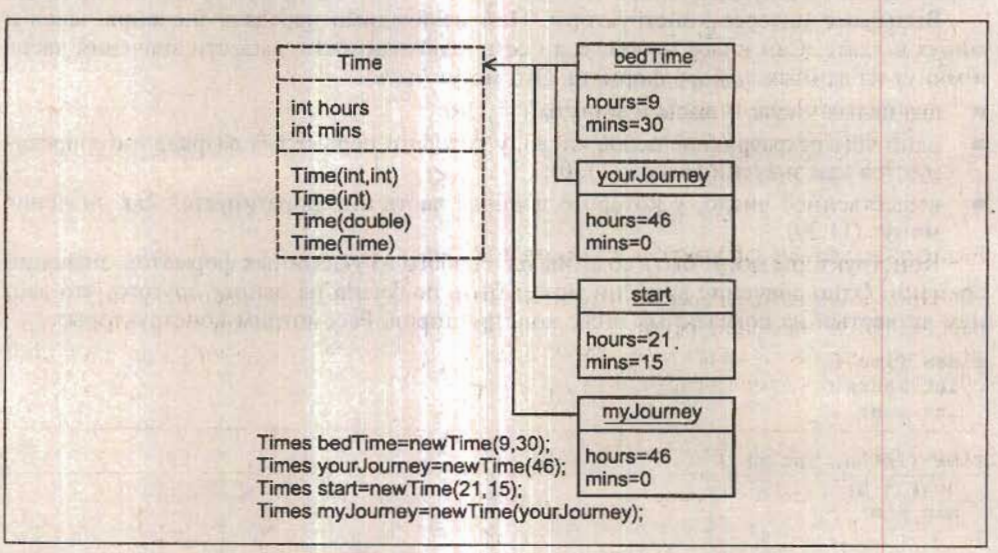


Рис. 3.7. Конструкция некоторых объектов класса `Time`

```
void addTime (int m) {
  min += m;
  hour+= min / 60;
  min %= 60;
}
```

Метод эффективно использует специальный оператор присваивания, который изменяет значение переменной в левой части. Соответственно, первый оператор можно прочитать следующим образом: "Прибавить `m` к значению переменной `min`" или "переменная `min` принимает значение `min` плюс `m`".

Какие еще методы понадобятся? Возможно, потребуется отобразить время, и мы можем создать метод `write`, однако вообще включать операторы вывода в общие

классы, такие как Time, не принято. Вместо этого мы должны определить метод toString, который может быть автоматически вызван методом println, как описано в разделе 3.1. В класс могут быть включены и другие методы, но только когда мы решим, как следует тестировать класс Time в нашей программе.

Тестирование класса

Один из лучших способов протестировать новый класс — использовать его в программе, которая уже работает. Для этой цели мы переделаем нашу первую программу PizzaDelivery. Изменения в ней необходимо сделать в тех местах, где указывается время. Эти фрагменты выделены подчеркиванием:

```
PizzaDelivery1 () {
 .. пропущенные строки
 for (int time = 900; time < 1300; time=addTime (time, 15))
     printaLine(time, area);
 }
}

void printaLine (int t, int a) {
  System.out.print(t+"\t");
  for (int queue = 0; queue <= 5; queue++)
    System.out.print(addTime(t,
      queue*makeTime + processTime + a*driveTime) + "\t");
  System.out.println();
}
```

Использование объектов в цикле for

Прежде всего мы переименуем программу, чтобы иметь в распоряжении обе ее версии. Затем надлежит решить, как следует перебирать значения времени в цикле, если они являются объектами. Оказывается, цикл for — намного более мощное средство, чем обычный счетчик целых чисел.

Цикл for

```
for (start; check; update) {
   body
 }
```

start и update представляют собой определенные значения, а check — это некоторое условие, зависящее от значений переменных start и update.
Цикл начинается с выполнения оператора start, затем проверяется условие check. Если условие истинно, выполняется тело метода body, после чего управление опять передается операторам update и check. Когда последний возвращает значение "ложь", происходит выход из цикла.

Другими словами, начальный оператор не должен присваивать значение целочисленной переменной: он может быть инициализирован. Для начала давайте определим два объекта времени, такие как:

```
Time open =  new Time (9.00);
Time close = new Time (13,0);
```

Здесь мы сознательно использовали вызов двух разных конструкторов, чтобы сделать особое ударение на их применении. Первый пересылает вещественное число, а второй использует два параметра. Сейчас начало цикла является простым:

```
Time ofDay = open
```

Здесь `ofDay` — это переменная цикла. Преобразованная часть цикла будет иметь следующий вид:

```
ofDay.addTime(15)
```

В данном случае, если мы устанавливаем условие таким образом:

```
OfDay < close // не сработает
```

оно не будет действительным. В предыдущем разделе указывалось, что мы не можем использовать условные операторы, такие как <, при работе с объектами. Условие должно быть определено и выполнено как метод, так что его вызов будет следующим:

```
ofDay.lessThan(close)
```

где `lessThan` возвращает булево значение на основании условия, которое его генерирует. Мы можем сейчас записать полный цикл вместе с его телом:

```
for (Time ofDay = open; ofDay.lessThan(close);
  ofDay.addTime(15)) {
  printaLine(ofDay, area);
}
```

Как и в любом другом цикле `for`, объект `ofDay` сначала принимает значение `open`, а затем начинается выполнение следующего цикла:

- проверка, достиг ли объект `ofDay` конечного значения `close`;
- выполнение тела цикла;
- изменение значения параметра цикла на 15 минут путем вызова метода `addTime`.

На рис. 3.8 показаны объекты цикла `for`. Каждый раз при вызове метода `ofDay.addTime` в цикле к объекту `ofDay` добавляются 15 минут. На рисунке видно, что этот объект идентичен объекту `open`. Следовательно, последний также будет изменяться. В этом случае нам придется обновлять его каждый раз при формировании таблицы для нового района. Но вместо этого нам необходимо создать копию объекта `open` для объекта `ofDay`, что легко выполнить с помощью нашего четвертого конструктора. В результате цикл примет вид:

```
for (Time ofDay = new Time(open); ofDay.lessThan(close);
       ofDay.addTime(15)) {
  printaLine(ofDay, area);
}
```

Рис. 3.8 служит прекрасным примером того, как с помощью диаграммы можно наглядно представить задачу, которая впоследствии будет решена средствами языка Java.

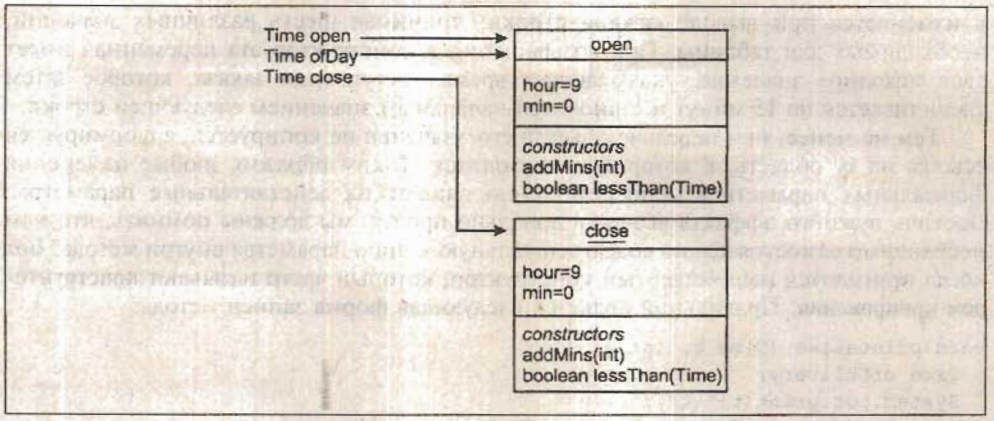


Рис. 3.8. Диаграмма моделирования цикла `for`

Объекты в роли параметров

При переходе к методу `printaLine` вызов метода `addTime` достаточно прост, а соответствующий цикл `for` выглядит так:

```
for (int queue = 0; queue <= 5; queue++) {
  t.addTime(queue*makeTime + processTime + a*driveTime;
  System.out.print(t + "\t");
}
```

Если вставить оператор вывода в этот цикл и задать четыре значения времени, с которыми ведется работа, мы получим ряд значений переменных, относящихся к двум вложенным циклам — одному в конструкторе и одному в методе `printaLine`.

open	close	ofDay	t	queue
09:00	**13:00**	**09:00**	**09:00**	0
09:00	13:00	09:15	**09:15**	0
09:00	13:00	09:35	**09:35**	1
09:00	13:00	10:00	**10:00**	2
09:00	13:00	10:30	**10:30**	3
09:00	13:00	11:05	**11:05**	4
09:00	13:00	11:45	**11:45**	5
09:00	**13:00**	**12:00**	**12:00**	0
09:00	13:00	**12:15**	**12:15**	1

В результате выполнения этого цикла и запуска программы на экране отобразится следующая часть таблицы:

```
9:0     9:15    9:35    10:0    10:30    11:5    11:45
12:0    12:15   12:35   13:0    13:30    14:5    14:45
```

Данная таблица является неправильной. В начале второй строки должно стоять значение 9:15, а не 12:00. Значение 12:00, очевидно, было принято объектом `ofDay`, действительным параметром метода `printaLine`, когда им заменили `t` (формальный параметр).

Здесь мы можем видеть, как происходит передача объектов в качестве параметров и чем она отличается от передачи значений простых типов. В программе `PizzaDelivery1`, когда методу `printaLine` передавалось целое значение времени, он получал копию значения и мог работать с ней, не воздействуя на оригинал. Переменная

t изменяется при выводе каждой строки, принимая шесть различных значений, необходимых для таблицы. При возвращении в конструктор эта переменная имеет свое исходное значение, указывающее время поступления заказа, которое затем увеличивается на 15 минут и становится начальным значением следующей строки.

Тем не менее, при передаче объекта его значение не копируется, а формируется ссылка на ту область, в которой он находится. Таким образом, любые изменения формальных параметров непосредственно влияют на действительные параметры. Достичь нужного эффекта все-таки довольно просто: мы должны помнить, что нам необходимо самостоятельно создать локальную копию параметра внутри метода. Вот когда пригодится наш четвертый конструктор, который часто называют **конструктором копирования**. Правильной является следующая форма записи метода:

```
void printaLine (Time t, int a) {
  Time ofDelivery;
  System.out.print(t+"\t");
  for (int queue = 0; queue <= 5; queue++) {
    ofDelivery = new Time(t);
    ofDelivery.addTime(queue*makeTime + processTime +
                       a*driveTime);
    System.out.print(ofDelivery + "\t");
  }
  System.out.println();
}
```

Каждый раз, когда мы хотим ввести новое значение в таблицу, мы создаем объект времени с копией переменной t, изменяем ее значение, выводим его на экран, а затем Java удаляет ее за нас, когда метод ofDelivery снова принимает значение t. Выводимая на экран последовательность теперь имеет следующий вид:

open	close	ofDay	t	ofDelivery	queue
09:00	**13:00**	**09:00**	**09:00**	null	0
09:00	13:00	09:00	09:00	**09:15**	0
09:00	13:00	09:00	09:00	**09:20**	1
09:00	13:00	09:00	09:00	**09:25**	2
09:00	13:00	09:00	09:00	**09:30**	3
09:00	13:00	09:00	09:00	**09:35**	4
09:00	13:00	09:00	09:00	**09:40**	5
09:00	**13:00**	**09:15**	**09:15**	null	
09:00	13:00	09:15	09:15	**09:30**	0
09:00	13:00	12:15	09:15	**09:35**	1

Проверка показала, что данная таблица является правильной.

Наконец, мы объединим все методы класса и протестируем программу.

Программа 2

```
class PizzaDelivery2 {

  /* Программа доставки пиццы, версия 2, Дж. М. Бишоп
   * ================================
   *
   * Обрабатывает значения времени доставки пиццы
   * с учетом длины очереди.
   *
   * Иллюстрирует разработку класса Time и использование
```

```
 * его объектов в методах и циклах for.
 */

int driveTime = 5;     //для района
int makeTime = 5;      //в очереди
int processTime = 10; //для заказа

PizzaDelivery2 () {
  for (int area = 1; area<= 1; area++) {
   printHeading(area);
   startTable();
   Time open = new Time (9.00);
   Time close = new Time (13.0);
   for (Time ofDay = open; ofDay.lessThan(close);
          ofDay.addTime(15)) {
       printaLine(ofDay, area);
   }
  }
 }

 void printaLine (Time t, int a) {
   Time ofDelivery;
   System.out.print(t+"\t");
   for (int queue = 0; queue <= 5; queue++) {
     ofDelivery = new Time(t);
ofDelivery.addTime(queue*makeTime+processTime+a*driveTime);
System.out.print(ofDelivery + "\t");
   }
   System.out.println();
 }

class Time {
  int hour;
  int min;

  Time (int h, int m) {
    hour = h;
    min = m;
  }

  Time (int mins) {
    hour = mins / 60;
    min = mins % 60;
  }
  Time (double t) {
    hour = (int) t;
    min = (int) t*100 - hour*100;
  }

  Time (Time t) {
    hour = t.hour;
    min = t.min;
  }

  void addTime (int m) {
    min +=m;
    hour+= min / 60;
```

```
      min %= 60;
    }

    boolean lessThan (Time t) {
      return hour < t.hour | (hour==t.hour & min < t.min);
    }

    public String toString () {
      return hour + ":" + min + " ";
    }
  }

  void printHeading (int area) {
    System.out.println("Pizza 2U Delivery estimates for "+"Area
                       "+area);
    System.out.println("========================== ================"+
  "=====\n");
}

void startTable() {
  System.out.println("Time now          Queue length");
  System.out.print("           ");
  for (int queue = 0; queue <=5; queue++)
    System.out.print(" "+queue+" ");
  System.out.println();
}

public static void main (String [] args) {
  new PizzaDelivery2 ();
 }
}
```

Ниже представлены данные, выводимые на экран в результате выполнения программы:

```
Pizza 2U Delivery estimates for Area 1
======================================

Time now    Queue  length
            0      1      2      3      4      5
9:0         9:15   9:20   9:25   9:30   9:35   9:40
9:15        9:30   9:35   9:40   9:45   9:50   9:55
9:30        9:45   9:50   9:55   10:0   10:5   10:10
9:45        10:0   10:5   10:10  10:15  10:20  10:25
10:0        10:15  10:20  10:25  10:30  10:35  10:40
10:15       10:30  10:35  10:40  10:45  10:50  10:55
10:30       10:45  10:50  10:55  11:0   11:5   11:10
10:45       11:0   11:5   11:10  11:15  11:20  11:25
11:0        11:15  11:20  11:25  11:30  11:35  11:40
11:15       11:30  11:35  11:40  11:45  11:50  11:55
11:30       11:45  11:50  11:55  12:0   12:5   12:10
11:45       12:0   12:5   12:10  12:15  12:20  12:25
12:0        12:15  12:20  12:25  12:30  12:35  12:40
12:15       12:30  12:35  12:40  12:45  12:50  12:55
12:30       12:45  12:50  12:55  13:0   13:5   13:10
```

Резюме

Данная глава содержит очень ценную информацию. В ней рассматривается операция присваивания, описана организация циклов, дан обзор простых типов данных и выражений. Основное внимание уделено трем из шести численных типов — `int`, `long` и `double`, а также логическому типу `boolean`. Затрагивалась также тема вывода на экран целых и вещественных чисел средствами Java. Обсуждение данной темы будет продолжено в главах 4 и 7.

Цикл `for` является одним из мощнейших средств языка Java, поэтому мы рассмотрели все возможные варианты его использования. Изучая методы, мы особое внимание уделяли тому, как происходит их объявление (вызов, задание формальных и действительных параметров). Теоретический материал подкреплен рядом примеров из реальной жизни, а завершает главу текст программы, предназначенной для определения времени доставки пиццы. В этом примере продемонстрированы способы использования объектов в циклах, а также рассмотрены вопросы управления объектами.

В главе раскрывались понятия, связанные со следующими ключевыми словами:

```
boolean     int
break       long
byte        return
double      short
false       static
float       true
for         void
```

Представленные ниже ключевые слова упоминались, но подробно будут рассмотрены позже:

```
char
final
while
```

Глава содержит описания таких объектов и методов из библиотеки языка Java:

```
Math            round
PI              abs
pow             max
sqrt            min
atan            String
cos             Calendar
sin             compareTo
tan             compareToIgnoreCase
toDegrees       equals
toRadians       equalsIgnoreCase
random          Date
```

Следующий перечень включает темы, на которых в главе сделано особое ударение:

- некоторые методы класса `Math`;
- приведение типа;
- простой оператор `for`;
- объявление простого метода типа `void`;
- вызов простого метода типа `void`;
- объявление формальных параметров;

- действительные параметры;
- объявление метода типа `void`;
- объявление типизированного метода;
- вызов типизированного метода;
- приоритет операторов (для переменных типа `boolean`);
- метод сравнения строк;
- класс `Date` языка Java;
- цикл `for`.



Вопросы

3.1. Если мы выплачиваем одну треть нашего дохода в качестве налога после вычитания всех льгот, как будет выглядеть формула расчета суммы налога, представленная в виде выражения языка Java?

а) `1/3*income – rebate`

б) `1/3 * (income - rebate)`

в) `(income - rebate) / 3`

г) `income/3 - rebate`

3.2. Сколько звездочек отобразится в результате выполнения следующего цикла:

```
for (int star = 9; star < 0; star++) {
 System.out.print('*');
}
```

а) 9

б) 0

в) 10

г) 8

3.3. Какой оператор `for` может быть использован для организации цикла, в котором осуществляется последовательный переход от одного десятилетия двадцатого столетия к другому:

а) `for (int year = 1990; year <= 1990; year ++)`

б) `for (int year = 1990; year < 2000; year ++)`

в) `for (int year = 1990; year <= 1990; year +=10)`

г) `for (int year = 1990; year <= 1990; year +10)`

3.4. Если переменная `x` имеет тип `double` и принимает значение `-3.00000000008`, что выведет на экран функция `System.out.print(x)`?

а) `-3.00000000008`

б) `-3.0`

в) `-3.0E-11`

г) `-3.8E-10`

3.5. Какое ключевое слово (или слова) обязательно должно присутствовать в типизированном методе:

а) `void`

б) `return`

в) `break`

г) `public static void`

3.6. Если мы объявляем `String name = "John"`, выражение `name = = "John"` примет значение `false`. Почему?

а) должно быть =, а не = =

б) `name` и `"John"` являются разными объектами

в) переменная `"John"` должна состоять из 8 символов и, следовательно, иметь четыре пробела в конце

г) выражение должно быть заключено в круглые скобки

3.7. Если мы хотим проверить, имеют ли два объекта древностей `a` и `b` одинаковые имена и цены, не затрагивая переменную `description`, мы можем использовать выражение:

а) `a.name == b.name & a.price == b.price`

б) `(a.name == b.name) && (a.price == b.price)`

в) `a.name.equals(b.name) && a.price == b.price`

г) `a==b & (a.description != b.description || a.description == b.description)`

3.8. Выполняя присваивание `time = addTime(time,15)` в программе `PizzaDelivery1`, определите, как оно будет осуществляться в программе `PizzaDelivery2`, где существует класс `Time`, содержащий метод `addTime`.

а) `time = Time.addTime(time,15)`

б) `time.addTime(time,15)`

в) `time = addTime(15)`

г) `time.addTime(15)`

3.9. Получить случайное число из диапазона 1—6 позволяет выражение:

а) `number = Random(6)+1`

б) `number = Math.random(6)`

в) `number = (int) (Math.random()*6 + 1)`

г) `number = Math.random()*6`

3.10. Если у нас есть два целых числа, которые возвращаются методом, мы можем использовать:

а) объект в качестве параметра

б) объект в качестве возвращаемого значения

в) два параметра

г) ответ а или б

Задачи

Примечание. Задачи 3.2 и 3.4 по объему аналогичны рассмотренному примеру о доставке пиццы. Для их решения потребуется применить все полученные навыки. Остальные задачи связаны с циклами и методами.

3.1. **Большая диаграмма.** Нарисуйте диаграмму моделирования и диаграмму вызова (как на рис. 3.3) для программы `LargeTemperatureTable` из примера 3.6.

3.2. **Участники марафона.** Существует пять основных кандидатов на участие в марафоне, который проводится в Саванне. Они приехали из Англии, Германии, Италии, Испании и Норвегии. Организаторам гонок должна быть предоставлена следующая информация о каждом бегуне: имя, страна, возраст и лучший результат, показанный участником до настоящего времени (в часах и минутах).

- Создайте класс, описывающий участника марафона. Вы можете использовать класс `Time` для представления лучшего показателя времени. Обычно время пробега составляет около 2 часов.
- Напишите короткую программу, чтобы проверить, правильно ли созданы пять объектов упомянутого выше класса для бегунов.
- Теперь напишите программу `Race`, которая создаст объекты для бегунов, сгенерирует случайным образом время пробега для каждого из них, а затем выведет список бегунов с подробными их описаниями, включая результаты пробега и лучший показатель. (Используйте для этого метод `Math.min`.)

Хотя на данном этапе мы не можем с легкостью вычислить победителя пробега, мы вернемся к этой задаче в главе 4.

3.3. **Контроль веса.** Два друга на протяжении нескольких недель пытались сбросить вес. У первого начальный вес составлял 100 кг, а объем талии — 98 см. Параметры второго — 85 кг и 95 см. Каждую неделю они фиксировали свои параметры и вычисляли, у кого эти показатели уменьшились значительнее.

- Разработайте и запрограммируйте класс для записи значений веса человека и объема его талии.
- Проверьте, как работает этот класс, создав объекты для двух людей.
- Напишите программу, которая в течение шести недель раз в неделю будет генерировать новые значения веса и объема талии. Значения эти должны с небольшими отклонениями соответствовать текущим (например, текущий вес +/− 1.5 кг, текущий объем талии +/− 2 см). Выведите новые значения и сохраните их в объектах. Следует также выводить информацию о том, кто из двоих друзей сбросил больший вес в каждом случае.
- Вычисляйте каждый раз общее значение сброшенного веса. В завершение выведите данные об общем сброшенном весе у двух друзей.

3.4. **Часовые пояса.** Компания, занимающаяся электронной коммерцией и расположенная в Лондоне, Великобритания, имеет клиентов в разных точках мира. Иногда ее сотрудникам приходится оформлять заказы по телефону. В таких случаях необходимо знать, какое время сейчас в определенном часовом поясе. Лондон работает по всеобщему времени (ВВ). Выведите таблицу со значениями времени от 0:00 до 23:00 для городов Ванкувер (−8 часов от ВВ), Нью-Йорк (−5 часов), Кейптаун (+2) и Сидней (+10). Используйте класс `Time`, а также метод `addTime` для вычисления значений времени в других часовых поясах.

3.5. **Преобразования.** Модифицируйте программу из примера 3.3 так, чтобы она выполняла преобразование миль в километры (1 миля = 1.6 км), а также долларов в грацы ($1 = G0.45).

3.6. **Другие таблицы.** Программу, которую мы использовали для печати таблицы преобразований, модифицируйте так, чтобы она позволяла выводить на печать таблицы для выполнения других преобразований, например:

— градусов Цельсия в градусы Фаренгейта;
— миль в километры;
— литров в галлоны;
— долларов в национальную валюту.

Для выполнения преобразований используйте типизированные методы.

3.7. **Голосование.** В административный совет входят три директора, за каждым из которых закреплен переключатель с положениями да/нет. Когда проводится голосование, загорается лампочка, если большинство проголосовало "за". Цепь, которая обеспечивает ее включение, можно представить в виде булевой функции:

```
L = a&(b|c)|b&c
```

Напишите программу, которая выводит таблицу, содержащую альтернативные значения да/нет для *a*, *b* и *c*, а также значения *L*, используя метод, возвращающий булево значение.

3.8. **Точное время.** Перед проведением марафона было условлено, что время будет фиксироваться с точностью до доли секунды. Дополните класс `Time` так, чтобы он учитывал значения в секундах и долях секунды (вы можете использовать переменную, имеющую тип `double`). Протестируйте класс с помощью небольшой программы.

3.9. **Зеленые бутылки.** Походная песня, которая может быть продолжена, начинается такими строками:

На стене висело десять зеленых бутылок,
Десять зеленых бутылок висело на стене,
Но если одна зеленая бутылка случайно упадет,
То девять зеленых бутылок останутся висеть на стене.
Девять зеленых бутылок висело на стене,
Но если одна зеленая бутылка случайно упадет,
То восемь зеленых бутылок останутся висеть на стене
и т. д.

... Одна зеленая бутылка висела на стене,
Но если одна зеленая бутылка случайно упадет,
На стене не останется ни одной зеленой бутылки.

Используя обратные вложенные циклы, рассмотренные в разделе 3.2, составьте алгоритм для вывода такой песенки, начиная с момента, когда осталось пять бутылок.

3.10. **Наклейки.** Переделайте программу из примера 3.2 таким образом, чтобы она обеспечивала вывод на печать трех этикеток с вашим именем и адресом, которые расположены поперек страницы. Необходимо создать восемь дубликатов, чтобы вся страница была заполнена.

3.11. **Наглядные таблицы.** Таблица преобразования в примере 3.6 является не очень удобной, поскольку значения увеличиваются по горизонтали, а не по вертикали.

Подумайте, как изменить программу так, чтобы значения увеличивались по вертикали.

3.12. **Расписание.** Удобно иметь под рукой бланк расписания, где можно фиксировать время проведения лекций. Напишите программу, которая выводила бы на печать расписание, в верхней строке которого перечислены дни недели, а слева (сверху вниз) указаны значения времени от 8.00 до 15.00.

3.13. **Гистограмма.** Проверьте, сможете ли вы создать класс на основе примера 3.4, который включает методы, рисующие линию и ось, и может быть вызван для вывода гистограммы, базирующейся на различных данных. Протестируйте его на следующих примерах:

а) генерирование случайных чисел, которые соответствуют значениям температур для данного месяца (например, от 10 до 30 градусов Цельсия);

б) установка соответствия между значениями функции $\sin x$ и значениями температур от 0 до 90°C.

3.14. **Дни рождения.** Вероятность того, что у двух людей из группы, состоящей из n человек, совпадают дни рождения, составляет:

$$p(n) = 1 - \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \ldots \times \frac{364 - n + 1}{365}$$

Напишите программу, которая определяет такую вероятность и выводит соответствующие значения для групп из 2-х и 60-ти человек. Создайте таблицу значений n и $p(n)$, исходя из того, что n принимает значения от 10 до 50. Если вы выполнили задание 3.13, нанесите на гистограмму значения для групп из 10, 20, ..., 50 человек.

3.15. **Цифровой треугольник.** Напишите программу, которая использует операторы `for`, а также `print/println` для выводя на экран следующего треугольника. Введите количество линий (пять, как показано ниже).

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Измените программу так, чтобы в треугольнике все числа выравнивались по центру. В результате мы получим треугольник Паскаля.

```
    1
   2 2
  3 3 3
 4 4 4 4
5 5 5 5 5
```

3.16. **Ряды Фибоначчи.** Так называются ряды чисел, в которых каждое последующее число является суммой двух предыдущих:

```
1  1  2  3  5  8  13  21  34  55
```

Напишите программу, выводящую на экран первые 50 значений ряда. Используя вложенные циклы, измените программу так, чтобы она выводила только каждое третье число. Какую закономерность вы заметили?

3.17. **Таблица умножения.** Давным-давно на обложках тетрадей печаталась таблица умножения:

```
4 times table     5 times table     6 times table
1 x 4 = 4         1 x 5 = 5         1 x 6 = 6
2 x 4 = 8         2 x 5 = 10        2 x 6 = 12
3 x 4 = 12        3 x 5 = 15        3 x 6 = 18
4 x 4 = 16        4 x 5 = 20        4 x 6 = 24
5 x 4 = 20        5 x 5 = 25        5 x 6 = 30
6 x 4 = 24        6 x 5 = 30        6 x 6 = 36
7 x 4 = 28        7 x 5 = 35        7 x 6 = 42
8 x 4 = 32        8 x 5 = 40        8 x 6 = 48
9 x 4 = 36        9 x 5 = 45        9 x 6 = 54
10 x 4 = 40       10 x 5 = 50       10 x 6 = 60
11 x 4 = 44       11 x 5 = 55       11 x 6 = 66
12 x 4 = 48       12 x 5 = 60       12 x 6 = 72
```

Напишите программу, которая использует подходящие методы и параметры для печати всех двенадцати таблиц умножения.

3.18. **Выпускники.** Добавьте к классу `Child` из программы о детском саде (задача 2.8) метод, который возвращает значение, указывающее, выпускается ли ребенок из детского сада в этом году. Решите, где должно быть объявлено значение года и как должен осуществляться доступ к нему: как к константе в главной программе, которая передается в качестве параметра, как к статической переменной в классе или иначе.

3.19. **Манускрипт.** Измените программу из примера 2.10 ("Книга") так, чтобы класс записывал количество страниц манускрипта, и добавьте метод для вычисления окончательного количества страниц, принимая во внимание редукцию на 80%. Проверьте класс с помощью программы-теста, которая создает 15 глав со сгенерированным случайным образом количеством страниц от 20 до 50.

3.20. **Шахматная доска.** Шахматная доска состоит из восьми рядов, в каждом из которых чередуются восемь полей белого и черного цветов. Напишите программу, создающую изображение шахматной доски, большую часть которой составляет метод `paint` с двумя вложенными циклами и двумя вызовами метода `fillRect`.

ГЛАВА 4



Ввод и вывод

В этой главе мы досконально изучим процессы ввода и вывода данных, а также познакомимся с двумя классами, которые предоставляют в распоряжение пользователей простые и в то же время мощные средства взаимодействия с программой. Чтобы применять классы ввода-вывода (ВВ) при работе как с экраном, так и с файлом, необходимо знать принципы создания и использования пакетов, знакомство с которыми также состоится в настоящей главе.

4.1. Интерактивный ввод данных

Программе часто приходится запрашивать извне значения для своих переменных. Необходимые программе данные могут извлекаться из таблиц, хранящихся на диске, а также представлять собой ответы на вопросы или списки значений. Термин **поток** применяется для обозначения последовательностей данных, которые поступают из одного источника (например, с клавиатуры или из файла на диске). Мы сначала рассмотрим, как осуществляется ввод данных в программу с клавиатуры, а затем перейдем к изучению методов ввода данных из файла и вывода данных в файл.

Потоки ввода

В языке Java эквивалентом объекта `System.out` является объект `System.in`. Хотя объект `System.in` требует предварительного объявления, его классом является **абстрактный** класс `InputStream`. Подробные сведения об абстрактных классах изложены в главе 9, а на данном этапе достаточно запомнить то, что не всегда можно создать объекты, относящиеся к ним. Объект `System.in` должен передаваться в качестве параметра конструктора другому классу, который называется `InputStreamReader`, в результате чего создается объект. Кроме того, следует отметить, что ввод данных в программу с клавиатуры удобнее всего осуществлять через буфер, так что после создания объекта его следует передать классу `BufferedReader`.

Это довольно сложный способ обработки вводимых с клавиатуры данных, но, как показано ниже, его применение в процессе программирования часто является целесообразным.

Объявление потока для ввода с клавиатуры

```
import java.io.*;

BufferedReader stream = new BufferedReader
            (new InputStreamReader(System.in);
```

Передает пакет `java.io`. Создает объект `InputStreamReader` на основе объекта `System.in` и использует его для создания объекта `BufferedReader`, которому присваивается идентификатор *stream*.

Сначала выполняется импорт пакета `java.io`, в котором определены три указанных выше класса. Затем создается новый поток, связанный с объектом `System.in`, который обладает всеми свойствами класса `BufferedReader`. Стандартное объявление имеет следующий вид:

```
BufferedReader in = new BufferedReader
            (new InputStreamReader(System.in));
```

Здесь `in` определяется как поток, который связан с клавиатурой. В разделе 4.5 мы увидим, как присоединять потоки к файлам.

В процессе чтения часто могут возникать ошибки, связанные с внезапным завершением данных или наличием у них неправильного формата. Такие события называют **исключениями**. Далее в главе рассказывается, как вести себя при их возникновении. Java требует, чтобы в каждом методе указывались все возможные исключения. Поэтому, прежде чем приступить к считыванию данных, необходимо добавить команду

```
throws IOException
```

после объявления каждого метода, в котором осуществляется чтение данных, и любого метода, который вызывает его и не обрабатывает исключения (см. пример 4.1).

Чтение строк

Подобно методу `println`, класс `BufferedReader` предоставляет в распоряжение пользователей метод `readLine`, который присваивает строку вводимого с клавиатуры текста строковой переменной.

Чтение строки
string = *stream*.readLine();
Получает строку данных типа `string` из потока *stream* и присваивает ее строке *string*.

Обратите внимание на два различия между этой конструкцией и конструкцией вывода:

- используется суффикс `-Line`, а не `-ln`;
- метод имеет тип, а не является `void`-методом, поэтому он вызывается с использованием процедуры присваиваниях, а не сам по себе.

Пример 4.1. Приветствие

Задача. Необходимо обеспечить выдачу приветствий на русском языке.

Решение. Напишите программу со встроенным приветствием и попросите пользователей ввести их имена.

Программа. Программа довольно проста. Мы должны установить связь с клавиатурой, а также добавить возможность генерирования исключения `IOException` в конструктор и метод `main`.

```
import java.io.*;

class Greetings {
```

```
/* Простая программа приветствия Дж.М.Бишоп, октябрь, 1996 г.
 * ----------------------------- Java 1.1, декабрь, 1997 г.
 *                               обновлена в мае 2000 г.
 */
Greetings () throws IOException {

  BufferedReader in = new BufferedReader
          (new InputStreamReader(System.in));

  System.out.print("Как Вас зовут? ");
  String name = in.readLine();
  System.out.println("Здравствуйте, " + name);
}

public static void main (String args []) throws IOException {
  new Greetings();
}
}
```

Тестирование. В результате выполнения программы формируется следующий текст (обычным шрифтом набраны вводимые данные, а полужирным выводимые):

```
Как Вас зовут?
Pierre Marchand
Здравствуйте, Pierre Marchand
```

Помните о том, что команда `readLine` позволяет считывать данные, вводимые с клавиатуры до тех пор, пока не будет нажата клавиша [Return].

Подход к чтению чисел

Java может с легкостью читать значения других типов, если они представлены в двоичной форме и хранятся в файле. Тем не менее, если мы хотим взаимодействовать с программой посредством клавиатуры, Java предлагает нам прочитать строку и сделать преобразование в программе явным образом. Стандартные средства преобразования данных распространяются с классами объектов, связанными с простыми типами. Подобные классы также содержатся в пакете `java.text` (см. главу 7). Эти классы могут также использоваться для форматирования при выводе данных.

Данные классы являются мощным средством, однако они сложны в применении, что досадно, поскольку чтение числа принадлежит к категории часто выполняемых операций. Вот как выглядит на языке Java операция чтения вещественного числа:

```
double d = Double.valueOf(in.readLine()).doubleValue();
```

Указанных сложностей можно избежать, реализовав соответствующие методы для работы с числами различных типов. Впоследствии можно собрать эти методы и создать класс, который выполнял бы необходимые функции.

Справедливости ради следует сказать, что классы `java.text` хорошо подобраны и позволяют вводить наряду с числами даже такие данные, как обозначения валют и символ процентов.

4.2. Дружественный интерфейс ввода-вывода

На основе сказанного можно сделать вывод о том, что вводом чисел в Java управлять сложнее, чем выводом. Все средства языка Java, предназначенные для ввода

данных (см. главу 7), являются довольно мощными, и нам придется смириться с тем, что данный язык программирования не предоставляет более простой альтернативы. Использование необработанного ввода может отвлечь читателя от нашей основной цели — изучения программирования, а также от решения реальных задач. По этой причине мы разработали два класса, которые будут широко использоваться в книге:

- `Display` — обобщенный пользовательский интерфейс, позволяющий осуществлять ввод и вывод данных в одно и то же время;
- `Stream` — простые средства ввода-вывода текстовых данных с помощью консоли языка Java, командного окна или из текстового файла.

Первый класс написан специально для недавно вышедшей в свет книги "Java Gently for Engineers and Scientist" из данной серии. Он предоставляет легкое в использовании средство управления графическим пользовательским интерфейсом. Второй класс основан на классе `Text`, который был написан в 1997 году для первого издания книги *Java Gently* и успешно использовался в программах тысячами пользователей. При подготовке настоящего издания этот класс был существенно модифицирован с учетом особенностей объектно-ориентированного программирования. Следует отметить, что при этом был сохранен простой интерфейс, благодаря которому класс стал таким популярным. Новый класс получил название `Stream`. Класс `Text`, его предшественник, по-прежнему можно применять в программах.

Класс Display

Обратите внимание на интерфейс класса `Display`, позволяющий многократно вводить пару чисел (рис. 4.1). Он состоит из двух панелей: левая служит для ввода данных , а правая — для вывода. Обе панели отображаются на экране одновременно. Каждая из них снабжена полосами прокрутки, обеспечивающими доступ к информации, которая не поместилась на панели. В нижней части окна находятся две кнопки: **Готово** и **Закрыть**. Первая обеспечивает взаимодействие между пользователем и программой, а вторая позволяет завершить работу программы и закрыть окно.

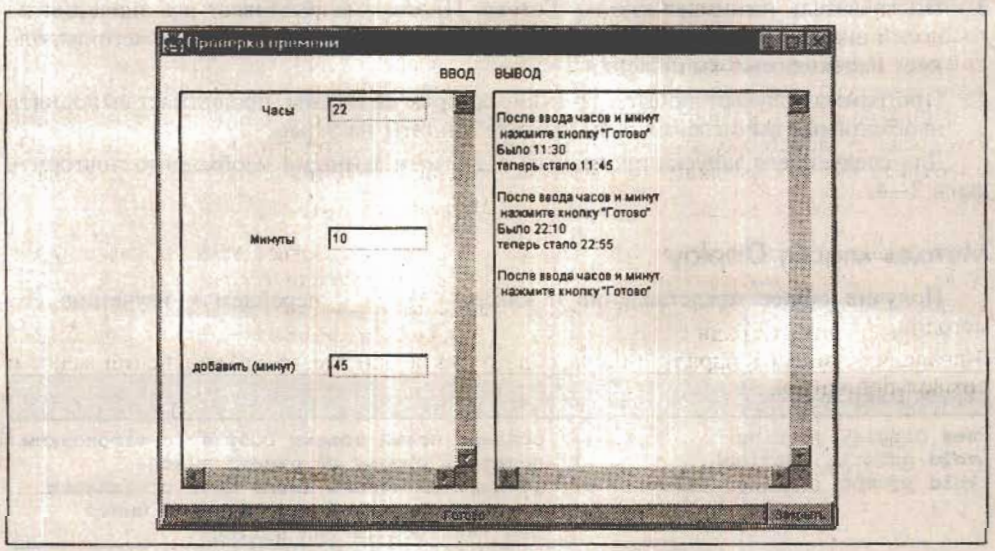


Рис. 4.1. Интерфейс, формируемый простой программой

Конечно, можно было бы с самого начала написать программу, генерирующую такое окно (мы научимся это делать в главе 10). Но чтобы это сделать, нам придется довольно долго программировать пользовательский интерфейс, в результате чего простые программы превратятся в монстров.

Панель вывода

Как показано на рис. 4.1, программа выполняет свою работу в три этапа. Перечислим их:

- Отображение сообщения о необходимости ввести данные.
- Вывод информации о действиях, которые должен выполнить пользователь.
- Считывание значений, введенных пользователем, выполнение вычислений и отображение результата.

Указанные действия повторяются программой до тех пор, пока не будет нажата кнопка **Закрыть**.

Панель ввода

На панели ввода отображаются данные, которые пользователь может изменять в случае необходимости. Данная панель применяется для выполнения следующих операций:

1. Программа предоставляет сведения о том, какие данные необходимо изменить. Для каждого значения создается стандартное поле с названием. Начальные значения могут передаваться в поля, что является очень полезным при тестировании программы, поскольку пользователю не придется думать о новых данных для каждого поля.
2. По мере необходимости пользователь изменяет значения в полях. Не все значения требуют изменений. Значения можно изменять до тех пор, пока не будут введены правильные данные.
3. Пользователь нажимает кнопку **Готово**. Программа считывает все значения из полей окна и обновляет свою внутреннюю таблицу значений, после чего выполняет необходимые вычисления.
4. Программа получает любые значения, которые ей нужны, продолжает выполнять необходимые вычисления и выводит результаты на экран.

Для следующего запуска программы с другими данными необходимо повторить шаги 2—4.

Методы класса Display

Получив общее представление о классе `Display`, перейдем к изучению его методов.

Класс `Display`

```
new Display (String)          // создает новый объект Display с заголовком
void println (String)         // печатает строку на панели вывода
void prompt (String, value)   // создает на панели ввода поле с подписью
                                 и начальным значением, которое имеет
                                 тип int, double или string
```

```
void ready (String)          // выводит сообщение о том, что программа
                                ожидает нажатия кнопки Готово
int getInt (String)          // получает значение типа int,
                                соответствующее метке, указываемой
                                в качестве параметра
double getDouble (String)    // считывает значение типа double,
                                соответствующее метке, указываемой
                                в качестве параметра
String getString (String)    // получает значение типа string,
                                соответствующее метке, указываемой
                                в качестве параметра
```

Программа может быть написана с использованием только первых пяти методов. Последние два — `getDouble` и `getString` — являются вариациями метода `getInt`.

Применение класса Display

При использовании класса `Display` первым шагом является создание объекта этого класса. Объявление имеет такой вид:

```
Display display = new Display ("Проверка времени");
```

В данной конструкции с маленькой буквы начинается имя объекта (`display`), а с большой — имя класса (`Display`). Это наша единственная ссылка на имя класса, так что в дальнейшем путаницы не будет. Конечно, мы могли бы дать объекту другое имя, например:

```
Display myWindow = new Display ("Проверка времени");
```

или

```
Display d = new Display ("Проверка времени");
```

В любой программе может генерироваться более одного окна, но необходимость в этом возникает крайне редко. Таким образом, целесообразно присвоить окну стандартное имя и употреблять его во всех программах. Мы выбрали имя `display`. Еще одно соглашение связано с применением `i` в качестве переменной цикла: это распространенный, хотя и не очень выразительный идентификатор.

Параметром, который передается классу `Display` при создании объекта, является заголовок окна, который отображается в его верхней строке (рис. 4.1).

Следующий этап — отображение начальных инструкций на панели вывода. Класс `Display` снова использует хорошо известный нам метод `println`, а также средства, с которыми мы познакомились в разделах 2.4 и 3.1. Для панели, изображенной на рис. 4.1, `println` может иметь следующий вид:

```
display.println("Test data set number " + i);
```

Единственное различие между указанным оператором и текстовым заключается в том, что строка передается нашему окну, а не методу `System.out` (для которого характерна специфическая и где-то даже старомодная связь с окном консоли или окном DOS).

Ввод данных

Обсудим, как осуществляется ввод данных с использованием интерфейса класса `Display`. Каждый элемент данных идентифицируется меткой, которая выводится рядом с ним. Поместить текстовое поле на панель ввода позволяет метод `prompt`:

```
display.prompt("Часы",11);
```

Начальным является значение 11, однако мы можем использовать также запись

```
display.prompt("Часы", 0);
```

или

```
display.prompt("Часы", a);
```

В последнем случае значение хранится в переменной a. Программа просит пользователя указать, что данные введены, вызвав метод ready, как показано в следующем примере:

```
display.ready("После ввода нажмите кнопку Готово");
```

В результате на экране отобразится соответствующее сообщение, а кнопка **Готово** станет активной.

Для считывания данных из текстовых полей в первую очередь должна быть нажата кнопка **Готово**. Не забывайте об этом! Без нажатия кнопки программа не сможет определить, когда пользователь завершил вносить изменения. Затем осуществляется вызов одного из трех методов get (выбор метода зависит от того, данные какого типа требуется получить):

```
h = display.getInt("Часы");
```

Передаваемая в качестве параметра строка позволяет определить, какое из полей на панели ввода нам нужно. Объект display возвращает текущее значение и присваивает его переменной. Допускается любой порядок выбора значений, так что каждый из следующих операторов является правильным:

```
m = display.getInt("Минуты");
h = display.getInt("Часы");
```

Пример 4.2. Тестирование класса Time

Нам уже приходилось тестировать класс Time в процессе работы с программой, предназначенной для составления таблицы с информацией о времени доставки пиццы. Сейчас мы можем использовать класс Time в простой программе, чтобы протестировать его модернизированный вариант. Результаты работы программы представлены на рис. 4.1, а сама программа имеет следующий вид:

```
import javagently.*;

class TimeDisplay {

    TimeDisplay () {
      Time time, newTime;
      Display d = new Display ("Проверка времени");
      d.prompt("Часы",11);
      d.prompt("Минуты",30);
      d.prompt("добавить (минут)",15);
      for (; ;) {
        d.ready("\nПосле ввода часов и минут\n нажмите кнопку \"Готово\"");
        int hour = d.getInt("Часы");
        int min = d.getInt("Минуты");
        time = new Time (hour, min);
        d.println("Было " + time);
        time.addTime(d.getInt("добавить (минут)"));
        d.println("теперь стало " + time);
```

```
    }
  }
//Здесь следует разместить класс Time.
 class Time {
    int hour;
    int min;

    Time (int h, int m) {
      hour = h;
      min = m;
    }

    Time (int mins) {
      hour = mins / 60;
      min = mins % 60;
    }

    Time (double t) {
      hour = (int) t;
      min = (int) t*100 - hour*100;
    }

    Time (Time t) {
      hour = t.hour;
      min = t.min;
    }
    void addTime (int m) {
      min +=m;
      hour+= min / 60;
      min %= 60;
    }

    boolean lessThan (Time t) {
      return hour < t.hour | (hour==t.hour & min < t.min);
    }

    public String toString () {
      return hour + ":" + min + " ";
    }
 }

  public static void main (String [] args) {
     new TimeDisplay ();
  }
}
```

Программа создала нужное нам окно, а поля ввода находятся на стадии инициализации. Затем программа войдет в бесконечный цикл. Это не является проблемой, поскольку программа завершит работу, когда пользователь нажмет кнопку **Закрыть**, а не тогда, когда сочтет, что выполнила достаточное количество итераций. Далее идет последовательность `ready-get-println`, после чего выполняются все вычисления. Данная программа послужит моделью для многих интерактивных программ ввода-вывода, с которыми мы будем работать.

Обратите внимание на то, что мы импортируем пакет `javagently`, содержащий класс `Display`. Как установить этот пакет на пользовательском компьютере, рассказывается в следующем разделе.

Тестирование. Какие значения надлежит использовать для тестирования? Тестированию подлежит метод `addTime`. Нам необходим набор значений часов, а также минут, которые будут добавляться к ним. Местом, в котором может появиться ошибка, является граница между сутками. Таким образом, для тестирования программы имеет смысл выбрать следующие значения:

часы	минуты	добавить	ожидаемый результат
23	30	15	23:45
23	30	29	23:59
23	30	30	00:00
23	30	60	00:60

Сразу отметим, что класс `Time` не отвечает полностью предъявляемым требованиям, поскольку он неправильно осуществляет перевод времени по окончании суток. Если прибавить 30 минут к 23:30, получим 24:00. А теперь в качестве упражнения модифицируйте класс `Time` таким образом, чтобы он выдавал верные значения, если по времени — полночь.

Класс Stream пакета javagently

В `javagently` для выполнения ввода-вывода предназначен класс `Stream`. Он позволяет использовать одиннадцать методов: четыре обеспечивают доступ к функциям ввода-вывода, четыре — к функциям ввода и три — к функциям форматирования выводимых данных.

Класс `Stream`

```
Константы
-----------
 READ = 0,
 WRITE = 1;

 Конструкторы
 --------------
 Stream (InputStream filename)
 Stream (String filename, int how)

 Ввод
 ------
 int    readInt ()
 double readDouble ()
 String readString ()
 char readChar ()

 Вывод
 --------
 void println  - для объектов и переменных типа String, int, double, char
 void print - для объектов и переменных типа String, int, double, char
 void close()

 Вывод - методы класса
 ---------------------------
 String format (int number, int align)
 String format (double number, int align, int frac)
```

Первый конструктор используется, когда известен поток `InputStream`, а также для установления связи с клавиатурой.
Второй конструктор применяется для установления связи с файлами, а параметр *how* указывает, предназначен ли файл для чтения или записи данных.

Четыре метода ввода читают данные из объекта `Stream` и интерпретируют их в зависимости от указанного типа. Если этот тип данных является некорректным, метод перейдет к следующему элементу. Пункты должны быть разделены пробелом или некоторым знаком пунктуации.
Методы `println` и `print` работают так же, как и в классе `System.out`.
Метод `close` всегда вызывается для любого файла. Он позволяет предотвратить потерю данных, если файл открыт для записи.
Методы форматирования являются методами класса и поэтому вызываются как `Stream.format`.

Если мы хотим вводить данные с клавиатуры, мы должны использовать первый конструктор, который разработан специально для этой цели, например:

```
Stream in = new Stream (System.in);
```

Затем мы можем воспользоваться набором из четырех методов ввода. Методы `readInt` и `readDouble` возвращают значения по запросу. Метод `readString` не является аналогом встроенного метода `readLine`, поскольку все методы класса `Stream` позволяют выводить несколько значений в одной строке. Эта возможность проиллюстрирована в следующем примере, а также в примере 4.10. Метод `readChar` считывает один символ. Все четыре метода ввода по умолчанию рассчитывают на то, что элемент данных, подлежащий чтению, отделен от других элементов пробелом, табулятором, запятой или символом конца строки. К сожалению, это означает, что мы можем успешно считывать данные не во всех случаях, а только когда они разделены упомянутыми символами.

Обычно для ввода данных применяется метод `System.out`, для которого автоматически устанавливается связь с экраном посредством оператора `in`, который, в свою очередь, связан с клавиатурой. Ниже приведен стандартный пример использования операций чтения и вывода.

```
System.out.print("Enter your age: ");
int age = in.readInt();
```

Два метода `format` предоставляют средства для управления выводом числовых данных (в процессе вывода числа преобразуются в строки). Эти методы не связаны с потоком, поэтому они могут быть использованы для работы с данными, которые должны быть записаны в файл. Что же означает тот факт, что они являются методами класса? Мы обсуждали этот вопрос в разделе 3.1, когда рассматривали класс `Math`. Чтобы восстановить в памяти эти сведения, сравните строки:

```
double y = Math.sin(x);
mugs.write();
```

метод `sin` является методом класса, а `write` — это обыкновенный метод объекта (или его экземпляра), который вызывается через этот объект. Методы класса существуют только в классе, ссылаться на них необходимо через идентификатор класса. Таким образом, вызов методов `format` осуществляется посредством имени класса `Stream`.

Вместе с форматами параметр `align` определяет минимальное число символов, которые необходимы для вывода числа. Так, если взять число 123 и задать для параметра `align` значение 6, при выводе мы получим три пробела перед числом 123. Благодаря этому числа можно аккуратно выровнять в колонке. Методы `format` обладают следующим важным свойством: если число не помещается в отведенный для него промежуток, данный промежуток расширяется справа, а цифры перед десятичной точкой в любом случае выводятся полностью. Для вещественных чисел

параметр `frac` определяет такой формат: дробная часть всегда содержит фиксированное количество цифр, а любые последующие цифры отсекаются. Например, выражение

```
System.out.println(Stream.format(x, 10, 4));
```

для различных значений x будут возвращены такие результаты:

```
-1234.5678        -1234.5678
1234.56789         1234.5679
45.67                45.6700
4                     4.0000
4.56789               4.5679
0                     0.0000
123456789     123456789.0000
777777.88888   777777.8889
```

Наконец, существуют методы, предназначенные для открытия и закрытия файлов, а также дублирования для них свойств методов `print` и `println` (подробный материал по данной тематике изложен ниже).

Класс `Stream` входит в пакет `javagently`. Вскоре вы узнаете, как компилировать, хранить и обеспечивать доступ к этому пакету. Полное описание класса `Stream` содержится в главе 7. Следующий пример демонстрирует, какие преимущества можно получить при его использовании.

Константы

Внимательный читатель заметит, что в классе `Stream` определены две константы, которые пишутся с большой буквы. В Java мы можем объявлять переменные, у которых начальные значения являются также конечными (т.е. не изменяются). Константа объявляется следующим образом:

Объявление константы
static final *type CONSTANTID = value;*
Объявляется константа `CONSTANTID`. Ей присваивается определенное значение, которое остается неизменным.

В Java имена констант принято писать с прописной буквы. Еще одна константа, с которой мы познакомились, — это `Math.PI`. Мы могли бы установить такие константы:

```
static final double KMTOMILES = 1.6;
static final int MILLENNIUM = 2000;
```

Пример 4.3. Форматирование большой таблицы температур

Чтобы продемонстрировать применение класса `Stream` для форматирования вывода, попробуем усовершенствовать программу, выводящую таблицу преобразования температур (пример 3.6). Для разделения столбцов мы использовали символ табуляции, однако это не дает нужного эффекта, когда числа имеют разное количеств знаков.

Таблица приобретет надлежащий вид при таком изменении операторов `print` в методе `outaLine`:

```
System.out.print(Stream.format(c,6));
System.out.print(Stream.format((int) fahrenheit(c),5));
```

При выводе значений температуры в градусах Фаренгейта мы столкнемся с небольшой трудностью. Метод, который мы определили, использует функцию `Math.round`, которая возвращает результат типа `long`, поэтому в первоначальной программе мы согласились с тем, что метод `fahrenheit` будет возвращать значение типа `long`. Однако метод `format` возвращает значение типа `int`. Поэтому нам придется преобразовать тип результата, прежде чем он будет принят.

```
Temperature Conversion Table
============================
```

C	F	C	F	C	F	C	F	C	F
0	32	1	34	2	36	3	37	4	39
5	41	6	43	7	45	8	46	9	48
10	50	11	52	12	54	13	55	14	57
15	59	16	61	17	63	18	64	19	66
20	68	21	70	22	72	23	73	24	75
25	77	26	79	27	81	28	82	29	84
30	86	31	88	32	90	33	91	34	93
35	95	36	97	37	99	38	100	39	102
40	104	41	106	42	108	43	109	44	111
45	113	46	115	47	117	48	118	49	120
50	122	51	124	52	126	53	127	54	129
55	131	56	133	57	135	58	136	59	138
60	140	61	142	62	144	63	145	64	147
65	149	66	151	67	153	68	154	69	156
70	158	71	160	72	162	73	163	74	165
75	167	76	169	77	171	78	172	79	174
80	176	81	178	82	180	83	181	84	183
85	185	86	187	87	189	88	190	89	192
90	194	91	196	92	198	93	199	94	201
95	203	96	205	97	207	98	208	99	210

Пример 4.4. Еще один вариант программы приветствия

Класс `Stream` очень "снисходителен" по своей природе: он обнаруживает неправильные числа и позволяет повторить ввод. Кроме того, он позволяет вставлять (а впоследствии игнорирует) пустые строки и пробелы между введенными данными. Этим классом не допускается вывод нескольких строк подряд: если мы напишем версию программы приветствия с использованием класса `Stream`, на экране отобразится только одно из имен Пьера:

```
import java.io.*;
import javagently.*;

class Greetings2 {
Greetings2 ()throws IOException {
  Stream in = new Stream (System.in);

  System.out.print("Как Вас зовут?");
  String name = in.readString();
  System.out.println("Здравствуйте " + name);
}

public static void main (String [] args) throws IOException {
  new Greetings2 ();
}
}
```

```
Как Вас зовут? Pierre Marchand
Здравствуйте Pierre
```

Программа `Greetings2` воспринимает только одно слово — Pierre (а не Pierre Marchand). Класс `Stream` позволяет размещать несколько элементов в одной строке. Воспользуемся этим для создания третьей версии программы, которая способна включать в приветствие имя из двух слов:

```
import java.io.*;
import javagently.*;

class Greetings3 {

  Greetings3 () throws IOException {
    Stream in = new Stream (System.in);

    System.out.print("Как Вас зовут ?");
    String name = in.readString();
    String surname = in.readString();
    System.out.println("Здравствуйте " + name +" "+ surname);
  }

  public static void main (String [] args) throws IOException {
    new Greetings3 ();
  }
}
```

```
Как Вас зовут? Pierre Marchand
Здравствуйте Pierre Marchand
```

Класс `Stream` очень "настойчивый": если ему нужно получить имя из двух слов, он во что бы то ни стало добьется своего. Таким образом, второй вызов метода `readString` не произойдет до тех пор, пока не будет введено второе слово.

Последовательность ввода

Обязательной является проверка завершения ввода данных. Предлагаем вашему вниманию несколько способов управления процедурой ввода строк переменной длины:

1. Если заранее известно количество символов, подлежащих вводу, достаточно отслеживать информацию о том, сколько символов уже введено.
2. Используйте исключение end-of-file для обнаружения символа конца файла.
3. По окончании ввода информации можно ввести какую-либо специальную комбинацию чисел или символов, например число 9999.
4. С помощью пользовательского интерфейса сигнализируйте программе о завершении ввода данных нажатием кнопки или другим подобным способом.

4.3. Создание пакетов и доступ к ним

Чтобы получить доступ к классу `Display` или `Stream` пакета `javagently`, в программу необходимо добавить соответствующий оператор:

```
import javagently.*;
```

Пакет `javagently` содержит три класса:

```
Stream
Display
Graph
```

Как использовать класс `Graph`, поясняется в следующих главах книги. Если нам нужен только класс `Stream`, достаточно такого оператора:

```
import javagently.Stream;
```

Поскольку `javagently` является нашим собственным пакетом, он не распространяется в скомпилированном виде в составе наборов JDK и Java IDE, которые вы, возможно, используете. Поэтому мы должны создать его, прежде чем приступить к работе с описанными выше операторами и к запуску программ из раздела 4.2. Выполните следующие несложные операции:

- Перейдите на один уровень вверх и создайте директорию `javagently`.
- Загрузите в эту директорию файлы классов `Display` и `Stream` с Web-узла (вы можете также напечатать тексты программ, обратившись к разделам 7.3 (`Stream`) и 13.4 (`Display`)).
- Оставаясь в этой директории, откомпилируйте пакеты `Stream.java` и `Display.java` (а также `Graph.java`, если он вам тоже нужен).
- Теперь в директории `javagently` будут храниться соответствующие файлы классов, включая и те, которые относятся к вспомогательным классам для вышеупомянутых.
- Добавьте в `autoexec.bat`, в строку `classpath`, еще одну директорию, расположенную на один уровень выше, чем `javagently`. Например, если путь к директории `javagently` — `C:\mywork\ javagently\`, новую директорию следует указать как `C:\mywork\`, а не `C:\mywork\javagently\`. Кроме того, должен быть указан путь к классам папки `bin`.
- Введите команду

  ```
  import javagently.*;
  ```

  в начало программы.

На рис. 4.2 показано, что должно получиться в результате выполнения перечисленных выше действий.

Оператор, импортирующий пакет в Java-программу, начинает процесс поиска в соответствии с указанным путем. Таким образом, мы включаем директорию `C:\COS110\` в `classpath`, поскольку в ней находится пакет `javagently`. Все классы Java теперь могут использовать пакет `javagently`, импортировав его. Если вы еще этого не сделали, обратитесь к пакетам `Display` и `Graph`, расположенным в директории `javagently`, а также повторите действия 2, 3 и 4.

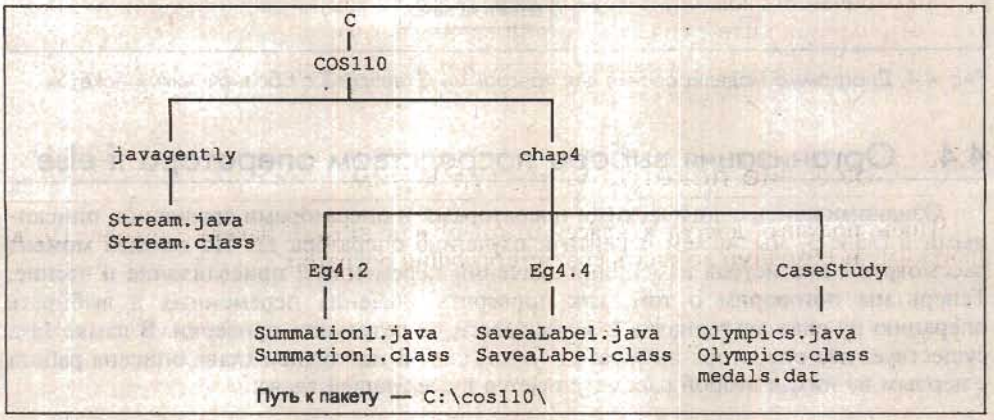


Рис. 4.2. Возможная система директорий для пакета javagently

Диаграммы моделирования для пакетов

Пакеты принято представлять на диаграммах моделирования специальными обозначениями (рис. 4.3). Для обозначения импортирования пакета применяется тот же символ, что и при инициализации, — стрелка. Однако путаница исключена, поскольку нельзя инициализировать пакет. Следовательно, программа `Greetings3` может быть представлена так, как показано на рис. 4.4. Трактовать данный рисунок можно следующим образом: объект `Greetings3` (созданный на основе класса `Greetings3`, который здесь не обозначен) импортирует два пакета, один из которых является пакетом Java, а другой — пакетом, который определен пользователем и содержит три класса.

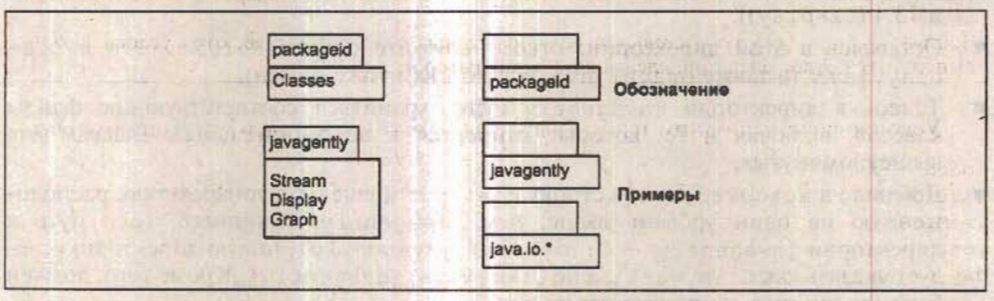


Рис. 4.3. Кассы различных пакетов

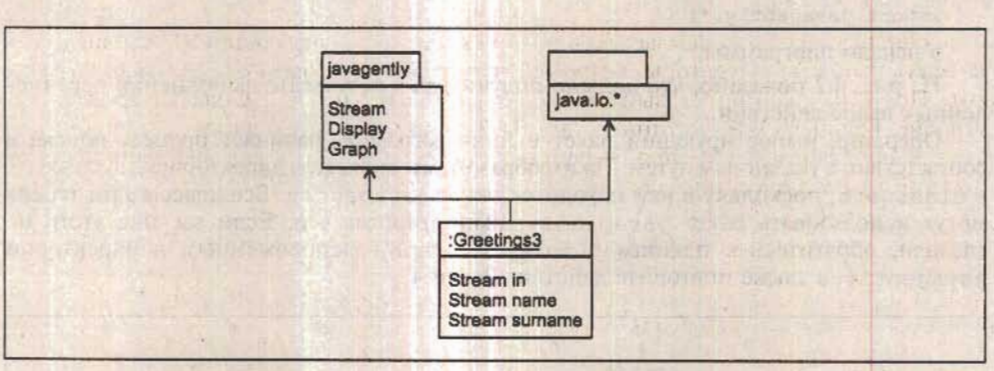


Рис. 4.4. Диаграмма моделирования для программы Greetings3 с обозначениями пакетов

4.4. Организация выбора посредством оператора if-else

Ознакомившись с логическими операторами и операторами сравнения, описанными в главе 3, мы можем перейти к изучению оператора `if`. На данный момент рассмотрены два метода изменения значений переменных: присваивание и чтение. Теперь мы поговорим о том, как проверять значения переменных и выбирать операцию из ряда альтернатив в зависимости от результата проверки. В языке Java существует два оператора выбора: `if` и `switch`. В настоящей главе описана работа с первым из них, а второй рассматривается в следующей главе.

Оператор if

Синтаксис оператора `if` имеет следующий вид:

Оператор `if`

```
if (condition) {
   statements1;
 }
 else {
   statements2;
 }
```

Проверяет условие *condition*. Если оно истинно, выполняются операторы *statements1*, иначе — *statements2*.
Часть `else` не является обязательной: если она не включена, то в случае, когда условие ложно, выполняться будет оператор, следующий за оператором `if`.

Условие является одним из булевых выражений (см. раздел 3.6). Например:

```
speed > speedlimit
(age >= 16) & (age < 75)
isaMinor
year == 1066
day != 29
initial != 'J'
time.lessThan(close)
answer.equals("Yes")
```

В операторе `if` мы ссылаемся на блок операторов, следующий за условием, как на **часть than**, а на операторы, которые следуют за оператором `else`, — как на **часть else**. Полный оператор `if` выполняется следующим образом. Сначала проверяется условие. Если результат истинный, выполняется часть `then`, а часть `else` пропускается. В противном случае пропускается часть `then`, а часть `else` выполняется. Вот простой пример оператора `if`:

```
if (number >= 0) {
  System.out.println("Positive")
}
else {
  System.out.println("Negative");
}
```

Элемент `else` выделен курсивом потому, что он не является обязательным и оператор `if` может быть использован в версии "if-then". Например:

```
if (day == 25) {
  System.out.println("Christmas, Hooray");
}
```

Изучение оператора `if` мы начнем с рассмотрения классического алгоритма поиска наибольшего числа.

Пример 4.5. Поиск наибольшего числа

Задача. Необходимо найти самое большое число в последовательности.

Решение. Программа последовательно просматривает числа, запоминая наибольший из просмотренных вариантов. При обнаружении числа, которое больше сохраненного

варианта, последний обновляется. Следует отметить, что отрицательные числа также подлежат обработке.

Алгоритм. Данный алгоритм представляет значительный интерес, поскольку он иллюстрирует метод индукции. Мы исходим из того, что самое большое число из *n* чисел уже найдено. Затем производится чтение числа *n+1*. Чтобы найти наибольшее из *n+1* чисел, надлежит сравнить новое число с наибольшим, найденным ранее, и, если оно превосходит его по значению, принять его за наибольшее. Этот процесс может повторяться столько раз, сколько потребуется.

С чего же начинается обработка? Прочитывается первое число, которое принимается за наибольшее. С этого момента и начинается выполнение программы.

Программа. Мы будем осуществлять ввод чисел с клавиатуры и использовать класс `Stream` для чтения данных.

```
import java.io.*;
import javagently.*;

class HighestValue {
  /* Программа поиска наибольшего значения, Дж.М.Бишоп, август, 1996 г.
   * ------------------------------------- Java 1.1, октябрь, 1997 г.
   *                                       Java 2 и новый класс Stream
   *                                       май, 2000 г.
   *
   * Находит самое большое значение в списке чисел.
   * Иллюстрирует применение оператора if.
   */

 HighestValue () throws IOException {

   Stream in = new Stream (System.in);

   System.out.println("***** Нахождение наибольшего числа *****");

   //Определяет, сколько значений будет введено.
   System.out.println ("Сколько чисел будет введено (1 или более) ?");
   int n = in.readInt();

   //Начинает обрабатывать последовательность.
   System.out.println("Введите эти числа");
   System.out.print("1>\t");
   int highest = in.readInt();

   //Читает и проверяет остальные числа.
   int number;
   for (int i = 2; i <= n; i++) {
     System.out.print(i+">\t");
     number = in.readInt();
     if (number > highest) {
       highest = number;
     }
    }

    System.out.println("Этого количества достаточно, спасибо.");
    System.out.println("Наибольшее число = " + highest);
  }

  public static void main(String[] args) throws IOException {
```

```
    new HighestValue ();
  }
}
```

Тестирование. Протестировать такой алгоритм можно несколькими способами. Например, введя самое большое значение сначала в начале, затем в конце, а в завершение — где-нибудь в середине. Еще один способ — ввод одинаковых чисел, за исключением одного. Проведите тестирование самостоятельно. Мы же покажем результат одного из запусков программы:

```
***** Нахождение наибольшего числа *****
Сколько чисел будет введено (1 или более) ?
5
Введите эти числа
1>      56
2>      -99
3>      23
4>      70
5>      -4
Этого количества достаточно, спасибо.
Наибольшее число = 70
```

Пример 4.6. Вычисление среднего арифметического

Задача. Членам научной экспедиции приходится в течение дня снимать различные показания. Они хотели бы вычислять средние значения на основе полученных данных. Любые показания, находящиеся вне заданного диапазона значений, должны быть отброшены.

Решение. Данная задача решается в два этапа:

- обработка введенных значений и вывод сообщения о результате;
- проверка принадлежности значения заданному диапазону и вычисление среднего значения.

Разработка. Часть программы, в которой используется оператор `if`, имеет следующий вид:

```
if (reading >= min & reading <= max) {
    total += reading;
    n++;
    d.print("K");
 }
 else {
   d.print("X");
 }
```

Другими словами, если операция ввода была проведена успешно и данные попадают в необходимый нам диапазон значений, мы вводим их в программу, увеличиваем значение `n` на единицу, а также выводим на печать символ `K`, который служит признаком того, что введенное значение нам подходит. В противном случае мы присваиваем показанию значение X и не включаем его в вычисления.

Также программа иллюстрирует, как указать программе, что введены все начальные данные, путем ввода строки в поле окна. Необходимый эффект может быть достигнут при вводе переменной `done` типа `boolean` в строку, которую мы читаем после каждого значения:

```
status = d.getString("Ввод закончен?");
done = status.equals("Да");
```

Значение переменной done проверяется в цикле for в самом его начале.

Программа. Полный текст программы имеет следующий вид:

```
import javagently.*;

class ReadingsAverages {
   /* Среднее значение показаний
    * -------------------------     версия с использованием класса Display
    *                               Дж.М. Бишоп, май, 2000 г.
    *
    * Вычисляет среднее арифметическое внутри заданного диапазона.
    * Иллюстрирует применение оператора if.
    */
Display d = new Display ("Вычисление среднего значения");

ReadingsAverages () {

  double total = 0;
  double reading;
  int min, max;
  int n;
  boolean done = false;
  String status;

  d.prompt("Минимальное",0);
  d.prompt("Максимальное",100);
  d.ready("После назначения границ диапазона \nнажмите кнопку \"Готово\"");
  min = d.getInt("Минимальное");
  max = d.getInt("Максимальное");
  d.println("Минимальное допустимое = "+min+"\nМаксимальное допустимое = "+max);
  d.prompt("Следующее значение",0);
  d.prompt("Ввод закончен?","Нет");
  d.println("Нажимайте кнопку \"Готово\" \n после ввода каждого значения");
  d.println("Введите \"Да\" после ввода всех чисел");
  for (n = 0; !done;) {
    d.ready(" ");
    reading = d.getDouble("Следующее значение");
    d.print(" " + reading);
    if (reading >= min & reading <= max) {
    total += reading;
    n++;
    d.print("К");
  }
  else {
    d.print("Х");
  }
  status = d.getString("Ввод закончен?");
  done = status.equals("Да");
  }
  d.println("\n Среднее значение для "+n+" чисел = "+
    Stream.format(total/n,8,3));
  }

  public static void main (String [] args) {
    new ReadingsAverages ();
  }
}
```

Диаграмма моделирования. Чтобы подчеркнуть тот факт, что доступ к классам `Stream` и `Display` осуществляется через объекты, мы включили в книгу диаграмму моделирования для этой программы (рис. 4.5). Эта диаграмма наглядно демонстрирует различия между классами и объектами. Класс `ReadingsAverages` имеет один статический метод — `main`. Данный метод, в свою очередь, создает объект класса, содержащий одну объектную переменную и конструктор. У конструктора есть несколько переменных, каждая из которых имеет простой тип или является строкой. Объектная переменная, `d`, создается как объект класса `Display` и включает все методы данного класса, как показано на рисунке.

На рис. 4.5 также представлен пакет в виде прямоугольника с верхним выступом. Классу `ReadingsAverages` передаются два пакета — `java.io` и `javagently`.

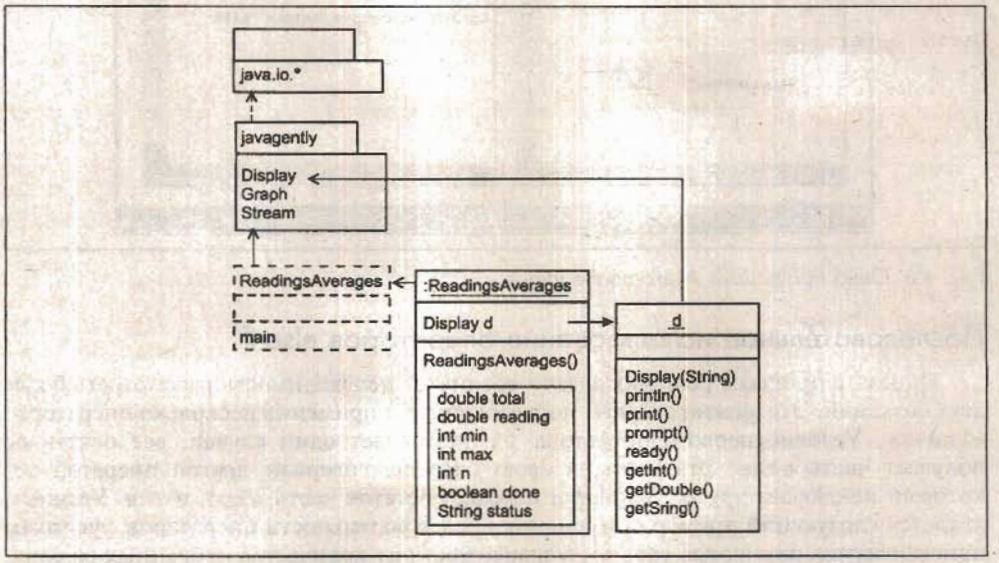


Рис. 4.5. Диаграмма моделирования для программы ReadingsAverages

Рассматриваемая диаграмма является полной. Хотя это и не упомянуто в программе, мы знаем, что пакет `javagently` использует пакет `java.io`, поэтому его мы тоже передаем. Класс `ReadingsAverages` импортирует пакет `javagently`. `ReadingsAverages` в качестве объекта, использующего идентификатор своего класса, инициализирует этот класс. Данный объект содержит одну переменную и один метод, переменные которого также изображены на диаграмме. Переменная `d` создается как объект класса `Display`, который, в свою очередь, определяет восемь содержащихся в нем методов. Эта взаимозависимость показана с помощью стрелки, которая направлена от пакета `javagently` к классу `Display`.

Тестирование. Стандартный набор выходных данных, полученных в результате выполнения программы, показан на рис. 4.6. Выбирая данные для тестирования, надлежит удостовериться, что в каждом случае будут задействованы все части оператора `if`. У нас только один такой оператор, поэтому нам необходима уверенность в том, что некоторые данные получат в результате метку K, а некоторые — X. Проверке необходимо также подвергнуть максимальное и минимальное значения.

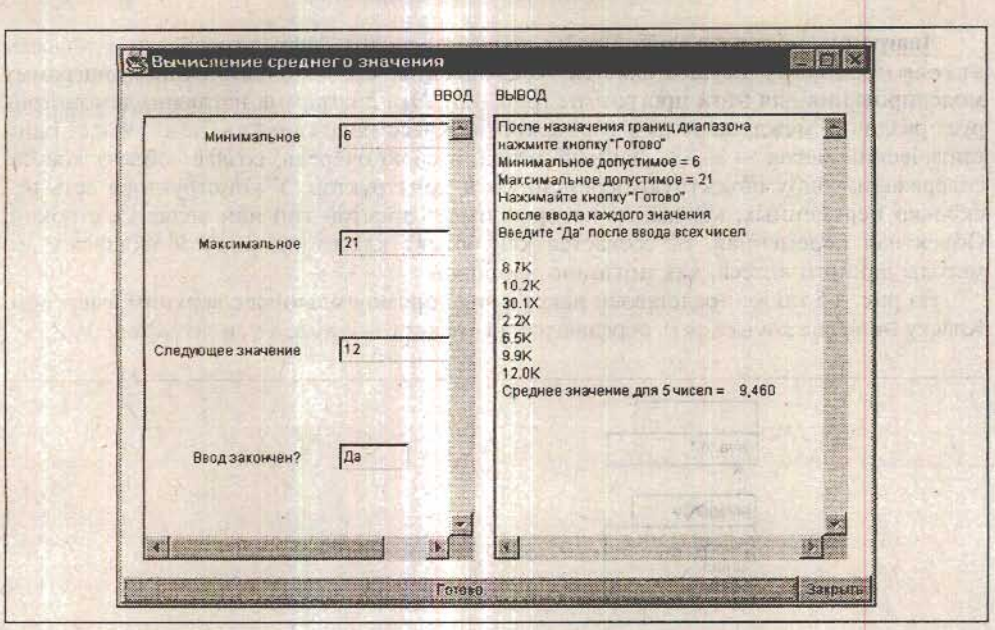


Рис. 4.6. Окно программы AveragingReadings

Последовательное использование операторов else-if

Иногда в процессе решения задачи возникает необходимость рассмотреть более двух ситуаций. Это можно сделать, последовательно применив несколько операторов `else-if`. Условие первого оператора `if` исключает один случай, все остальное получает часть `else`. Эта часть, в свою очередь, содержит другой оператор `if`, который исключает другое условие и передает остаток части `else`, и т. д. Удачным является следующий пример применения последовательности операторов: ученикам присваиваются различные уровни, в зависимости от количества набранных баллов:

```
int marks;
char grade;

if       (marks >= 80) grade="A";
else if  (marks >= 70) grade="B";
else if  (marks >= 60) grade="C";
else if  (marks >= 50) grade="D";
else                   grade="E";
```

Заслуживают внимания следующие моменты, касающиеся оператора `if`:

- Условия надлежит располагать в строгом порядке так, чтобы каждое из них исключало определенное количество баллов. Поэтому строка, которая присваивает класс D (количество баллов больше 50), будет достигнута только в том случае, если установлено, что это количество меньше 60.
- Последний класс — E — используется для всех остальных значений и не требует проверки условия.
- Большое значение имеет формат записи последовательных операторов `if`. Система условий должна быть ярко выражена.

- Оператор пишется в той же строке, что и условие, для повышения удобочитаемости.

В первую очередь следует проверять наиболее часто употребляемые условия. Если существует вероятность того, что кто-то не сдаст экзамен, целесообразно расставить условия в следующем порядке:

```
int marks;
char grade;

if       (marks < 50) grade="E";
else if  (marks < 60) grade="D";
else if  (marks < 70) grade="C";
else if  (marks < 80) grade="B";
else                  grade="A";
```

Рассмотрим еще один пример. Предположим, в группе насчитывается 300 студентов и нам необходимо распределить их по трем аудиториям. Мы решили произвести разбиение на группы по фамилиям, представленным в алфавитном порядке. В списке учащихся фамилии Джонсон и Паулус возглавляют вторую и третью сотни. Мы можем воспользоваться методом, который возвращает название аудитории, где должен находиться студент, следующим образом:

```
String room (String name) {
  if (name.compareTo("Jonson") < 0) {
    return "OPV 2-23";
  }
  else if (name.compareTo("Paulus") < 0) {
    return "GW4-10";
  }
else {

   return "EBE 4-21";

  }

}
```

Это метод будет вызываться следующим образом:

```
System.out.println("Your lecture room is " +room(you));
```

В качестве упражнения модифицируйте рассмотренный метод так, чтобы он мог подсчитывать количество имен для данной аудитории.

4.5. Ввод данных в файл и вывод данных из файла

Хотя при использовании в программе процедуры ввода данных с клавиатуры упрощается работа с ней, это не является лучшим решением, когда требуется ввести значительный объем информации. Во многих случаях удобнее сохранить данные в файле и считывать их оттуда при необходимости. Класс `Stream` может быть использован как для ввода данных в файл, так и для считывания их из него. Осуществить чтение из файла `mydata` позволяют следующие операторы:

```
Stream in = new Stream ("mydata", Stream.READ);

double x = in.readDouble();
```

В программе мы можем иметь несколько потоков указанных видов, используемых для одновременного ввода данных.

Вывод данных в файл

Часто важно иметь возможность сохранять выходные данные в файле. И не только потому, что таким образом легко хранить результаты работы и распечатывать их, но и потому, что очень часто программы формируют слишком много данных, которые трудно анализировать на экране. К тому же мы можем получить данные, которые не предназначены для просмотра, а являются входными для другой программы. Снова воспользовавшись классом `Stream`, мы получим:

```
Stream fout = new Stream ("results.out", Stream.WRITE);
```

Чтобы записать числа в файл, не придерживаясь строго какого-либо формата, следует применить метод `println`. Если строка содержит несколько чисел, нам придется самостоятельно разделить данные пробелами, иначе их нельзя будет ввести снова. Например, если `x` равен 4,5, а `y` — 0,001, оператор

```
fout.print((x) + (y));
```

создаст в файле следующую запись:

```
4.50.001
```

Поэтому при работе с файлами рекомендуется осуществлять посредством класса `Stream` форматированный вывод данных, как в следующем случае:

```
fout.print(Stream.format(x,10,4)+ Stream.format(y,10,4));
```

Еще один важный момент, касающийся вывода данных в файл, состоит в том, что они должны быть закрыты перед тем, как программа завершит свою работу. В противном случае все записи будут потеряны. Метод класса `Stream` имеет следующий вид:

```
fout.close();
```

Пример 4.7. Нумерация деталей

Задача. Техническая лаборатория Саванны составила перечень из 200 деталей, которые используются в ее изделиях, и сохранила его в файле. Техники хотели бы присвоить каждой детали серийный номер, начинающийся с кода лаборатории — 5000.

Решение. Рассматривайте детали как строки текста и копируйте их в поток выводимых данных, добавляя серийные номера. Гибкость программы повысится, если начальное значение можно будет вводить с клавиатуры в ответ на соответствующий запрос.

Пример. Предположим, в нашем распоряжении имеется файл `parts.dot`, содержащий следующую информацию:

```
Widgets, curly
Widgets, plain
Gadgets, purple
Thingies, bolted
Ghizmos, striped
```

а необходимая таблица с номерами будет записана в файл parts.out таким образом:

```
5000 Widgets, curly
5001 Widgets, plain
5002 Gadgets, purple
5003 Thingies, bolted
5004 Ghizmos, striped
```

Программа. Программа использует четыре потока — fin, fout, in и out. Только один из них, out, заранее определен в языке Java, поэтому все остальные объявлены в программе — каждый своим способом. Поток System.in может быть открыт как объект класса Stream, поскольку мы хотим читать из него числа.

```
import java.io.*;
import javagently.*;

class PartCodes {

  /* Программа для нумерации деталей. Дж.М.Бишоп, 1990 г.
   * ------------------------------  Java 1.1 - июль, 1999 г.
   *                                 обновление - май, 2000 г.
   *                                 для класса Stream
   *
   *Присваивает номера деталям, считывая информацию о них из файла.
   *Иллюстрирует ввод в файл и вывод из него данных.
   */

public static void main(String args []) throws IOException {
  new PartCodes ();
}

PartCodes () throws IOException {

  Stream in  =  new Stream (System.in);
  Stream fin =  new Stream ("parts.dat",Stream.READ);
  Stream fout = new Stream ("parts.out",Stream.WRITE);

  int number = 0;
  String part;
  System.out.println("****** Детали из файла parts.dat ******");
  System.out.print("С какого числа начать нумерацию деталей? ");
  number = in.readInt();
  System.out.print("Вся информация записывается в файл parts.out");
  for (int i = number; i < number+5; i++) {
    part = fin.readLine();
    fout.println(i+" "+part);
    System.out.print('.');
  }
  System.out.println();
  fout.close();
  }
}
```

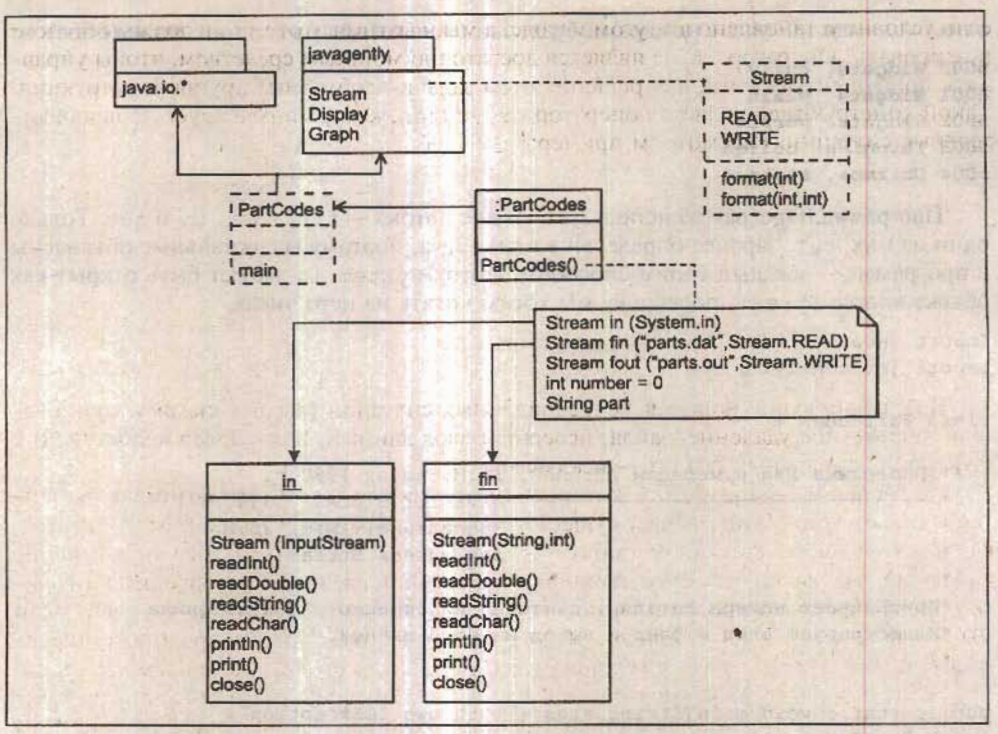


Рис. 4.7. Диаграмма моделирования для программы PartCodes

Тестирование. После запуска программы мы увидим на экране следующее:

```
****** Детали из файла parts.dat ******
С какого числа начать нумерацию деталей?
Вся информация записывается в файл parts.out.....
```

Чтобы показать, что происходит при чтении файла, мы использовали прием, позволяющий сэкономить время. Каждый раз, когда запускается цикл, мы печатаем на экране точку, означающую, что прочитана следующая строка. Если файл имеет большой объем, это позволяет пользователю увидеть, что программа все еще выполняется.

Диаграмма моделирования. Интересная особенность диаграммы, изображенной на рис. 4.7, заключается в том, что на ней четко показана разница между методами, относящимися к объекту `Stream`, и теми, которые принадлежат к классу. Потоки `in` и `fin` являются объектами класса `Stream`, а `fout` еще не инициализирован (для сохранения места в памяти). Конструктор потока `fin` будет копировать свои параметры в переменные объекта `fin`. Более детально переменные данного типа рассматриваются в главе 8.

4.6. Обработка исключений

Оператор `if` имеет средства контроля за состоянием текущего метода. Мы можем проверить значения данных, к которым имеем доступ, и отреагировать соответствующим образом в зависимости от полученных результатов. Но как следует поступить,

если условие установлено в другом методе, а мы, вызывая этот метод, должны на него реагировать? Оператор `if` не является достаточно мощным средством, чтобы управлять такими ситуациями, для решения этой задачи необходима другая конструкция. Такой конструкцией является оператор `try-catch`, который реагирует на исключительные ситуации. Рассмотрим пример:

```
try {
  for (;;)
    number = Stream.readInt(fin);
    total+=number;
}
catch (EOFException e) {
  System.out.println("All the data read");
}
```

На случай возникновения непредвиденных ситуаций (разрыв связи у пользователя, ошибочное удаление файла, неверный ввод данный, host-сервер недоступен) у Java есть специальный объект — **исключение**.

Исключение — это объект, который сигнализирует о том, что возникла нестандартная ситуация. Хотя в Java существует много заранее определенных объектов, связанных с исключениями, мы можем создать и свой собственный. Если исключение возникает вне нашей непосредственной программной среды, нас необходимо информировать о том, что произошло, после чего мы сможем управлять ситуацией. Если мы никак не прореагируем, работа выполняемого метода будет завершена, а информация об исключительной ситуации будет передана методу, вызвавшему текущий метод. Этот процесс может повторяться до тех пор, пока необработанное исключение не будет передано виртуальной машине Java, которая, в свою очередь, завершит работу программы.

Рис. 4.8 иллюстрирует процесс, когда один метод, у которого есть готовый обработчик, вызывает другой. В зависимости от того, как структурирована оставшаяся часть метода A, мы можем либо продолжить работу после обработки исключения, либо остановить программу.

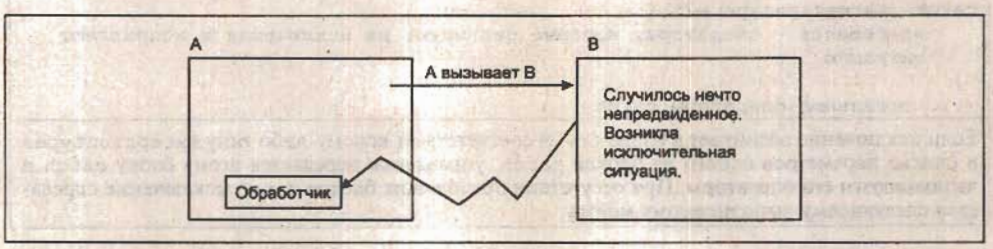


Рис. 4.8. Процесс обработки исключительной ситуации

Теперь давайте рассмотрим конкретный пример, связанный с вводом данных в программе. Утомительно вводить значения с клавиатуры, если их много, а если они сохранены в файле, то пользователь программы вряд ли захочет подсчитать заранее точное количество значений, которые необходимо ввести. Поэтому надлежит обеспечить возможность каким-либо способом указать, что поток данных заканчивается. Одним из способов решения данной задачи является использование исключений языка Java.

Параметр `e` имеет тип исключения. Для него определен метод `getMessage`, который может пригодиться, как вы увидите позже, для выявления более общего исключения, такого как `IOException`, и последующего вывода информации о нем.

При работе с клавиатурой пользователь, завершив ввод данных, нажимает специальную клавишу или комбинацию, чтобы сообщить системе об окончании данной процедуры. Это может быть клавиша [Esc], комбинация [Ctrl-D] или [Ctrl-Z]. Специальный символ, присутствующий в комбинации клавиш, фиксируется классом `Stream` и передается вызывающему методу путем генерирования исключения `EOFException`.

Обработка исключений

В исходных методах ввода данных языка Java каждый раз осуществляется проверка, исчерпан ли поток вводимых данных (достигнут ли конец файла либо указал ли пользователь, что ввод закончен стандартным путем с помощью клавиатуры). Если в результате проверки возвращается значение `true`, метод ввода данных вызывает исключение `EOFException`, после чего процедура ввода должна отреагировать на него соответствующим образом.

Определение исключительных ситуаций

Чтобы программа отреагировала на заранее определенную исключительную ситуацию, такую как `EOFException`, мы должны определить две части оператора — `try` и `catch`:

- `try` — указать блок операторов, в котором вызов любого метода или выполнение другой операции может привести к возникновению исключительной ситуации (данный блок должен предваряться ключевым словом `try`).
- `catch` — снабдить блок одним или несколькими обработчиками и поставить перед ними ключевое слово `catch`.

Обработчик сам по себе является блоком, который содержит операторы, связанные с сообщением об исключительных ситуациях или ошибках. Данные операторы применяются по следующей схеме.

Оператор try-catch

```
try {
    statements - операторы, при выполнении которых может возникнуть
    исключение, в том числе вызовы метода, при которых данная
    ситуация также может возникнуть
}
catch (Exceptiontype e1) {
    statements - операторы, которые реагируют на исключения и исправляют
    ситуацию
 }
catch (Exceptiontype e2) {
    statements - операторы, которые реагируют на исключения и исправляют
    ситуацию
 }
 ... остальные операторы catch
```

Если исключение возникает в блоке `try` и соответствует какому-либо типу `Exceptiontypes` в списке параметров одного из блоков `catch`, управление передается этому блоку `catch` и выполняются его операторы. При отсутствии подходящих блоков `catch` исключение передается следующему выполняемому методу.

Таким образом, фрагмент программы, ответственный за чтение данных до конца файла, имеет следующий вид:

```
try {
   for ...
   number = Stream.readInt(fin);
   }
catch (EOFException e) {
  display.println("Все данные считаны")
}
```



```
   System.out.println("С какого числа начать нумерацию деталей?");
   number = in.readInt();
   System.out.print("Вся информация записывается в файл parts.out");

 try {
   for (int i = number; ; i++) {
     part = fin.readLine();
     fout.println(i+" "+part);
     System.out.print(".");
   }
 }
  catch (EOFException e) {
   System.out.println("\nВсе данные считаны");
```

Параметр e имеет тип исключения. Для него определен метод getMessage, который может пригодиться, как вы увидите позже, для выявления более общего исключения, такого как IOException, и последующего вывода информации о нем.

При работе с клавиатурой пользователь, завершив ввод данных, нажимает специальную клавишу или комбинацию, чтобы сообщить системе об окончании данной процедуры. Это может быть клавиша [Esc], комбинация [Ctrl-D] или [Ctrl-Z]. Специальный символ, присутствующий в комбинации клавиш, фиксируется классом Stream и передается вызывающему методу путем генерирования исключения EOFException.

Обработка исключений

Как вам уже известно, операции ввода-вывода могут привести к появлению исключительных ситуаций, которые должны быть обработаны, что оказывает влияние на соответствующие заголовки методов. Обработка исключений осуществляется следующим образом:

- **Возникновение исключения.** Если исключение не может быть обработано, оно передается вызывающему методу; если исключение перехватывается и частично обрабатывается, мы можем передать его вместе с оператором throw.
- **Объявление исключения.** В методе следует объявить исключения, которые могут быть обработаны вызывающим его методом.

Пример 4.8. Нумерация деталей

Давайте модифицируем программу PartCodes так, чтобы число изделий было неограниченным, а программа сама определяла, где конец файла. В следующем примере измененные строки программы подчеркнуты.

```
import java.io.*;
import javagently.*;

class PartCodes {

  /* Нумерация деталей. Дж.М. Бишоп, 1990 г.
   * -----------------  Java 1.1, июль 1999 г.
   *                    изменена в мае 2000 г.
   *
   *
   * Присваивает номера деталям, осуществляя чтение из файла.
   * Иллюстрирует обработку исключения EOF.
   */


PartCodes () throws IOException {

  Stream in  =  new Stream (System.in);
  Stream fin =  new Stream ("parts.dat",Stream.READ);
  Stream fout = new Stream ("parts.out", Stream.WRITE);

  int number = 0;
  String part;
  System.out.println("****** Детали из файла parts.dat ******");
```

```
  System.out.println("С какого числа начать нумерацию деталей?");
  number = in.readInt();
  System.out.print("Вся информация записывается в файл parts.out");

try {
  for (int i = number; ; i++) {
    part = fin.readLine();
    fout.println(i+" "+part);
    System.out.print(".");
  }
}
 catch (EOFException e) {
  System.out.println("\nВсе данные считаны");
}
  fout.close();
}

public static void main(String args []) throws IOException {
  new PartCodes ();
}
}
```

Другими словами, конец цикла уже не является условием оператора `for`, он определяется при чтении файла.

Вероятен вопрос относительно того, почему и метод `main` и конструктор должны содержать пункт `throws IOException`, когда мы перехватываем исключение. Причина заключается в том, что существует другое исключение — `FileNotFound`. Сейчас мы рассмотрим способ, с помощью которого данное исключение может быть обработано.

Восстановление работы программы при исключениях

В предыдущем примере показано, как следует обработать исключение для изящного окончания программы. В ряде случаев вместо завершения выполнения метода мы хотим вернуться в начало и повторить операцию, приведшую к возникновению исключения. Примером такой ситуации может служить открытие файла. Если имя файла введено неправильно, пользователю можно дать шанс ввести имя заново. Алгоритм, работающий описанным выше образом, показан в псевдокоде на рис. 4.9.



Рис. 4.9. Алгоритм безопасного открытия файла

Для программы нумерации деталей, точнее, для того ее фрагмента, который несет ответственность за открытие файла, реализовать указанный алгоритм можно следующим образом:

```
/* Объявляет поток с клавиатуры и выполняет инициализацию,
 * чтобы файловый поток тоже в него входил.
 * Java требует, чтобы объекты, используемые в дальнейшем,
 * обязательно имели значения.
 */

Stream fin = new Stream (System.in);

//Делает пять попыток, запрашивая корректное имя файла
for (int tries = 1; tries <= 5; tries++) {
  try {
    display.prompt("Имя файла");
    display.ready("Нажмите кнопку Готово");
    filename = display.getString("Имя Файла");
    fin = new Stream (filename.Stream.READ);
    display.println("Файл" + filename +" открыт успешно");
    break;
  }
  catch (FileNotFoundException e) {
    display.println("Файл "+filename + " не найден");
    if (tries<5) {
      display.println("Введите другое имя файла.");
    }
    else {
      display.println("Слишком много попыток: ");
      display.println("Поток по умолчанию соединен с клавиатурой");
    }
}
```

Эти операторы могли бы быть вызваны в начале конструктора. В следующем примере мы увидим, как создавать методы вне конструкторов и добавлять их к уже существующим классам и пакетам. А пока мы рассмотрим ряд связанных с ними моментов.

Существует правило, согласно которому на любое значение, которое задано вне блока `try`, следует ссылаться как на потенциально неустановленное, поскольку исключение не может быть обработано до того, как оно было определено. Компилятор проверяет, соблюдается ли это правило. Подразумевается, что переменным сначала должно быть присвоено значение по умолчанию. Если таковым является имя файла, это довольно просто осуществить. Но если это сам файл, то мы должны быть осторожны. Файлу нельзя присвоить любое имя, поскольку его может не существовать. Поэтому мы используем достоверный файловый объект — клавиатуру.

Заключая последовательность операторов `try` в цикл, мы ограничиваем количество обращений к имени файла. Если пользователь, выполнив пять попыток, так и не введет корректное имя файла, программа прекратит работу. В блоке `catch` мы определяем, является ли текущая попытка последней, и, если да, выводим соответствующее сообщение.

Если файл открыт успешно, программа выйдет из цикла до использования всех пяти попыток: для этой цели предназначен специальный оператор прерывания.

Пример 4.9. Надежное средство открытия файла

Поскольку у нас неоднократно будет возникать необходимость в открытии файла при создании различных программ, рассмотрим как можно создать стандартную утилиту открытия файла. При написании собственного пакета от нас потребуется выполнение уже описанных действий, но так, чтобы его класс и методы были общедоступными.

Разработка метода `open` требует некоторых размышлений. Мы могли бы использовать уже созданный конструктор для класса `Stream` с дополнительным параметром, определяющим количество попыток ввода. Проблема в случае выбора такого способа возникает при обращении к пакету `javagently`. Правильное решение — создание второго пакета, `myutilities`, который мы будем использовать для решения собственных задач. В данный пакет должны входить классы (не методы). Добавим в него класс `Filer`, содержащий метод `open`, объявляемый следующим образом:

```
public static Stream open (String filename) throws IOException (...)
```

Метод `open` имеет модификатор `public`, т. к. будет использоваться вне отдельной программы. Он имеет тип `static`, потому что объект класса `Filer` создаваться не будет. Параметром метода является предполагаемое имя файла, передаваемое программой: вне метода `open` мы должны установить связь с клавиатурой, чтобы при необходимости обеспечить ввод других имен файлов. Если после пяти попыток файл не будет открыт, мы можем обработать исключительную ситуацию.

Здесь представлен новый метод `open`, включенный в класс `Filer` и предназначенный для пакета `myutilities`.

```
package myutilities;

import java.io.*;
import javagently.*;

public class Filer {

   /* Класс Filer, Дж.М. Бишоп, июнь 2000 г.
    * -----------  Java 1.2
    *              основан на классе FileMan
    *              с июня 1997 г.
    *
    * Обеспечивает открытие файла при задании корректного
    * имени файла, дает пять попыток.
    * Иллюстрирует использование исключений для циклов try-for.
    */

public static Stream open (String filename) throws IOException {
   Stream in = new Stream (System.in);
     if (filename.equals(" ")) {
     System.out.print("Какой файл нужно открыть?");
     filename = in.readString();
   }

   for (int count = 0; count < 5; count++) {
     try {
       return new Stream(filename, Stream.READ);
     } catch (FileNotFoundException e) {
         System.out.println("Файл "+filename+" не существует.");
         if (count < 4) {
```

```
            System.out.println("Попытайтесь еще раз");
          }
          filename = in.readString();
        }
      }
      throw new FileNotFoundException ();
    }
}
```

Данная программа имеет несколько интересных особенностей, самой важной из которых является то, что в конце цикла, если не введено правильное имя файла, обрабатывается исключение `FileNotFoundException`. Идея заключается в том, что исключение должно быть найдено программой и обработано ею. Эта обработка заменяет вывод на экран сообщения "По умолчанию связь установлена с клавиатурой". Как может быть использован класс `Filer`, показано в примере 4.10.

Исключения, связанные с обработкой чисел

Другим важным исключением является `NumberFormatException`, которое возникает при неудачной попытке конвертировать строку в число. Например, целое число не может содержать десятичную точку, а в состав чисел не могут входить посторонние символы. Рассмотрим фрагмент примера 4.6, связанный с вводом. Нам следует ввести два целых числа и одно число типа `double`. Если пользователь вводит вещественное число в том случае, когда следует ввести целое, соответствующий метод `get` не сможет быть выполнен. Защита данной части программы может быть обеспечена следующим образом:

```
for (; ;) {
  try {
    display.ready("Press ready when min and max adjusted");
    min = d.getInt("Minimum");
    max = d.getMax("Maximum");
    // ввод успешен, поэтому
    break; // следует выйти из цикла
 }
 catch (NumberFormatException e) {
   display.println("Error in input: "+e.getMessage());
   display.("Minimum and maximum should be integers");
 }
 // попробуйте еще раз
}
```

В данном случае мы заставили пользователя ввести правильные данные, используя бесконечный цикл.

Другим исключением, связанным с числами, является `ArithmeticException`, которое возникает, когда выполняется некорректное вычисление (например, деление на ноль).

Пример 4.10. Олимпийские медали

Задача. Количество золотых, серебряных и бронзовых медалей, завоеванных спортсменами каждой страны на Олимпийских играх, было сохранено в файле. Необходимо вывести эти результаты, а также общее количество медалей для каждой страны.

Решение. Начнем с создания цикла, очень похожего на тот, который мы использовали в примере 4.8. Данные хранятся в файле, поэтому нет необходимости заниматься вводом информации. Информация в файле упорядочена следующим образом: название страны, количество золотых, серебряных и бронзовых медалей. Название страны представляет собой строку, а три остальных значения являются целыми числами. Например:

```
Australia     9     9    22
China        16    22    12
Cuba          9     8     8
France       15     7    15
```

Часть программы, которая решает поставленную нами задачу, является очень простой. Наибольший интерес для нас представляет тот фрагмент, который проверяет, является ли допустимым имя файла. Здесь мы должны использовать метод `Filer`.

Программа. Мы сначала приведем листинг программы, а затем опишем ее составные части.

```
import java.io.*;
import javagently.*;
import myutilities.*;

class Olympics {
   /* Программа отображения сведений об олимпийских медалях.
    * ------------------------------------------------------
    *                                   Дж.М. Бишоп,
    *                                   декабрь 1996 г.
    *                                   Java 1.1 октябрь 1997 г.
    *                                   Модифицирована для классов
    *                                   Stream и Filer в мае 2000 г.
    *
    * Считывает и суммирует количество медалей, завоеванных странами
    * на Олимпийских Играх.
    * Использует класс Filer для открытия файла.
    * Иллюстрирует концовку с исключением (последовательность try-for).
    */

Olympics () throws IOException {
   System.out.println("**** Олимпийские медали ****");
   System.out.println();

   Stream in = new Stream(System.in);
   //По умолчанию задается ввод данных с клавиатуры.
   Stream fin = new Stream(System.in);

   try {
     System.out.print("Из какого файла считывать статистику по медалям?");
     String filename = in.readString();
     fin = Filer.open(filename);
   }
   catch (FileNotFoundException e) {
     System.out.println("У Вас есть пять попыток");
     System.out.println("По умолчанию поток соединен с клавиатурой.");
   }
   String country;
   int gold, silver, bronze, total, all = 0;
   System.out.println("\nСтрана\t\tЗолото\tСеребро\tБронза\tВсего");
```

```
  try {
    for (;;) {
        country = fin.readString();
        gold = fin.readInt();
        silver = fin.readInt();
        bronze = fin.readInt();
        total = gold + silver + bronze;
        System.out.print(country);
        if (country.length() < 8) {
          System.out.print("\t");
        }
       System.out.println("\t"+gold+"\t"+silver+"\t"+bronze+"\t"+total);
    all += total;
    }
   } catch (EOFException e) {
  System.out.println("Всего было разыграно " +all+" медалей.");
    }
 }

 public static void main (String[] args) throws IOException {
  new Olympics ();
 }
}
```

Алгоритм. Первая последовательность `try-catch` не содержит цикл, однако один из них существует в классе `Filer`, который ее вызывает. В данном классе последовательность `try` включена в цикл `for`, чтобы обеспечить возможность осуществить пять попыток при открытии файла. Если классу `Filer` не удалось открыть файл, выполняется последовательность `catch` из класса `Olympics` и выводится сообщение. Поток `fin` задан для пакета объекта, благодаря чему мы можем при необходимости осуществить ввод данных с клавиатуры. Ввод данных и вычисления производятся вне цикла `try-for`, так что, как только будет зафиксирован конец данных, цикл завершится и будет выведено результирующее сообщение.

Тестирование. Результирующими данными программы являются следующие:

```
**** Олимпийские медали ****

Из какого файла считывать статистику по медалям?chc.txt
Файл chc.txt не существует.
Попытайтесь еще раз
medals.dat

Страна          Золото  Серебро Бронза  Всего
Австралия       9       9       22      40
Китай           16      22      12      50
Куба            9       8       8       25
Франция         15      7       15      37
Россия          14      5       20      39
Испания         4       5       12      21
Нигерия         3       7       1       11
Германия        18      22      14      54
США             23      19      24      66
Япония          10      2       9       21
Польша          4       6       8       18
Венгрия         2       6       8       16
Болгария        1       1       0       2
Всего было разыграно 400 медалей.
```

Резюме

Значение переменной может быть установлено двумя способами: путем присваивания или путем ввода. Для ввода в языке Java необходимо сначала установить связь с потоком. После этого можно читать строки данных. Благодаря классу `Stream` становится возможным чтение чисел и символов. Допускается объявление файлов как потоков ввода и вывода. Программы могут получать данные и передавать их, используя сразу несколько потоков.

Существует также два способа изменения хода программы (что зависит от условий в ней). Оператор `if` подходит для проверки локального состояния внутри метода. Он может включать только часть `if`, а может быть организован как последовательность операторов `else-if`. Если условие встречается в другом методе, возникает исключительная ситуация, которая может быть выявлена и обработана. Существуют стандартные способы продолжения работы программы при возникновении таких ситуаций и повторения необходимых операций.

Классы-утилиты могут быть помещены в пакеты, многократное применение которых обеспечивает оператор `import`. Пакет `javagently` — это особый пакет, который создан специально для данной книги. Пакет `myutilities` мы разработали для размещения собственных классов.

В данной главе рассматривалось употребление следующих ключевых слов:

```
catch          package
else           throws
if             try
import         public
```

Описаны следующие инструменты из библиотеки языка Java:

```
java.io          System.in
BufferedReader   readLine
```

и наших собственных библиотек

```
javagently       myutilities
Display          Filer
Stream
```

Дан обзор таких исключительных ситуаций:

```
IOException                 NumberFormatException
FileNotFoundException       ArithmeticException
EOFException
```

Следующий перечень включает темы, на которых в главе сделано особое ударение:

- объявление потока для ввода с клавиатуры;
- чтение строки;
- класс `Display`;
- класс `Stream`;
- объявление константы;
- оператор `if`;
- оператор `try-catch`.

Вопросы



4.1. Назовите оператор, обеспечивающий связь с клавиатурой:

а) `Stream in = System.in;`

б) `Stream in = new Stream(System.in);`

в) `Stream in = Stream (System.in);`

г) `Stream System.in = new Stream(System.in);`

4.2. У нас есть окно, в котором расположено поле ввода **File name** (имя файла). Почему метод `display.getString(filename)` выдает сообщение "ERROR: No such input label:"?

а) имя файла должно быть в кавычках

б) следует использовать метод `readString`, а не `getString`

в) сначала мы должны нажать кнопку **Готово**

г) методу `getString` требуется точное имя поля, т.е. "File name"

4.3. Для пользователей персонального компьютера: если класс `Stream` откомпилирован в директории C:\Book\JG\javagently, каким должен быть путь к классу?

а) Stream.class

б) C:\Book\JG\javagently\

в) C:\Book\JG\

г) C:\

4.4. Из вводимой последовательности чисел мы хотим записать только те, которые входят в диапазон 30—40. Как должна выглядеть конструкция с оператором `if`, предназначенная для управления этим процессом?

а) `if (number >=30 & <= 40)`

б) `if (number >=30 | number < 40)`

в) `if (30 <= number >= 40)`

г) `if (number >= 30 & number <= 40)`

4.5. Программа, использующая класс `Stream`, ожидает чтения из файла трех целых значений, которые затем будут присвоены переменным `a`, `b` и `c`. Что произойдет, если строка, соответствующая `b`, является ошибочной строкой, а присваиваемые значения таковы: `45`, `2r3`, `67` и `84`?

а) `a=45 b=67 c=84`

б) `a=45 b=23 c=67`

в) `a=45 b=undefined c=67`

г) `a=45 program crashes`

4.6. Программа, использующая класс `Display`, имеет три целочисленных поля со значениями `45`, `67` и `84`. Сначала нажимается кнопка **Готово**, а затем первое значение изменяется пользователем на `55`. После этого трижды вызывается метод `getInteger`. Каковы будут значения трех целых чисел?

а) `55 67 84`

б) `45 67 84`

в) `error 67 84`

г) `45 55 67`

4.7. Необходимо прочитать все 210 чисел из файла. Если мы организуем цикл `for` в методе `try` следующим образом:

```
try {
  for (int i = 0; i <=10; i++) { …
    int n = in.getInteger();
  }
 }
 catch (EOFException e) {
 }
```

сколько чисел будет прочитано?

а) 200

б) 210

в) 10

г) ни одного

4.8. Оператором, с помощью которого можно получить доступ к классам ввода-вывода языка Java, является:

а) `import java.io;`

б) `import java.io.*;`

в) `package java.io;`

г) `import javagently.*;`

4.9. Если `x` имеет значение `456.23458`, что даст в результате `display.println (Stream.format(x, 4, 2))`? (В ответах символ _ является символом пробела).

а) `_456.23`

б) `456.23`

в) `456.2`

г) `456.`

4.10. На рис. 4.7 класс `Stream` имеет два метода `format`. Они не показаны в объектах класса `Stream`, таких как `in` и `fin`, поскольку:

а) они статические

б) они имеют одно и то же имя

в) это экономит место на диаграмме

г) для них требуются другие классы в `javagently`

Задачи

Обратите внимание на то, что задачи 4.1—4.7 представляют собой расширенные версии задач, которые были рассмотрены ранее.

4.1. **Преобразования.** Модифицируйте программы из примеров 3.5 и 3.11 таким образом, чтобы считать данные, значения которых находятся в определенных для таблиц пределах.

4.2. **Усовершенствование класса Filer.** Расширьте класс `Filer`, добавив к нему другой метод, который также называется `open`, но имеет параметр, показывающий, сколько попыток можно сделать.

4.3. **Остановка песни.** Вернемся к задаче 3.9. Модифицируйте программу так, чтобы она приостанавливалась после каждого куплета песни и ждала ввода команды остановки или продолжения работы. Или, наоборот, сделайте так, чтобы она реагировала на комбинацию [Ctrl-Z] или [Ctrl-D], останавливаясь в конце следующего куплета.

4.4. **Расход топлива.** Модифицируйте программу 2.6 таким образом, чтобы она осуществляла чтение с дисплея всех значений, которые могут быть изменены.

4.5. **Снова средние значения.** Измените программу 2.5 так, чтобы она считывала отметки из файла и выводила на печать средний балл для каждого студента, а также вычисляла общий средний балл. Для определения конца файла используйте исключительную ситуацию.

4.6. **Победители марафона.** Пользуясь сведениями из задачи 3.2, обеспечьте ввод информации о бегуне в интерактивном режиме. Модифицируйте программу так, чтобы она могла определять победителя пробега (подсказка: используйте для решения данной задачи пример 4.5).

4.7. **Контроль веса.** Измените задачу 3.2 так, чтобы данные читались с дисплея, а не генерировались как случайные числа.

4.8. **Средний возраст.** В молодежный клуб ходят 24 ребенка, которые распределены по четырем звеньям. Возраст детей от 7 до 11 лет. Напишите программу, которая считывает возраст детей в каждом звене, определяет средний возраст для каждого звена группы из 24 детей, а также для группы в целом.

4.9. **Разбиение файлов**. Предположим, у нас есть файл с численными данными, часть которых — отрицательные, а часть — положительные. Нужно сначала создать два файла для раздельного хранения положительных и отрицательных чисел, а затем вывести оба файла, причем файл с положительными значениями должен быть выведен первым. Используйте подходящий генератор случайных чисел, чтобы создать файлы, общее количество чисел в которых должно быть равно 100.

4.10. **Длина дорожек компакт-диска.** Компакт-диск имеет 22 дорожки. Используя окно для определения времени звучания каждой из них в часах и минутах, вычислите общее время воспроизведения диска. Используйте поле со значением `1`, которое преобразуется в нуль, чтобы сигнализировать программе об окончании ввода данных.

4.11. **Длина дорожек компакт-диска и обработка исключительных ситуаций.** Модифицируйте задачу 4.10, используя обычные средства ввода-вывода языка Java (а также класс `Stream`). Чтобы сигнализировать программе о том, что уже введена информация о последней дорожке, используйте комбинации клавиш [Ctrl-D] и [Ctrl-Z], а также исключительные ситуации.

4.12. **Заказ комнаты.** Нам нужна система, которая подбирает наиболее подходящую комнату для размещения определенного количества людей.

- Создайте файл с номерами комнат и количеством людей, которое каждая из них может вместить.
- Выполнение программы должно начинаться с вывода на экран окна, где требуется указать количество людей и имя заказчика.
- Считывайте данные из файла, пока не найдете такую комнату, которая не более чем на 10% больше, чем требуется.
- Выведите отчет о заказе, если он успешно выполнен.

— Если программа не нашла комнату, отвечающую заданным критериям, она должна предложить любую свободную комнату, в которой может разместиться необходимое количество людей, даже если она слишком большая.

4.13. **Резервирование комнаты.** Мы модифицируем задачу 4.12 следующим образом:

— Сейчас мы можем оформить заказ, занося во второй файл данные с пометкой рядом с заказанной комнатой и именем заказчика. Этот файл затем станет файлом ввода, и данную комнату уже нельзя будет заказать.

— Программа может сообщать, что данная комната идеально подходит, но в данный момент заказана другим лицом.

4.14. **Эксперименты.** Сотрудники лаборатории снимают показания, а требуемый результат — это частное от деления первого показания на второе. Все показания хранятся в файле. Напишите программу, которая обрабатывает файл и отслеживает исключительные ситуации, возникающие при делении на ноль. В этом случае программа должна запросить замену обоих значений для получения точных данных и продолжить обработку результатов, пока не будет достигнут конец файла.

4.15. **Экзамены.** Экзаменационный билет содержит четыре вопроса в разделе A и четыре в разделе B. Ответ на каждый вопрос оценивается по 20-балльной шкале. Студенты должны ответить на пять вопросов, причем из них два должны быть из раздела A и два из раздела B. Если даны ответы на большее количество вопросов, первые засчитываются, а последующие отбрасываются. Количество баллов при отсутствии ответа на вопрос равно 0.

Напишите программу, которая для каждого студента считывает из файла восемь полученных отметок и определяет общее количество баллов. Вывод должен быть сделан в файл и соответствовать входным данным, показанным ниже. При нарушении правил должны быть выведены соответствующие сообщения.

Пример исходных данных и результат выглядит так:

```
Sample Input                          Sample Output
Section A       Section B             Result Comment
10  15  0   0   20  8   17  0         70
10  9   7   20  0   0   0   10        36     Too many from A.
5   6   10  0   19  5   3   14        45     More than 5. Too many from B.
```

4.16. **Показатели уровня осадков.** Показатели уровня осадков в мм можно получить для каждого дня за последние четыре недели. Мы хотим знать общий уровень осадков для каждой недели, самый влажный день и самую сухую неделю. Напишите программу, которая будет читать в несколько приемов 28 показателей из файла и выводить в другой файл три фрагмента требуемой информации вместе с входными данными. Ниже представлены возможные данные и полученные на их основе результаты:

```
Sample data                           Sample results
3  0  0  7  8  21 0                   39 mm
0  1  1  0  0  0  4                    6 mm
9  6  7  0  0  0  0                   22 mm
0  0  0  0  0  0  1                    1 mm
The wettest day was day 6.
The driest week was week 4.
```

4.17. **Счет партии в гольф.** Площадка для гольфа города Саванна имеет девять лунок. Предположим, чтобы загнать мяч в лунку, игроку требуется от одного до пяти

ударов. Количество очков, набранных игроком в гольф, равно количеству ударов, выполненных им у каждой лунки. В зависимости от результата предыдущей партии для получения фактического результата у игрока из счета вычитается определенное количество очков. Игрокам также интересно знать, кто из них набрал необходимую сумму очков. Победителем считается тот, кто набрал наименьшее количество очков. Если бы подсчет очков при игре в гольф был компьютеризирован, то пример входных и выходных данных имел бы следующий вид:

```
Player  Handicap  Shots per hole      Total  Result  Under par?
1       6         1 3 6 2 1 4 3 2 4   26     20      yes
2       3         2 2 2 2 4 4 4 2 2   24     21      yes
3       2         4 5 4 3 4 1 3 5 4   33     31      no
The winner is player 1 with a handicapped result of 20.
```

Напишите программу, которая

— считывает из файла информацию о количестве ударов, выполненных у каждой лунки для нескольких игроков, учитывая исключительную ситуацию, когда данные заканчиваются;

— вычисляет общий счет каждого, а также счет после вычитания очков и сравнивает полученное количество очков с номинальным;

— определяет победителя и его счет;

— выводит результаты вместе с входными данными.

4.18. **Счетчики на автостоянках.** Департамент транспорта города Саванна хочет принять решение относительно того, стоит ли начинать кампанию по борьбе с нелегальной парковкой. Определенное число инспекторов этого департамента были посланы в различные районы города, где ограничено время парковки. В каждом районе установлены свои ограничения. Каждый офицер Департамента транспорта должен следить за любыми десятью машинами в своем районе и записывать, в течение какого времени транспортное средство было припарковано в зоне с ограниченным временем парковки. Если время парковки более 50% машин будет превышать допустимую норму, Департамент транспорта примет решение развернуть массированную кампанию. Напишите программу, которая

— считывает из файла информацию о количестве районов, ограничении времени и общем времени парковки десяти транспортных средств в каждом районе;

— определяет количество машин в каждом районе, нарушивших ограничение;

— принимает решение, следует развернуть кампанию или нет;

— определяет район, в котором ситуация является наихудшей, и выводит его название на экран;

— выводит результаты и входные данные в другой файл.

Результирующие данные должны иметь следующий вид:

```
Number of zones: 3

Area  Limit  Parking times                Over limit
1     60     20 40 70 35 45 78  34 56 73 5    3
2     45     62 47 68 40 53 62 120  8 15 72   7
3     30     66 32 41 89  7 25  29 33 54 17   6
A campaign must be mounted.
Concentrate on area 2
```

ГЛАВА 5



Управление потоком

5.1. Требования к программе

По структуре хорошо спроектированной программы легко определить, каким образом связаны ее части и насколько надежна эта связь. Можно провести аналогию с мостом: каждая подпорка и каждая заклепка играет свою роль, но вместе они прекрасно выполняют общую функцию. Нередко мосты бывают очень красивыми. То же относится и к программам. Когда мы их создаем, нашей целью является разработать такую структуру, чтобы, во-первых, добиться необходимого результата, а во-вторых — сделать программу удобочитаемой, эффективной и приемлемой для многократного использования. Это и есть свойства хорошей программы. Именно о них пойдет речь в настоящей главе.

Корректность

Кажется очевидным, что завершенная программа должна быть правильной, однако немалая часть программного обеспечения в наши дни не лишена ошибок (именуемых на жаргоне "глюками"). В защиту разработчиков такого программного обеспечения можно сказать, что программирование является одним из самых сложных направлений в инженерии, а также одним из самых новых (более новой является, наверное, только генная инженерия). Лишь в середине 80-х годов стали появляться книги о разработке программного обеспечения, а международные структуры приняли стандарты относительно его надежности. Итак, как же обеспечить корректность программного обеспечения? Существует два золотых правила:

1. При написании программы следует придерживаться установленного порядка программирования.
2. Необходимо тщательно протестировать готовую программу.

Цель данной книги — выработать у читателей правильные навыки и побудить их использовать упомянутые правила.

Тестирование играет важную роль. Независимо от того, насколько тщательно оно выполнено, не всегда удается предусмотреть все ситуации, которые могут возникнуть во время работы программы. Ситуация усложняется, если пользователь может изменять ход процесса, вводя данные. В таких случаях нужно особенно тщательно подбирать данные для теста.

Удобочитаемость

Чтобы составить свое мнение о программе, мы должны иметь возможность прочитать ее и, возможно, даже обсудить с другими.

Удобочитаемость программы можно повысить за счет следующих приемов:

- использование выразительных идентификаторов (например, `timeInMins` вместо `t`);
- тщательный выбор формата программы (разбивка на фрагменты, размещение пустых строк);
- ввод комментариев, которые объясняют назначение последующих операторов;
- продуманный выбор типов и классов.

Мы можем продвинуться дальше и разработать стиль, который позволит осуществлять быстрый поиск нужных участков кода и эффективно общаться в среде программистов. Например, мы можем организовать наши методы так, чтобы объявления всегда предшествовали операторам. Кроме того, часто методы классов располагают в алфавитном порядке.

Многократное использование программ

Ошибки в программном обеспечении, а также неспособность фирм производить его вовремя существенно ухудшают ситуацию в индустрии программного обеспечения. Одним из путей ее улучшения является многократное использование программного обеспечения, которое было испытано и проверено ранее. Пакеты языка Java — классический пример такого обеспечения: вместо того чтобы каждый раз создавать класс для хранения и управления случайными числами, мы можем использовать готовые средства.

Когда же реализуется возможность многократного использования? При написании новой программы мы просматриваем те классы, которые уже имеются у нас в наличии и которые могут быть использованы или даже адаптированы. В то же время мы стараемся писать собственные классы так, чтобы они соответствовали всем требованиям. В языке Java имеются специальные конструкции классов, которые служат для обеспечения их многократного применения.

Эффективность

Плохо, когда программа работает медленно или имеет слишком большой объем. Существует множество нехитрых приемов, с помощью которых мы сможем программировать, экономно расходуя ресурсы без ущерба для качества. Вот несколько способов, позволяющих сделать программу более компактной:

- переменные объявляются по мере необходимости;
- методы используются максимально часто во избежание копирования повторяющихся операторов;
- простые вычисления производятся один раз, после чего сохраняется результат;
- для вычислений выбирается наиболее подходящий тип чисел, а не только `double`.

Разработка методов

Существует целый ряд способов, позволяющих группировать операторы таким образом, при котором программы становятся удобочитаемыми, гибкими и эффективными. Их обзор представлен в настоящей главе. Будут освещены также вопросы относительно того, когда имеет смысл сделать из группы операторов метод и как лучше группировать методы в классы. При формулировке ответа на первый вопрос надлежит учитывать следующие моменты:

1. Будут ли операторы использоваться в программе более одного раза?

2. Ссылаются ли операторы на данные, которые изменяются каждый раз при выполнении программы?
3. Используется ли для решения подобных подзадач группа операторов?
4. Не становится ли текущий метод слишком длинным (более 20 строк)?
5. Велика ли вероятность того, что данный метод будет применяться в другой программе?

Если утвердительный ответ дан на вопрос 1 или 2, желательно использовать методы (хотя это и не является обязательным требованием), иначе программа будет иметь очень плохой стиль. Если же "да" прозвучало в ответ на вопрос 3, 4 или 5, то методы следует использовать обязательно. Вернемся к примеру 3.4. Эта программа могла быть написана как сплошной набор операторов в конструкторе. При этом мы использовали бы не метод, создающий столбец гистограммы, а метод, рисующий ось координат (который содержит совершенно независимую функцию), что привело бы к значительному увеличению размера программы.

Другим местом, где необходимы методы, являются классы: они представляют собой средства, с помощью которых мы взаимодействуем с данными, хранящимися в объекте (примерами являются классы `Curio` и `Times`).

Программирование сверху вниз

Написание программы начинается с создания ее маленькой части, которая постепенно усложняется. Одним из лучших методов разработки программы является программирование сверху вниз (или нисходящее программирование), при котором задача делится на несколько подзадач, а решение каждой из них реализуется как вызов определенного метода. Впоследствии мы можем продолжить разработку каждого метода, возможно, даже разбивая его на подзадачи. Этот процесс продемонстрирован в примере 5.2.

Защита и доступность

Все объектно-ориентированные языки поддерживают средства, с помощью которых переменные и методы могут быть защищены от вторжения извне. Причины, по которым может осуществляться вторжение, будут исследованы далее, а здесь мы лишь отметим, что контроль над доступностью в языке Java обеспечивается посредством модификаторов, используемых для объявления метода или переменной, как показано ниже:

Контроль над доступом
private *declaration* **public** *declaration* *declaration*
Все элементы, объявленные как `private`, доступны только внутри объекта. Все элементы, объявленные как `public`, доступны для объектов любого другого класса, который импортирует класс, содержащий данный элемент. Если модификатор доступа не указан явно, элементы доступны для объектов любого другого класса того же пакета.

Таким образом, объявление `private` будет использоваться для переменных и методов, применяемых при выполнении процессов и вычислений внутри класса, которые не обязательно показывать пользователю. До сих пор наши классы не были

достаточно сложными, чтобы продемонстрировать данное требование. Однако уже в конце настоящей главы мы рассмотрим пример (программа для игры "Камень-ножницы-бумага"), в котором оно учтено.

Некоторые разработчики считают, что объявление вида доступа `private` или `public` всегда должно быть явным. Работая с Java Gently, мы избежали необходимости делать это, поскольку не требовалось использовать переменные и методы с видом доступа `private`, а применение модификатора `public` в максимальной степени соответствовало нашим требованиям. Если классы не входят в пакеты, как класс `Filer` (пример 4.9), то отсутствует необходимость расширять вид доступа к ним до `public`.

На диаграмме моделирования также можно указывать вид доступа по умолчанию. Чтобы подчеркнуть, какой вид доступа имеет переменная или метод — `private` или `public`, перед ними добавляется знак "+" или "–" (см. диаграмму для класса `Stream` в разделе 7.3).

Правила разработки класса

Методы являются средствами группировки операторов. На следующем уровне используются классы, которые позволяют группировать методы. Обычно в группу объединяются методы, которые служат общей цели. Например, класс `Math` обеспечивает выполнение тригонометрических методов, а класс `Stream` — методов ввода-вывода.

Круг выполняемых классом действий намного шире, чем объединение методов в группу. Мы можем включить в класс данные, хранящиеся в переменных. Разработка класса является краеугольным камнем объектно-ориентированного программирования. Далее мы рассмотрим основные принципы разработки программ, после чего проанализируем примеры, в каждом из которых эффективно используются классы и объекты.

1. **Согласованность.** Класс должен содержать набор операторов и данных, позволяющих решать строго определенный круг задач. Например, класс `Curio` определяет и обрабатывает данные о товарах, предлагаемых в лавке древностей, а класс `Times` обеспечивает выполнение операций со значениями часов и минут.
2. **Объединение однородных методов в класс.** Отдельное приложение может содержать несколько связанных классов, а не один. Например, класс `Curio` не предоставляет никаких операторов ввода-вывода, кроме `toString`. Мы можем включить в программу также класс, обеспечивающий форматирование выводимых данных надлежащим образом.
3. **Сокрытие информации.** Класс должен предоставлять пользователю только ту информацию, которая ему нужна, и не более. За счет этого можно предотвратить злоупотребление данными и обеспечить безопасность при работе с классом. От пользователя следует скрывать также методы, которые не имеют к нему отношения, а необходимы только классу. Чтобы скрыть переменные, их надлежит объявить как `private`. По умолчанию все данные, находящиеся в одной директории с классом, доступны для него. Более полно эта тема освещена в главе 9.
4. **Доступ к данным посредством методов.** Как следует из пункта 3, во всех стандартных классах языка Java соблюдается правило, согласно которому данные скрыты от пользователя и могут быть сделаны доступными с помощью точно определенных методов, которыми обычно являются `get` и `set`. Таким образом обеспечивается защита данных от нежелательного изменения. Например, класс `Date` (см. пример 1 главы 3) не позволяет выполнять операции со значениями времени, но предоставляет для этой цели методы `getTime` и `setTime`.

5. **Инициализация объекта.** При создании объектов целесообразно копировать в них начальные значения для дальнейшего применения вместо того, чтобы каждый раз присваивать значения заново. Например, при использовании класса `Curio` мы устанавливаем начальные значения для каждого объекта этого класса (а не изменяем их), поэтому они всегда доступны. Конечно, некоторые значения должны быть изменены в обязательном порядке (например, запас товаров магазина), однако основная цель — сделать вызов методов как можно более простым.

Остальные правила будут приводиться по мере необходимости.

5.2. Циклы while и do с выходом по условию

Наше знакомство с циклами состоялось во время разговора о том, как управлять заданиями, которые должны неоднократно выполняться в теле программы. Мы рассмотрели два вида циклов:

- циклы со счетчиком, организованные с использованием оператора `for`;
- неопределенные циклы, организованные с помощью оператора `for`, в которых для выхода используются прерывания и исключения.

Как объяснялось при обсуждении исключений, место, где исключение обрабатывается, находится на один уровень ниже того места, где оно зафиксировано. Условие достижения конца цикла является скрытым, если только оно не указано явно в имени исключения (например, `EOFException`).

Сейчас мы рассмотрим третью и самую важную группу циклов. Циклы эти основаны на условии, которое является видимым и фиксированным в месте, определенном самим циклом.

Синтаксис цикла с выходом по условию

Циклы с выходом по условию формируются с использованием операторов `while` и `do`. Оператор `while` в цикле используется следующим образом:

Оператор `while`

```
while (condition) {
    statements
}
```

Проверяется условие `condition`. Если оно истинно, выполняется оператор `statements`, после чего условие проверяется снова. Этот процесс повторяется до тех пор, пока условие не станет ложным. В этом случае управление передается оператору, следующему после цикла.

Вот простой пример цикла `while`:

```
int number = 0;
while (number <= 10) {
  System.out.println(number+" ");
  int number = (int) Math.random()*20;
}
```

Выполнение цикла обязательно начнется, поскольку переменной `number` присвоено нулевое значение, а выход из цикла осуществляется, когда это значение больше 10. Сначала выводится значение переменной, а затем случайным образом генерируется новое число в пределах от 0 до 20. Условие, при котором происходит остановка цикла, выполняется тогда, когда значение выходит за верхний предел

диапазона, т.е. становится больше 10. Таким образом, цикл продолжается, пока переменная `number <= 10`.

Следовательно, операторы на какой-то стадии должны изменять условие, иначе цикл будет выполняться бесконечно. Важно перед входом в цикл `while` инициализировать соответствующие переменные, чтобы условие могло быть корректно проверено.

Циклы `while` и `for` могут содержать оператор прерывания `break`, который служит для выхода из цикла независимо от условия. Так, в приведенном выше коде мы можем прервать цикл после вывода значения `number`, если оно равно 5. Для этого надлежит модифицировать код следующим образом:

```
int number = 0;
while (number <= 10) {
  System.out.println(number+" ");
  if (number == 5) break;
  int number = (int) Math.random()*20;
}
```

Это нельзя приравнивать к добавлению еще одного условия, как в следующем случае:

```
int number = 0;
while (number <= 10 && number != 5) {
  System.out.println(number+" ");
  int number = (int) Math.random()*20;
}
```

Здесь появление значения 5 будет зафиксировано прежде, чем произойдет его вывод.

Пример 5.1. Наибольший общий делитель

Задача. Необходимо найти наибольший общий делитель двух чисел.

Решение. Одно из возможных решений — поиск всех делителей для каждого числа и сравнение обоих списков с целью нахождения наибольшего из значений. Следует отметить, что данный способ является наиболее быстрым.

Алгоритм. Предположим, у нас есть числа a и b, причем a больше b, а их наибольший общий делитель (НОД) равен f. Таким образом, $a - b$ и b также будут иметь НОД f. Мы воспользуемся этим фактом, постепенно заменяя большее из чисел их разностью, пока не получим два одинаковых числа. Этим числом и будет НОД, даже если он равен 1.

Примеры. Следующие примеры иллюстрируют этот алгоритм в действии.

a	*b*	*|a - b|*
65	**39**	**26**
26	**39**	**13**
26	**13**	**13**
13	**13**	**НОД**

a	*b*	*|a - b|*
99	**66**	**33**
33	**66**	**33**
33	**33**	**НОД**

Программа. Программа содержит два цикла `while`. Так как мы собираемся запускать ее несколько раз, а также использовать класс `Display`, мы организуем

внешний цикл `while(true)`, который представляет собой более элегантную конструкцию, чем цикл `for(; ;)`, использовавшийся нами ранее. Программа останавливается при нажатии кнопки **Закрыть** на экране.

Второй цикл `while` является вложенным. Он выполняет две операции: выводит текущие значения и изменяет впоследствии *a* или *b*. Условие легко задать с помощью выражения `(a!=b)`, где *a* не равно *b*.

```
import javagently.*;

class HCFRepeat {

   /* Программа вычисления НОД. Дж.М. Бишоп, август 1996 г.
    * ======================== Версия с классом Display, июль 1999 г.
    *                          обновление - май 2000 г.
    * Вычисляет наибольший общий делитель двух целых чисел.
    * Демонстрирует работу цикла while.
    */

HCFRepeat () {
  Display display = new Display("Наибольший общий делитель");
  display.println("*****Наибольший общий делитель*****");
  display.prompt("Число a", 567);
  display.prompt("Число b", 123);
  int a, b;

  while (true) {
    display.ready("После ввода чисел \nнажмите кнопку \"Готово\"");
    a = display.getInt("Число a");
    b = display.getInt("Число b");
    display.println("a\tb");
    while (a != b) {
      display.println(Stream.format(a,6)+"\t"+Stream.format(b,6));
      if (a > b) {
       a -=b;
      } else {
        b -=a;
      }
  }
  display.println("Наибольший общий делитель: " + a);
  display.ready("После ввода нового комплекта чисел \nнажмите кнопку
\"Готово\"");
 }
}

public static void main (String args []) {
  new HCFRepeat();
}
}
```

Тестирование. После запуска программы на экран будет выведено окно, приведенное на рис. 5.1.

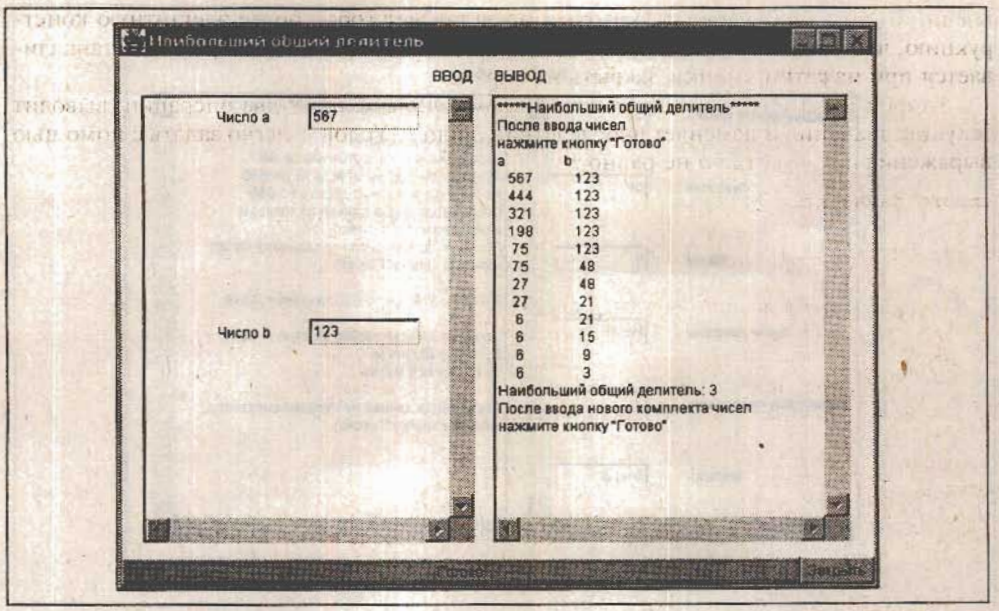


Рис. 5.1. Выходные данные программы вычисления НОД

Пример 5.2. Повседневная работа лавки древностей

Задача. Пришло время усовершенствовать программу, написанную для лавки древностей. После изучения операторов `while` и `if` мы можем добавить в программу следующие функции:

- определение объема продаж за день;
- проверка того, является ли достаточным запас товаров для выполнения продаж в вычисленном объеме.

Решение. Сначала мы определим, можно ли оставить без изменений класс `Curio` или он требует корректировки. Данный класс выводит на печать квитанцию, а в рассматриваемом случае было бы лучше, чтобы класс, моделирующий множество объектов реального мира, давал возможность управлять вводом-выводом тому, кто его вызывает. Для ввода нам необходимо знать, какая древность продается, в каком количестве и когда закрывается магазин. Мы будем использовать класс `Display` и получим результаты, представленные на рис. 5.2.

Алгоритм. Программа состоит из двух частей. Сначала мы создаем окно программы и объекты. Используя метод программирования сверху вниз, мы можем выразить эту часть на высшем уровне, как в следующем примере.

```
report();
stockTheStore();
openTheStore();

while (open) {
  sellCurios();
  open = display.getString("Магазин").equals("Открыт");
  available();
}
 display.println("Магазин закрыт. \n До свидания");
```

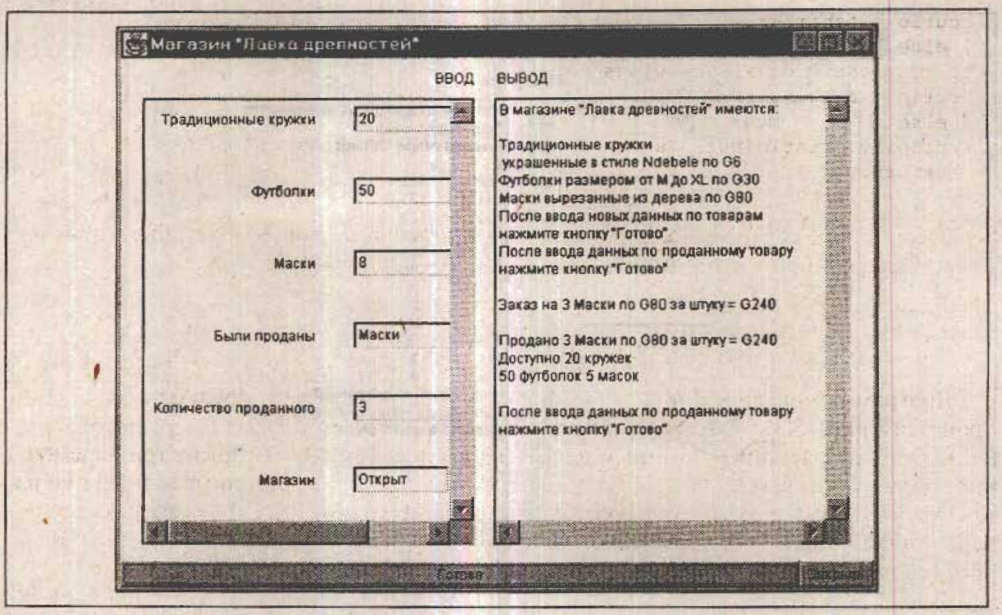


Рис. 5.2. Выходные данные усовершенствованной программы Curio Store

Методы `report` и `available` входили в программу `CurioStore2`. В данной программе мы подчеркнули три метода, которые впоследствии будут усовершенствованы. У нас уже есть некоторый опыт в проведении инвентаризации товаров: мы не охватили только ввод данных. Это несложно сделать вне окна программы. При открытии магазина (нажатии кнопки **Готово**) были добавлены текстовые поля с четвертого по шестое для ввода данных о продажах и указания, что магазин закрыт. Продажа древностей требует выполнения операций ввода-вывода. Данная программа является несовершенной, поскольку метод `sell` класса `Curio`, хранящий информацию о запасах товаров, может принимать отрицательные значения.

```
void sell (int n) {
  // значение наличного товара может стать отрицательным
  stock -= n;
}
```

На этом мы пока завершим рассмотрение первой части программы и выполним ее первый запуск. Конечным аспектом является ввод информации о том, что было продано. Программа должна перебрать все элементы списка, чтобы найти ссылку на объект, например:

```
Curio curio;

  display.ready("После ввода данных по проданному товару \n
                 нажмите кнопку \"Готово\"");
  String curioName = display.getString("Были проданы");
  int Curiosold = display.getInt("Количество проданного");
  if (curioName.equals(mugs.name)) {
   curio = mugs;
  } else
  if (curioName.equals(tshirts.name)) {
```

```
 curio = tshirts;
} else
if (curioName.equals(carvings.name)) {
 curio = carvings;
} else {
  display.println(curioName + " нет в наличии в магазине");
  return;
}
```

Здесь `curio` содержит ссылку на любой из трех объектов и может быть использован в дальнейшем, например:

```
curio.sell(curiosSold);
display.println("\nЗаказ на"+ curiosSold + " " + curioName + " по G"+
curio.price+" за штуку = G" + curio.price*curiosSold);
```

Диаграмма моделирования для метода `sellCurios` демонстрирует одну из стадий процесса (рис. 5.3). Обратите внимание на то, что метод `sellCurios` объекта `CurioStore3` расширен обычным для методов способом. Он содержит три переменные, одна из которых — `Curio`. Если переменная `curioName` принимает значение из текстового поля `Футболки`, она соотносится с именем в объекте `tshirts`, а объектная переменная `curio` получает такое же значение, что отражено на диаграмме. В процессе выполнения оператор

```
curio.sell(curiosSold);
```

ссылается на метод `sellCurios` объекта `tshirts`, т.е. вместо значения 50 устанавливается 46.

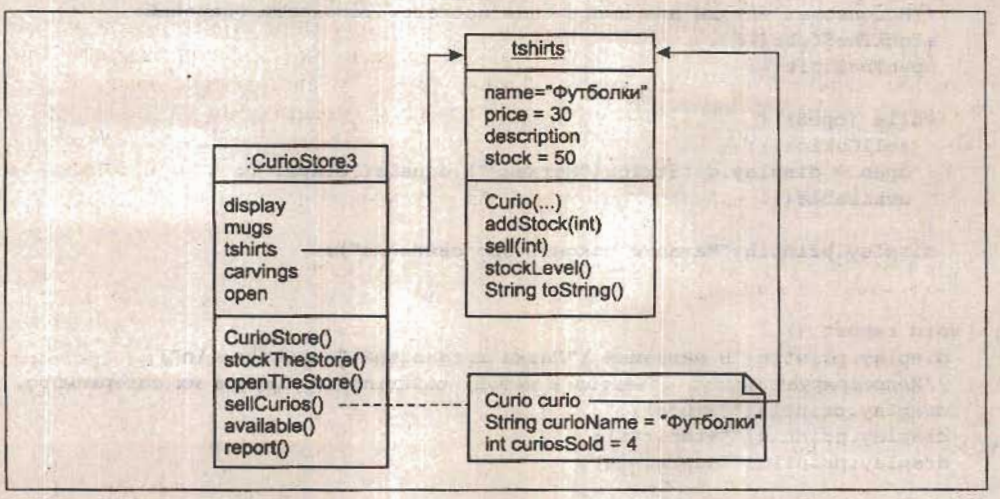


Рис. 5.3. Диаграмма моделирования продаж древностей

Программа

```
import javagently.*;

class CurioStore3 {
  /*Третья версия программы "Лавка древностей" Дж.М.Бишоп, апрель 2000 г.
```

```
 *----------------------------------------
 *Этот магазин имеет свой уровень запасов.
 *Иллюстрирует работу объектно-ориентированной программы.
 *Использует класс Display для организации ввода-вывода.
 */

public static void main (String [ ] args) {
 new CurioStore3 ();
}

//Объявление объектов, связанных со всеми методами.
Display display = new Display ("Магазин \"Лавка древностей\"");
Curio mugs, tshirts, carvings;
boolean open;

//Конструктор, в котором выполняется основная
//часть работы и осуществляется инициализация.
CurioStore3 () {
  mugs = new Curio("Традиционные кружки ", 6, " украшенные в стиле
                   Ndebele", 20, display);
  tshirts = new Curio("Футболки ", 30, "размером от M до XL", 50, display);
  carvings = new Curio("Маски", 80, "вырезанные из дерева", 8, display);

  //Выводит отчет о запасах в магазине на
  //момент начала работы программы.
  report();

  //Использует методы для выполнения последовательности операций.
  stockTheStore();
  openTheStore();

  while (open) {
    sellCurios();
    open = display.getString("Магазин").equals("Открыт");
    available();
  }
  display.println("Магазин закрыт.\nДо свидания");
}

void report () {
 display.println("В магазине \"Лавка древностей\" имеются: \n");
 //Иллюстрирует доступ объектов к методу toString для вывода их содержимого.
 display.println(""+mugs);
 display.println(""+tshirts);
 display.println(""+carvings);
}
 void stockTheStore () {
   display.ready("После ввода новых данных по товарам
                  \nнажмите кнопку \"Готово\"");
   mugs.addStock(display.getInt(mugs.name));
   tshirts.addStock(display.getInt(tshirts.name));
   carvings.addStock(display.getInt(carvings.name));
 }
```

```
    void openTheStore () {
      display.prompt("Были проданы", "                    ");
      display.prompt("Количество проданного", 0);
      display.prompt("Магазин","Открыт");
      open = true;
    }

void sellCurios () {
  Curio curio;

  display.ready("После ввода данных по проданному товару
                \nнажмите кнопку \"Готово\"");
  String curioName = display.getString("Были проданы");
  int CurioSold = display.getInt("Количество проданного");

  if (curioName.equals(mugs.name)) {
     curio = mugs;
  } else
  if (curioName.equals(tshirts.name)) {
    curio = tshirts;
  } else
  if (curioName.equals(carvings.name)) {
    curio = carvings;
  } else {
    display.println(curioName + " нет в наличии в магазине.");
    return;
  }
  display.println("\nЗаказ на "+CurioSold + " " + curioName +" по G"+
                  curio.price+" за штуку = G" + curio.price*CurioSold);

  curio.sell(Curio.Sold);
  display.println("\nПродано "+CurioSold + " " + curioName +" по G"+
                  curio.price+" за штуку = G" + curio.price*CurioSold);
  }

void available () {
  display.println("Доступно "+mugs.stock+" кружек \n "
                 +tshirts.stock+" футболок "
                 +carvings.stock+" масок\n");
  }
}

class Curio {

   String name;
   int price;
   String description;
   int stock;

   Curio (String n, int p, String d, int t, Display display) {
     name = n;
     price = p;
     description = d;
     display.prompt(name,t);
   }
```

```
    void addStock (int n) {
      stock += n;
    }
    void sell (int n) {
      // значение наличного товара может стать отрицательным
      stock -= n;
    }

    int stockLevel () {
      return stock;
    }

    //Метод println получает доступ к данному методу, чтобы
    //преобразовать содержимое объекта Curio в строку.
    public String toString () {
      return name + " "+description+" по G" + price;
    }
}
```

Далее мы рассмотрим конструкцию, которая позволит повысить эффективность обработки информации о продажах при низком уровне запасов. Но прежде мы продолжим разговор о циклах с выходом по условию.

Оператор do

Синтаксис оператора `do` подобен синтаксису оператора `while`. Первый начинает свою работу с того, что хотя бы один раз выполняет тело цикла перед проверкой условия. Хотя данное свойство часто очень полезно, программисты чаще используют цикл `while`.

Оператор do

`do {`
`  statements`
`} while (condition);`

Операторы `statements` выполняются один раз. Затем проверяется условие *`condition`*. Если оно ложно, цикл повторяется. Иначе цикл заканчивается и начинает выполняться следующий за ним оператор.

Разработка цикла с выходом по условию

В процессе разработки циклов с выходом по условию следует принимать во внимание следующие два момента:

- условие должно быть инициализировано;
- условие должно изменяться во время выполнения цикла.

Если условие не инициализируется, цикл будет обрабатывать некорректную или даже неопределенную информацию.

Прежде чем перейти к решению реальной задачи, рассмотрим небольшой пример, который иллюстрирует работу циклов с выходом по условию и в котором используются булевы переменные и методы. В рассматриваемой программе предпринимается попытка найти в ящике пару носков одного цвета. Итак, мы имеем:

```
PickaSock();
PickAnotherSock();
```

```
while (!aPair()) {
  DiscardaSock();
  PickAnotherSock();
}
```

При инициализации цикла предполагается, что у нас в руках находится пара носков. Это необходимо для того, чтобы проверка пары была выполнена корректно. Цикл правильно сформулирован, поскольку условие изменяется каждый раз при выполнении тела цикла, когда выбирается новый носок. Тем не менее, у данного цикла есть две бреши.

Во-первых, не исключено, что пара не может быть составлена. В этом случае условие выхода не будет выполнено, и цикл будет продолжаться бесконечно. Во-вторых, в ящике может не оказаться носков, чтобы начать подбирать пару. Несмотря на то что могут быть выполнены инициализирующие операторы, программа при такой исходной ситуации не может начать работу. Таким образом:

- необходимо предотвратить ситуацию, когда невозможно начать поиск;
- необходимо предотвратить ситуацию, при которой поиск не может быть закончен.

Это можно осуществить за счет использования соответствующих дополнительных условий. Мы должны определить, хватит ли носков (т.е. имеется ли в наличии хотя бы два), чтобы приступить к поиску пары, и проследить, когда ящик опустеет. Оба условия основаны на количестве носков в ящике, а мы берем на себя ответственность за предоставление соответствующего числа. Корректная версия цикла имеет следующий вид:

```
if (NumberofSocksinDrawer >= 2) {
  PickaSock ();
  PickAnotherSock ();
  while (NumberofSocksinDrawer > 0 && !aPair()) {
    DiscardaSock ();
    PickAnotherSock ();
  }
}
{На данном этапе пара уже может быть найдена или еще не найдена}
```

Если циклом управляют сразу несколько частей условия, важно знать в конце, какая из них привела к выходу из цикла. В данном примере следовало бы обеспечить возможность определить, был ли поиск успешным или нет. Это называется **дополнительным действием** и выполняется путем повторной проверки некоторых условий, например:

```
if (aPair()) {
  System.out.println("Подобрана пара носков.");
} else {
  System.out.println("Увы, пара не найдена.")
}
```

Запомните: если условия связаны между собой (а часто так оно и есть), нужно проявлять внимание к тому, какое из них проверяется. В данном случае было бы неправильно тестировать, пуст ящик или нет, следующим образом:

```
if (NumberofSocksinDrawer == 0)
  System.out.println("Увы, пара не найдена.");
else
  System.out.println("Подобрана пара носков.");
```

Ведь пара может быть найдена при самом последнем прохождении цикла. Ящик был бы также пуст, но в данном случае это не имеет значения.

Упражнение. Напишите оператор `if`, формирующий сообщения следующего содержания: найдена пара носков или нет, был ли ящик вначале пустым или стал таковым в процессе поиска.

5.3. Исключения, определяемые пользователем

В разделе 4.6 рассказывалось как реагировать на исключения, генерируемые Java, такие как `EOFException` и `FileNotFoundException`. Мы также можем определить собственное исключение и использовать его для обмена сигналами между классами. Иногда посредством исключений удобнее управлять программой, чем с помощью оператора `if`.

Новое исключение, соответствующее возникшей у пользователя ситуации, определяется следующим образом:

Объявление нового исключения

```
class name extends Exception {
    public name () { }
    public name (String s) {
        super (s)
    }
}
```

name — это имя, которое будет присвоено новому исключению. Второй конструктор передает любую специально определенную строку обработчику исключений нашей системы, благодаря чему она будет доступна для таких методов, как `getMessage`.

Это определение должно быть сделано там, где его смогут увидеть операторы `try` и `catch`, генерирующие или улавливающие его. Сейчас мы имеем класс исключений `name`, а также можем генерировать объекты исключений, созданные на основе этого класса, как показано ниже:

Генерирование исключения

```
throw new name ("message");
```

В сообщении *message* раскрывается суть исключительной ситуации.

Сообщение может быть использовано оператором, обрабатывающим исключения (catch), для предоставления информации пользователю путем вызова метода `getMessage`, или же может оставаться пустым. Определенные пользователем исключения нечасто используются при обычном программировании, поскольку требуемый сигнал может быть получен во время вызова метода, который выдает информацию о состоянии объекта (или его части), а затем применяет оператор `if`. Только когда состояние объекта и то, с чем его нужно сравнить, полностью скрыты в классе, использование исключения имеет смысл. Тем не менее, мы рассмотрим пример, демонстрирующий работу исключений.

Пример 5.3. Проверка наличия товаров на складе

Задача. Требуется отслеживать объем запасов в лавке древностей и отклонять заказы, в которых указано большее количество товаров, чем имеется в магазине.

Решение. Данная задача может быть решена двумя способами. Мы можем использовать оператор `if` и вызвать метод `stockLevell` или же применить исключение.

Программы. Первый вариант прост:

```
if (curio.stock <= CurioSold) {
  curio.sell(CurioSold);
    display.println("Заказ на "+ CurioSold + " " + curioName +" по G"+
                    curio.price+" за штуку = G" + curio.price*CurioSold);
  }
  else {
    display.println("Извините, но в магазине имеется только "+ curio.stock + "
                    \nединиц данного товара.");
}
```

А вот второй вариант, в котором используются исключения:

```
try {
   curio.sell(CurioSold);
   display.println("Заказ на "+ CurioSold + " " + curioName +" по G"+
                   curio.price+" за штуку = G" + curio.price*CurioSold);
}
catch (StockException e) {
  display.println(e.getMessage());
}
```

Чтобы активизировать исключительную ситуацию, мы должны объявить ее, указать, что мы собираемся ее отбросить, а затем отбросить ее. Отброс исключительной ситуации предполагает применение оператора `if`. Объявление имеет вид:

```
class StockException extends Exception {
    StockException (String s) {
       super(s);
    }
}
```

Метод `sell` в классе теперь имеет следующий вид:

```
void sell (int n) throws StockException {
  if (stock >= n) {
    stock -= n;
  }
  else {
    throw new StockException
      ("Извините, но в магазине имеется только "+stock+"
       \nединиц данного товара.");
  }
}
```

Возможный набор выходных данных показан на рис 5.4

Удачный пример исключения, определенного пользователем, приведен в разделе 15.2.

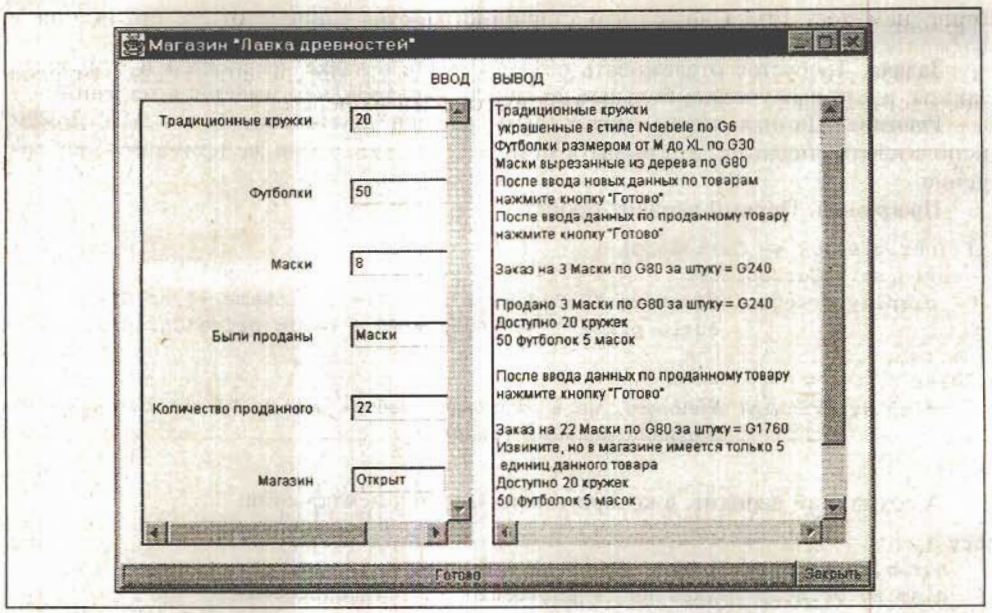


Рис. 5.4 Выходные данные программы

5.4. Тип char

Строка в Java — это объект. Даже одиночный символ, взятый в кавычки (например, "a"), является строкой. В Java предусмотрен более простой тип для одиночных символов, вводимых с клавиатуры. Обычно это:

- Символы от A до Z
- Символы от a до z
- ! @ # $ % ^ & * ( ) _ + - = {} [] : " | ; ' ((? /

Дополнительно мы можем представить еще ряд команд, позволяющих вводить ESC-символы:

- \b - возврат на позицию назад со стиранием символа;
- \t - табуляция;
- \n - новая строка;
- \f - новая страница.

Поскольку нажатие этих клавиш ([Tab], [Enter]) приводит к выполнению соответствующих действий, указать их в тексте программы можно только таким образом.

Символы в Java заключаются в одинарные кавычки (например, `'a'`). Для них действительными являются следующие операции:

- присвоение;
- сравнение;
- объединение в строки.

Результат сравнения двух символов с целью обнаружения наибольшего или наименьшего зависит от порядка, который установлен в Java для символов. На данном

этапе нам достаточно знать, что старшинство букв зависит от их положения в алфавите (так, 'g' всегда меньше 'z').

Существует несколько полезных методов класса, определенных для символов посредством класса Character. Ниже представлен их перечень.

Методы класса Character
`int digit (char ch, int radix)`
`char fordigit (int digit, int radix)`
`int getNumericValue(char ch)`
`boolean isDigit(char ch)`
`boolean isLetter(char ch)`
`boolean isLowerCase(char ch)`
`boolean isUpperCase(char ch)`
`char toLowerCase(char ch)`
`char toUpperCase(char ch)`
`... и 13 других`

На первый взгляд может показаться, что эти методы созданы для определенных примеров обработки символов. К сожалению, считывание символов в Java требует привлечения некоторых более изощренных принципов, рассмотрение которых мы отложим до главы 7. А тем временем мы будем интенсивно применять в работе нужные нам методы, пока изложение книги не приведет нас к этой главе. Существует один строковый метод, который пригодится нам уже в самое ближайшее время, например, в таком выражении:

```
char charAt(int index)
```

Если мы считываем символ как строку, метод `charAt(0)` преобразует ее в тип `char`.

Наконец, Java предоставляет в распоряжение пользователей посредством управляющего символа \u пространство для дополнительных 10000 символов, которые недоступны на обычных клавиатурах. Это символы в формате Unicode, которые делают Java интернациональным языком. Существуют последовательности Unicode для букв кириллицы, греческого, древнееврейского, арабского, тибетского и многих других алфавитов. Все они соответствуют интернациональным стандартам. Эти тонкости пока не представляют для нас особого интереса, поскольку мы будем писать программы, используя символы латинского алфавита.

Преобразование символов

Символы могут быть преобразованы в числа и наоборот. Следующее преобразование выполняется автоматически, следовательно, все числовые операторы применимы к символам:

```
char может быть преобразован в int или long или float или double
```

Для обратного преобразования следует выполнить операцию приведения типа. Так, действительной является следующая последовательность операторов:

```
char digit;
int n;
n = digit;
n++;
digit = (char) n;
```

В результате выполнения этой последовательности указанный символ будет увеличен на одно значение. Например, если переменная `digit` имела значение `'j'`, то оно станет равным `'k'`. Поскольку для символов уже определены аккуратные методы, маловероятно, что нам понадобится преобразовывать их в целые числа, если только не потребуется использовать их в вычислениях, как в следующем примере.

Пример 5.4. Вывод на печать набора символов

Задача. Необходимо определить, какие дополнительные символы доступны в нашей системе.

Решение. Мы можем организовать цикл, в котором перебираются числа, начиная с нуля, и для каждого из них выводится соответствующий ASCII-символ. В результате пробного запуска программы было установлено, что печатаемые символы начинаются только с 32-го ASCII-символа.

Программа. Хотя программа довольно проста, мы обнаружим в ней несколько новых функциональных возможностей. В программе два раза для символьной переменной выполняются арифметические операции (соответствующие фрагменты подчеркнуты). Для обозначения границ цикла (начального и конечного значений) используется шестнадцатеричное представление символов в формате Unicode. Выводятся эти значения в десятичном представлении. В цикле числа перебираются от 32 до 255.

```
import java.awt.*;
import javagently.*;

class TestUnicode {

  /* Программа для демонстрации системы Unicode.
   * ========================================
   *            автор - Дж. М. Бишоп май 2000 г.
   *
   * Демонстрирует преобразование символов, а также использование
   * шестнадцатеричного кода в формате Unicode.
   */

TestUnicode() {
   for (char code = "\u0020"; code <="\u00FF"; code++) {
      System.out.print(Text.format(code,5) + " " + code);
      if (code%10 == 0) {
        System.out.println();
      }
   }
}

public static void main (String [] args) {
new TestUnicode();
}
}
```

Тестирование. В результате выполнения программы мы получим следующий результат:

```
32    33  ! 34  " 35  # 36  $ 37  % 38  & 39  ' 40  (
41  ) 42  * 43  + 44  , 45  - 46  . 47  / 48  0 49  1 50  2
51  3 52  4 53  5 54  6 55  7 56  8 57  9 58  : 59  ; 60  <
61  = 62  > 63  ? 64  @ 65  A 66  B 67  C 68  D 69  E 70  F
71  G 72  H 73  I 74  J 75  K 76  L 77  M 78  N 79  O 80  P
81  Q 82  R 83  S 84  T 85  U 86  V 87  W 88  X 89  Y 90  Z
91  [ 92  \ 93  ] 94  ^ 95  _ 96  ' 97  a 98  b 99  c 100 d
101 e 102 f 103 g 104 h 105 i 106 j 107 k 108 l 109 m 110 n
111 o 112 p 113 q 114 r 115 s 116 t 117 u 118 v 119 w 120 x
121 y 122 z 123 { 124 | 125 } 126 ~ 127   128 ? 129 ? 130 ?
131 ? 132 ? 133 ? 134 ? 135 ? 136 ? 137 ? 138 ? 139 ? 140 ?
141 ? 142 ? 143 ? 144 ? 145 ? 146 ? 147 ? 148 ? 149 ? 150 ?
151 ? 152 ? 153 ? 154 ? 155 ? 156 ? 157 ? 158 ? 159 ? 160
161 Ў 162 ў 163 J 164 ? 165 Г 166 | 167 § 168 Е 169 © 170 Є
171 « 172 ¬ 173  174 ® 175 Ї 176 ° 177 ± 178 2 179 3 180 '
181 µ 182 ¶ 183 . 184 , 185 1 186 є 187 » 188 j 189 S 190 s
191 ї 192 А 193 Б 194 В 195 Г 196 Д 197 Е 198 Ж 199 З 200 И
201 Й 202 К 203 Л 204 М 205 Н 206 О 207 П 208 <ETH> 209 С 210 Т
211 У 212 Ф 213 Х 214 Ц 215 * 216 Ш 217 Щ 218 Ъ 219 Ы 220 Ь
221 Э 222 <THORN> 223 Я 224 а 225 б 226 в 227 г 228 д 229 е 230 ж
231 з 232 и 233 й 234 к 235 л 236 м 237 н 238 о 239 п 240 <eth>
241 с 242 т 243 у 244 ф 245 х 246 ц 247 ч 248 ш 249 щ 250 ъ
251 ы 252 ь 253 э 254 <thorn> 255 я
```

5.5. Оператор switch

Условный оператор `if` предоставляет возможность выбора выполняемого оператора при наличии нескольких условий. Предположим, необходимо произвести несколько простых проверок для заданных значений. Конструкция, которую мы получили бы, указав несколько последовательных операторов `if-else`, является довольно громоздкой. Благодаря оператору `switch` становится возможным **отбор с использованием ключа**. Ниже показан синтаксис этого оператора.

Оператор switch

```
switch (switch-expression) {
     case value : statement; break;
     case value : statement; break;
     ...
     default : statement; break;
}
```

Выполняется "оценка" выражения `switch-expression`, которое должно иметь один из целочисленных типов или тип `char`. Результат вычисления данного выражения сравнивается с перечисляемыми ниже значениями `value`. Если такое значение имеется, сначала выполняется соответствующий оператор `statement`, а затем — все остальные операторы, пока не встретится оператор `break` или не будет достигнут конец оператора `switch`. Если в операторе `switch` отсутствует проверяемое значение, выполняется оператор, определенный по умолчанию (относящийся к `default`). В теле оператора значение может быть указано только один раз.

Оператор `switch` рассматривает значение, указанное в выражении `switch-expression`, и, начиная с первого варианта, пытается найти совпадение. При обнаружении совпадения выполняется соответствующий оператор, и, если после него указан оператор `break`, управление передается в конец оператора `switch`. Употребление ключевого слова `default` обеспечивает перехват всех остальных значений, которые не указаны в теле `switch`. Использование операторов `break` и ключевого слова `default` с определенным для него оператором не является обязательным, но считается в программировании хорошим тоном. При отсутствии оператора `break` управление переходит к следующему указанному в теле оператору. Этот принцип можно выгодно использовать, если для нескольких значений-вариантов нужно выполнить один и тот же оператор. В этом случае все значения, кроме последнего, перечисляются без оператора `break`, который определяется для последнего значения. В следующем примере показано, как все это применять на практике.

В результате вычисления выражения `switch-expression` мы получаем значение целочисленного типа или типа `char`. Оно не может быть рациональным числом. Значения, указанные в вариантах `case`, имеют тот же тип, что и значения `switch-expression`. Как отмечалось ранее, для одного оператора можно задать как один, так и несколько вариантов. Не имеет значения, в какой последовательности представлены варианты, важно, чтобы каждое значение указывалось только один раз. В качестве примера рассмотрим небольшой стишок, в котором объясняется, сколько в каждом месяце дней:

И в июне, и в апреле, в сентябре и ноябре,
По четыре есть недели, 30 дней в календаре.
Остальные все похожи, дней у них тридцать один,
Лишь февраль один не может выбрать из возможных длин.
Он длиною в 28 дней в обычный год,
А вот в каждый високосный, еще день себе берет

Основываясь на том, что переменная `month` имеет значение целочисленного типа и каждому проверяемому месяцу ставится в соответствие некоторое число (январю — 1, февралю — 2 и т.д.), с помощью оператора `switch` можно написать метод, позволяющий определить количество дней в месяце (исключая високосный год):

```
switch (month) {
  case 9:
  case 4:
  case 6:
  case 11: days = 30; break;
  case 2: days = 28; break;
  default: days = 31; break;
}
```

Оператор, указываемый после каждого варианта, может представлять собой блок и включать несколько операторов. Эта возможность используется в следующей версии метода, в которой учитываются високосные года и для февраля указывается корректное количество дней. Вместо того чтобы выполнять вычисления, определяющие, какой это год (что несложно сделать: достаточно выполнить деление на 4, и если результат равен 0, значит, год високосный), мы сделаем так, чтобы методу, в котором находится оператор `switch`, в качестве параметра передавалось булево значение, указывающее, високосный год (`leapyear`) или нет. В следующей версии операторы `break` являются излишеством, поскольку оператор `return` выполняет такую же функцию.

```
int daysInMonth (int month, boolean leapyear) {
  switch (month) {
```

```
    case 9:
    case 4:
    case 6:
    case 11:
      return 30;
    case 2: {
      if (leapyear) {
        return 29;
      }
      else {
        return 28;
      }
    }
    default: return 31;
  }
}
```

В качестве альтернативы мы могли бы передать год как параметр и выполнить в методе `daysInMonth` описанные выше вычисления. В следующем примере в центре внимания — операторы `switch`. Выражениями `switch-expression`, как правило, являются символы (тип `char`).

Пример 5.5. Курсы обмена валют

Задача. Лавку древностей часто посещают иностранные туристы, которым удобнее платить за покупки любой из основных валют.

Решение. На панель ввода будет добавлено текстовое поле для указания символа валюты. После подсчета суммы в местных денежных единицах грацах, которую должен заплатить покупатель, мы вводим символ валюты и посредством оператора `switch` находим соответствующий курс для пересчета. На экран выводится сумма в указанной валюте и в грацах (рис. 5.5).



Рис. 5.5. Выходные данные программы Curio Store, включающей процедуру конвертирования валюты

Алгоритм. Данная задача решается в два этапа: 1 — ввод символа валюты, 2 — конвертирование стоимости покупок с последующим выводом полученного значения на экран. Очевидно, что использование оператора `switch` является идеальным вариантом для второй части. Вот как он может выглядеть:

```
switch (currencySymbol) {
  case 'Y': factor = yenExchange; break;
  case '$': factor = dollarExchange; break;
  case 'D': factor = markExchange; break;
  case 'P': factor = poundFactor; break;
  case 'G': factor = 1; break;
  default: ...;
}
```

Это наиболее понятный и эффективный, но не единственный способ решения данной задачи. Того же результата можно достичь, используя последовательность операторов `if`, как описано в разделе 4.4. Но в перечне вариантов оператора `switch`, который чем-то похож на таблицу, учитываемые варианты представлены более наглядно и компактно. Поэтому, если ситуация позволяет применить оператор `switch`, предпочтение следует отдать именно ему.

Что должно выполняться в операторе, соответствующем ключевому слову `default`, которому перейдет управление при вводе некорректного символа? Очевидно, должно быть выведено сообщение, информирующее пользователя о том, что введенный символ недействителен. Но как вновь перейти к вводу? Один из вариантов — объявление и генерирование объекта-исключения. Но это будет излишним, поскольку у нас есть вся информация, которая необходима для обработки такой ситуации. Кроме того, мы неоднократно говорили, что объекты-исключения предназначаются для коммуникаций между классами.

Итак, мы объявим булеву переменную, которой изначально присваивается значение `true`, а при вводе неверного символа валюты — значение `false`. Для этого мы поместим оператор `switch` в оператор `do-while`, который анализирует значение булевой переменной.

```
void sellCurios () {
  Curio curio;
  double factor;
  double yenExchange =  0.04;
  double dollarExchange = 5.25;
  double markExchange = 2.8;
  double poundFactor = 8.16;
  char currencySymbol;
  boolean symbolRead = true;

  display.ready("После ввода данных по проданному товару \nнажмите кнопку
          \"Готово\"");
  String curioName = display.getString("Были проданы");
  int CurioSold = display.getInt("Количество проданного");

  if (curioName.equals(mugs.name)) {
     curio = mugs;
  } else
  if (curioName.equals(tshirts.name)) {
    curio = tshirts;
  } else
```

```
  if (curioName.equals(carvings.name)) {
    curio = carvings;
  } else {
    display.println(curioName + "нет в наличии в магазине.");
    return;
  }

    currencySymbol = display.getString("Используемая валюта").charAt(0);
    switch (currencySymbol) {
      case 'Y': factor = yenExchange; break;
      case '$': factor = dollarExchange; break;
      case 'D': factor = markExchange; break;
      case 'P': factor = poundFactor; break;
      case 'G': factor = 1; break;
      default : factor = 1;
            symbolRead = false;
            display.println("Неправильный символ, попытайтесь еще раз");
    }

  display.println("\nСумма к оплате за: "+CurioSold + " " + curioName +"
     по G"+ curio.price+" \nза единицу товара = G"+curio.price*CurioSold);

  curio.sell(CurioSold);

  display.println("\nПродано "+CurioSold + " " + curioName +" по G"+
        curio.price+" \nза единицу товара = G"+curio.price*CurioSold);

   if (currencySymbol != 'G') {

    display.println("что составляет "+ currencySymbol +

    Stream.format(curio.price*CurioSold/factor,10,2));
  }
}
```

В следующих версиях программы Curio Store будет предусмотрена возможность ежедневного изменения курсов валют.

В каких случаях лучше не использовать оператор switch

Понятность оператора `switch` делает его применение закономерным в тех случаях, когда требуется сделать выбор из ряда значений. Однако его нельзя использовать, когда выбор базируется на условиях. Например, следующий оператор `switch` не будет действительным в Java:

```
/* Недопустимо в языке JAVA */
switch (number) {
  case <  0: Addtonegatives;
  case == 0: Donothing;
  case >  0: Addtopositives;
}
```

Вариантами выбора должны быть целочисленные значения или символы. Для указания этих условий подойдет последовательность операторов `if-else`.

Оператор `switch` неприменим также для проверки строк, которые являются более сложными элементами, чем те, с которыми он может работать. В следующей главе мы увидим, как согласовать (связать в одну структуру) строки, которые будут выступать в качестве ключей, и значения (раздел 6.5).

5.6. Практикум 2. Игра "Камень-ножницы-бумага"

Это популярная игра, рассчитанная на двух человек. Каждый из игроков выбирает "оружие", которое он будет использовать в споре с противником. Это могут быть камень, ножницы или бумага. Победитель определяется по следующим принципам:

- камень бьет ножницы (он может разбить их);
- ножницы бьют бумагу (они могут разрезать ее);
- бумага бьет камень (камень можно в нее завернуть).

Если оба игрока выбрали один и тот же предмет, тогда исходом игры будет ничья. Мы бы хотели запрограммировать компьютер так, чтобы он играл в эту игру с пользователем. Каждый делает свой выбор, и компьютер определяет победителя.

Решение

Чтобы запрограммировать любую игру, рассчитанную на двоих, следует пройти следующие этапы:

- вывести на экран инструкции (или опустить их по требованию);
- инициализировать начальное состояние игры;
- организовать цикл, позволяющий играть много раз, пока не будет дана команда завершить игру;
- сгенерировать выбор компьютера;
- получить информацию о выборе пользователя;
- решить, кто из двоих выиграл;
- сыграть следующую партию или выйти из игры.

Мы рассмотрим поочередно каждый этап.

Этап 1. Инструкции

Компьютерные игры принято снабжать инструкциями, которые включают не только правила игры, но и дополнительные сведения, например о том, как вводить ответы. Всегда следует указывать две вещи: нужно ли после ввода ответа нажимать клавишу [Enter] и воспринимаются ли прописные буквы так же, как и строчные. К счастью, в Java мы можем использовать символьный метод (метод класса `Character`) для изменения любых входных данных с целью представления их в том или ином регистре.

Этап 2. Инициализация

Как же компьютер сделает свой выбор? Желательно, чтобы он осуществлял выбор случайным образом, что возможно при использовании генератора случайных чисел. Нам подойдет простой генератор случайных чисел класса `Math`, если мы умножим значение на 3 и присвоим ему тип `integer`. Какие же значения мы получим? Если мы начнем со значений от 0 до 0.999, то получим значения от 0 до 2.999. Выполнив приведение типа, мы получим целое число, лежащее в диапазоне от 0 до 2. (Вы можете проверить это с помощью небольшой тестовой программы.)

Альтернативой является более сложный генератор целых чисел из пакета `java.util`. Он работает с объектами `Random`, где генерируемая последовательность чисел может быть инициализирована из **определяющего числа**. Если определяющее число вводится пользователем, у компьютера остается мало шансов всегда выигрывать. Мы можем использовать этот класс, чтобы смоделировать выбор одного из трех возможных видов "оружия", объявив:

```
Random dice = new Random (in.readInt());
```

В результате вызова метода `nextInt()` для объекта `dice` будет возвращено целое число, относящееся к заданному диапазону, которое имеет тип `int`, т.е. может быть положительным или отрицательным. Чтобы получить из него необходимые нам значения (0, 1 или 2), мы сделаем его положительным (операция `abs`) и разделим по модулю 3:

```
mychoice = Math.abs(dice.nextInt()) % 3;
```

Этап 3. Организация цикла

Мы собираемся использовать неопределенный цикл `while`, поскольку выход из цикла (при нажатии клавиши [Q]) осуществляется между получением нашего выбора и получением выбора, сделанного компьютером. Для выхода из цикла применяется оператор прерывания `break`. Цикл является основной чертой главного класса, который "играет" в игру, и имеет следующий вид:

```
import java.io.*;
import javagently.*;

class PlayGame {

  /* Игровая программа. Дж.М.Бишоп, август 1996 г.
   * ----------------  Java 1.1
   *                   Обновление - июнь 2000 г.
   * Драйвер для игры "Камень-ножницы-бумага".
   * Вызывает методы класса RSP, чтобы получить
   * информацию о выборе играющего и выборе компьютера,
   * отображает данные о том, кто победил.
   * Для завершения игры необходимо нажать клавишу Q.
   * Демонстрирует использование цикла с выходом по условию.
   */

  public static void main (String [] args) throws IOException {
    new PlayGame ();
  }

  PlayGame () throws IOException {

    // Эта строка будет изменяться для каждой игры.
    RSPGame mygame = new RSPGame ();

    mygame.startGame ();
    while (true) {
      mygame.makemyChoice();
      if (mygame.getyourChoice ()=='Q') break;
      mygame.winner();
    }
    System.out.println("Спасибо за участие в игре");
  }
}
```

Этап 4. Генерирование выбора компьютера

Выбор компьютера будет осуществляться с помощью генератора случайных чисел. Компьютер будет передавать одно из числовых значений — 0, 1 или 2, а пользователь — символы `R`, `S` или `P`. *Фактически это не составит проблем при* написании метода.

Этап 5. Генерирование выбора пользователя

На данном этапе следует быть особенно внимательным. Обязательно следует убедиться в том, что введенный символ является одним из тех, употребление которых разрешено. Если это не так, необходимо вывести сообщение и попросить пользователя повторить ввод. Для решения данной задачи нам подойдет цикл do, поскольку в нем проверка производится после того, как данные введены. Базовая конструкция имеет следующий вид:

```
do {
  System.out.print ("Введите R S P, или для выхода - Q");
  yourchoice; = Text.readChar(in);
} while (yourchoice!='R' & yourchoice != 'S'
       & yourchoice != 'p' & yourchoice != 'Q');
```

Эта простая последовательность заставляет игрока вводить заглавные буквы. Данная программа имеет и более сложную версию. Обратите внимание на то, что начальное сообщение служит также сообщением об ошибке.

Этап 6. Выбор победителя

Это наиболее сложная часть алгоритма. Компьютер может выбрать один из трех вариантов. Результат будет зависеть от того, какой выбор сделал игрок. Таким образом, существует девять возможных комбинаций. Для решения данной задачи мы воспользуемся методом с параметрами.

Рассмотрим первый вариант, когда компьютер выбирает камень. В зависимости от выбора игрока компьютер оценивает результат следующим образом:

- игрок выбрал ножницы — компьютер победитель;
- игрок выбрал бумагу — компьютер проигравший;
- игрок выбрал камень — ничья.

Эта процедура повторяется компьютером для случая, когда выбраны ножницы, за исключением того, что в списке выборов игрока элементы расположены в другом порядке. Очевидно, что наиболее подходящим средством в данном случае является метод (report), что и продемонстрировано в следующей программе.

Этап 7. Выход из игры

Выход из игры можно сделать очень простым. Например, мы можем вывести благодарность пользователю или статистику о том, сколько игр выиграл каждый игрок.

Программа. В программе эффективно используются все операторы управления, которые были представлены в настоящей главе — while, do и switch. Программа состоит из двух классов. Структура класса playGame практически не зависит от игры, за исключением того, что он создает объект mygame, относящийся к определенному классу, в данном случае — RSPGame. Класс RSPGame содержит три закрытые переменные и один закрытый метод, которые не должны быть видны вне объекта игры. На рис. 5.6 представлена диаграмма моделирования для данной программы.

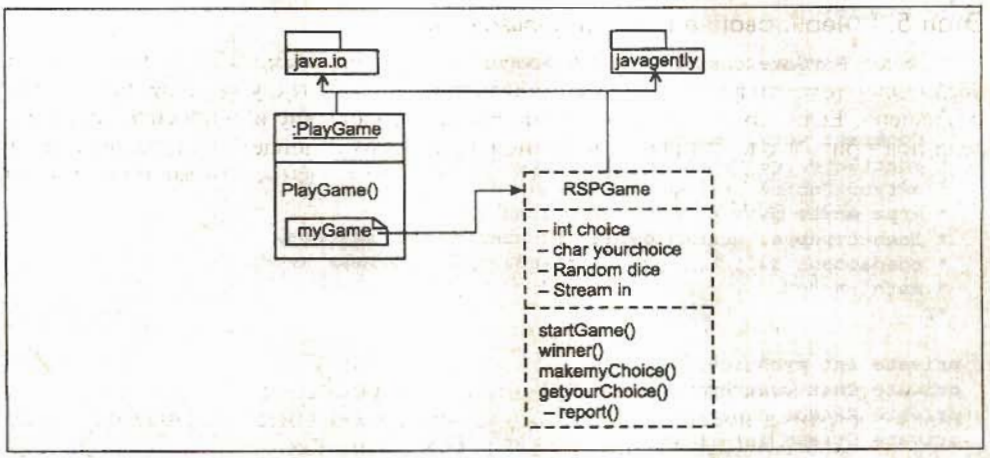


Рис. 5.6. Диаграмма моделирования для игры "Камень-ножницы-бумага"

```
import java.io.*;
import javagently.*;

class PlayGame {

  /* Игровая программа. Дж.М.Бишоп, август 1996 г.
   * ----------------- Java 1.1
   *                   Обновление - июнь 2000 г.
   * Драйвер для игры "Камень-ножницы-бумага".
   * Вызывает методы класса RSP, чтобы получить
   * информацию о выборе играющего и выборе компьютера,
   * выводит данные о том, кто победил.
   * Игра завершается при нажатии клавиши Q.
   * Демонстрирует работу цикла с выходом по условию.
   */

  public static void main (String [] args) throws IOException {
    new PlayGame ();
  }

  PlayGame () throws IOException {

    RSPGame mygame = new RSPGame ();
    mygame.startGame ();

    while (true) {
      mygame.makemyChoice();
      if (mygame.getyourChoice ()=='Q') break;
      mygame.winner();
    }
    System.out.println("Спасибо за участие в игре");
  }
}
import java.io.*;
import javagently.*;
import java.util.Random;
```

```
class RSPGame {

  /* Класс RSPGame, автор -  Дж М Бишоп, август 1996 г.
   * ------------------      Java 1.1 - декабрь 1997 г.
   *                         Обновление - июнь  2000 г.
   * Содержит четыре метода:
   * startGame, getmyChoice, getyourChoice и Winner.
   * getyourChoice возвращает символ, поэтому
   * игра может быть остановлена, если это Q.
   * Демонстрирует использование типизированных методов,
   * операторов switch, циклов, генератора случайных чисел
   * Math.random.
   */

  private int mychoice;
  private char yourchoice; // R S P
  private Random dice;
  private Stream in = new Stream(System.in);

  void startGame () throws IOException {
    System.out.println("Давайте поиграем в игру \"Камень-ножницы-бумага\"");
    System.out.println("Чтобы убедиться в том, что я не жульничаю,"+
                       " инициализируйте меня, введя число.");
      dice = new Random(in.readInt());
  }

  void winner () {
    // При вызове метода report первым параметром является выбор
    // компьютера.
    // Второй параметр - это выбранный вариант, который может быть побит
    // компьютером.
    // Третий параметр - это вариант, когда компьютер может быть побит.
    // R - rock, камень; S - scissors, ножницы; P - paper, бумага.

    switch (mychoice) {
      case 0 : report ('R','S','P'); break;
      case 1 : report ('S','P','R'); break;
      case 2 : report ('P','R','S'); break;
    }
  }

  void makemyChoice () {
    // nextInt возвращает число типа integer, принадлежащее заданному
    // диапазону.
    // Мы должны свести его к 0, 1 или 2.
    mychoice = Math.abs(dice.nextInt()) % 3;
  }

  char getyourChoice () throws IOException {
    do {
      System.out.print("Введите R, S, P или Q для выхода");
      yourchoice = Character.toUpperCase(
           in.readString().charAt(0));
    }  while (yourchoice !='R' && yourchoice != 'S'
            && yourchoice != 'P' && yourchoice != 'Q');
    if (yourchoice != 'Q') {
      System.out.print("Вы выбрали "+yourchoice+", а ");
```

```
    }
    return yourchoice;
  }

  private void report (char me, char Iwin, char youWin) {
    System.out.println("я выбрал " + me);
    if (yourchoice == Iwin) {
      System.out.println("Я выиграл!");
    } else
    if (yourchoice == youWin) {
      System.out.println("Вы выиграли!");
    }
    else {
      System.out.println("Ничья");
    }
  }
}
```

Тестирование. Данные, выводимые на экран в результате выполнения программы, приблизительно выглядят следующим образом:

```
Давайте поиграем в игру «Камень-ножницы-бумага».
Чтобы убедиться в том, что я не жульничаю, инициализируйте меня, введя число.
67
Введите R, S, P или Q для выхода R
Вы выбрали R, а я выбрал S
Вы выиграли!
Введите R, S, P или Q для выхода S
Вы выбрали S, а я выбрал R
Я выиграл!
Введите R, S, P или Q для выхода P
Вы выбрали P, а я выбрал R
Вы выиграли!
Введите R, S, P или Q для выхода P
Вы выбрали P, а я выбрал P
Ничья
Введите R, S, P или Q для выхода S
Вы выбрали S, а я выбрал P
Вы выиграли!
Введите R, S, P или Q для выхода R
Вы выбрали R, а я выбрал S
Вы выиграли!
Введите R, S, P или Q для выхода. Q
Спасибо за участие в игре
```

Как видите, выбор компьютера абсолютно случайный, и он иногда проигрывает!

Коллективная игра

Коллективные игры очень популярны. "Камень-ножницы-бумага" рассчитана на двух игроков, но мы можем, не затрачивая больших усилий, показать, каким образом компьютер, подобно гроссмейстеру, может играть несколько партий одновременно. Для этого нам потребуются подпроцессы, которые мы изучим в главе 13.

Резюме

Цикл `while` является основным управляющим циклом. Он проверяет условие в начале цикла. Цикл `do` проверяет условие в конце. Оператор `break` позволяет проверить одно или несколько условий в середине цикла и выйти из него. При программировании циклов `while` полезно использовать булевы переменные для хранения значений установленных условий (`true` или `false`), пока не придет время проверять, продолжать выполнение цикла или нет. Использование циклов `while` тесно связано с чтением входных данных, при этом требуется обработка некоторых результирующих данных посредством объектов-исключений.

Оператор `switch` обеспечивает выбор действия из большого количества возможных вариантов в зависимости от значений. Ряд значений может требовать выполнения одного и того же оператора. Значениями могут быть только целые числа или символы.

В данной главе рассматривалось употребление следующих ключевых слов:

```
case        public
char        switch
default     throw
do          while
private     extends
```

Описаны такие компоненты библиотеки языка `Java`:

```
Exception           isDigit
super               isLetter
Character           isLowerCase
digit               isUpperCase
forDigit            toLowerCase
getNumericValue     toUpperCase
```

Следующий перечень включает темы, на которых в главе сделано особое ударение:

- контроль доступа;
- оператор `while`;
- оператор `do`;
- определение новой исключительной ситуации;
- генерирование исключительной ситуации;
- методы класса `Character`;
- оператор `switch`.

Вопросы



5.1. Если метод не имеет модификатора доступа, то он будет виден для:

а) объектов его класса

б) любого объекта класса его пакета

в) любого другого объекта

г) не будет виден

5.2. При выполнении следующего оператора `while` сколько символов будет выведено, если в качестве обрабатываемых данных выступает слово KALAMZOO?

```
int ch = ' ';
while (ch != 'Z') {
  System.out.println(ch);
}
```

а) 6

б) 5

в) 0

г) невозможно определить

5.3. После выполнения следующего оператора `switch` каким должно быть значение переменной `value`?

```
int coin = 5;
switch (coin) {
  case 1: value = 0.01;
  case 2: value = 0.02;
  case 3: value = 0.05;
  case 4: value = 0.20;
  case 5: value = 0.50;
  case 6: value = 1.00;
  case 7: value = 2.00;
  default: value = 0;
}
```

а) 0.50

б) 2.00

в) 5

г) 0

5.4. Что произойдет, если на вход оператора `switch` поступит значение, для которого не определен оператор после ключевого слова `case`, а также отсутствует вариант выбора `default`?

а) возникнет ошибка исполнения

б) будет выбран произвольный оператор из числа существующих в теле структуры `switch`

в) будет выполнен последний из имеющихся операторов

г) не будет выполнен ни один из операторов, определенных внутри `switch`, и программа продолжит свою работу

5.5. Цикл `while` завершает свою работу, когда:

а) определенное для его выполнения условие принимает значение `true`

б) определенное для его выполнения условие принимает значение `false`

в) определенное для его выполнения условие принимает значение `true` или выполняется оператор `break`

г) определенное для его выполнения условие принимает значение `false` и выполняется оператор `break`

5.6. Символьное значение, следующее за `ch` в алфавитном порядке, может быть получено с помощью выражения

а) `ch ++`

б) `nextChar(ch)`

в) `((int) ch) ++`

г) получить невозможно

5.7. Когда мы генерируем определенный пользователем объект-исключение, таковым является:

а) класс

б) переменная

в) объект

г) формальный параметр

5.8. Как сделать более эффективным условие, определенное в практикуме 2?.

```
while (yourchoice! = 'R' & yourchoice != 'S'
            & yourchoice!= 'P' & yourchoice != 'Q');
```

а) заменить операторы `&` операторами `&&`

б) заключить в скобки каждое из четырех сравнений

в) переписать его следующим образом:

```
while (yourchoice != {'R', 'S', 'P', 'Q'});
```

г) повысить эффективность невозможно

5.9. Если преобразовать цикл `while` в программе по определению максимального общего множителя в цикл `do-while`, результирующий код будет выглядеть следующим образом:

а) `do {if (a>b) a -=b else b-=a;} while (a != b);`

б) `do {if (a>b) a -=b else b-=a;} while (a == b);`

в) `if (a!=b) do {if (a>b) a -=b else b-=a;} while (a != b);`

г) `if (a!=b) do {if (a>b) a -=b else b-=a;} while (a == b);`

5.10. Принцип, согласно которому операции ввода-вывода определяются не в классе, который описывает объект, а в главном классе программы, который создает экземпляр класса первого типа:

а) разделение обязанностей

б) антислияние

в) сокрытие информации

г) закрытый доступ

Задачи

Примечание. Первые восемь задач посвящены корректному формулированию циклов. Большинство из них рассчитано на использование циклов `while` и `do-while`, но можно применять и циклы `for`.

5.1. **Рулон пленки.** Сотрудница библиотеки обвертывает книги пленкой. Она использует рулон шириной 2,2 метра и длиной 100 метров. Размер книг лежит в диапазоне от 20—14 до 21—30 см. Чтобы обернуть книгу, требуется кусок пленки, размер которого составляет два размера книги плюс по 4 см с каждой стороны. Если для поступившей книги не подходит оставшийся кусок, сотрудница отрезает от рулона новую полосу. Выберите произвольные размеры для 20 книг и покажите, как можно эффективно использовать рулон указанного размера.

5.2. **Транскрипция слов.** В английском языке есть слова иностранного происхождения, произношение и написание которых было адаптировано. Напишите программу, которая анализирует некоторый текст и корректирует окончания слов, например:

— `ize` меняет на `ise`

— `or` меняет на `our`

5.3. **Комментарии.** В файле хранится некоторый текст. Он разбит на параграфы, которые разделены пустыми строками. Напишите программу, которая считывает текст, а затем выводит его, игнорируя фрагменты в фигурных скобках. Следует обеспечить вывод предупреждений в следующих случаях:

— внутри фигурных скобок обнаружена еще одна скобка;

— обнаружена закрывающая скобка без соответствующей ей открывающей скобки;

— в параграфе отсутствует парная закрывающая скобка.

В конце каждого параграфа необходимо выводить статистику о том, какая доля текста (в процентах) приходится на комментарии в скобках (пробелы не учитываются). *Подсказка*: эта программа идеально подходит для проверки методов, предназначенных для работы с символами; при ее написании целесообразно использовать булевы переменные.

```
Пример вводимых данных:
Этот отрывок текста имеет такую же длину, что и следующий комментарий.
{Этот комментарий имеет такую же длину, что и предыдущий отрывок
текста. Можете проверить.} Можете проверить. { Для этого
Пример выхода программы:
Этот отрывок текста имеет такую же длину, что и следующий комментарий.
Можете проверить.
*** Нет закрывающей скобки

Комментарии составляют 50% текста
```

5.4. **Чувствительные лекарства.** Есть лекарства, которые восприимчивы к изменениям температуры и портятся, если в течение суток она изменяется более чем на 30°C. Температура измеряется и фиксируется каждые два часа. Напишите программу, которую могли бы использовать работники аптеки, чтобы определить, нужно ли помещать лекарства в специальные камеры.

5.5. **Управляемые аппараты.** Для управления некоторыми устройствами требуется вводить определенные последовательности чисел. Если последовательность

состоит из одинаковых чисел, во время программирования можно облегчить свою работу. Напишите программу, которая считывает применяемые последовательности и выводит их в виде n*m, где m — вводимый элемент, а n — число, указывающее, сколько раз должен повторяться элемент m. Эти инструкции выводятся в новой строке и заключаются в скобки. Следует отметить, что последовательность может состоять из одного числа. Вот один из вариантов вводимых и выходных данных программы:

```
20 20 20 20 20 20 20 20 20 20
(10*20)
50 50 50 50 50
(5*50)
30
(1*30)
 0
(1*0)
```

5.6. **Кролики!** Одна любительница домашних животных обожает кроликов. Исходя из площади, необходимой для проживания одного кролика, она подсчитала, что в ее новом доме хватит места для 500 любимцев. Но кролики отличаются плодовитостью, и каждая пара взрослых особей, достигших 3-х месяцев, через каждые три месяца производит на свет в среднем по два кролика. Напишите программу, с помощью которой ее бедный муж смог бы ей доказать, что через несколько месяцев в доме не останется места даже для хозяев. Определите, сколько времени потребуется для перенаселения дома, и выведите таблицу, где бы отдельно указывалось количество взрослых особей, количество "несовершеннолетних" кроликов и общее поголовье.

5.7. **Вновь Фибоначчи.** В задаче 3.16 требовалось вывести последовательности Фибоначчи. Измените программу так, чтобы выводилось только каждое третье значение. Какое свойство характерно для этих значений?

5.8. **Содержание.** Напишите программу, формирующую страницу с оглавлением для книги с указанием номеров страниц. Данные должны быть считаны из файла, выходные данные программы необходимо направить на принтер. Для примера возьмем следующие входные данные:

```
(Первые шаги (Компьютер, 5) (Решение проблем (Определение, 10)
(Общие черты, 15) (Алгоритмы, 20)) (Программы и процедуры, 25))
(Типы и циклы (Типы (Целые числа, 30) (Символы, 36)
(Булевы переменные, 43)) (Циклы (Циклы, управляемые счетчиком, 49)
(Циклы, управляемые условием, 52)))
```

В скобках приведены заголовки глав, разделов и подразделов, а числа, которые за ними следуют, являются номерами страниц. Номер страницы указывается, если структура не имеет подструктур. В результате обработки приведенных выше данных мы получим следующее:

```
Содержание
1. Первые шаги
   1.1. Компьютер                5
   1.2. Решение проблем
     1.2.1. Определение          10
     1.2.2. Общие черты          15
     1.2.3. Алгоритмы            20
   1.3. Программы и процедуры    25
```

2. Типы и циклы
2.1. Типы
2.1.1. Целые числа 30
2.1.2. Символы 36
2.1.3. Булевы переменные 43
2.2. Циклы
2.2.1. Циклы, управляемые счетчиком 49
2.2.2. Циклы, управляемые условием 52

5.9. **Таможенный сбор.** В Саванне специальным налогом облагаются все ввозимые иномарки. В настоящий момент с каждой машины взимается сумма, зависящая от ее стоимости. По этому критерию машины делятся на три категории:

Цена	Сумма налога
< G25 000	10%
от G26 000 до G100 000	G5000
> G100 000	15%

Напишите программу, которая выводит таблицу, состоящую их трех колонок, где указывается стоимость машины при ее покупке, сумма налога и сумма, которую придется заплатить гражданину Саванны за возможность ездить на понравившейся иномарке.

5.10. **Почтовые штампы.** Почтовое управление Саванны решило автоматизировать процесс отправки почты и для проставления штампов использовать принтер. Максимальная сумма, которую можно проставить на штампе, равна `G99.99`, а его дизайн очень прост:

```
--------------
|            |
| Саванна    |
| G14.30     |
| Авиапочтой |
|            |
--------------
```

Саванна разбита на три почтовые зоны. Отправка письма весом в 10 г стоит соответственно:

а) 50с

б) 90с

в) G1.10

При доставке авиапочтой стоимость удваивается. Напишите программу, которая предлагает пользователю указать вес своего письма, зону доставки и способ доставки — авиа или обычный. Приняв данные, программа должна вывести соответствующий штамп. Используйте класс для штампов и начните программу с создания трех объектов для различных зон.

5.11. **Курсы обмена.** Усовершенствуйте программу из примера 5.5 так, чтобы курсы можно было обновлять каждый день перед началом торговли.

5.12. **Номера (в отеле) с ограничениями.** Вернемся к примеру 4.12. Переделайте программу так, чтобы в ней использовались:

а) циклы `while`;

б) определенные пользователем объекты-исключения.

Рассмотрите достоинства различных версий программы, учитывая простоту программирования, эффективность применения конструкций и объем программы.

ГЛАВА 6



Массивы и таблицы

6.1. Простые массивы

Мы начали осознавать, что нам необходимо обеспечить возможность хранения и манипулирования в программе множествами значений. При относительно небольшом количестве значений можно использовать простые переменные, но давайте рассмотрим следующий пример.

Предположим, что в результате выполнения каких-то операций мы получили несколько сотен результирующих значений, относящихся к диапазону 0—19. Необходимо определить, с какой частотой встречается каждое значение. Мы могли бы установить 20 счетчиков, по одному для каждого результата. При вводе очередного результата значение соответствующего счетчика будет увеличиваться на единицу. Программа получится громоздкой, если мы введем 20 различных имен для счетчиков, а затем будем использовать большой оператор `switch` каждый раз, когда потребуется обновить значение какого-нибудь из счетчиков. Таким образом, целесообразно прибегнуть к концепции **i-той переменной**, согласно которой мы считываем введенное значение (допустим, `i`), обращаемся к счетчику `counteri` и обновляем его значение. В языках программирования для реализации описанной функции предусмотрены **массивы**.

Описание массива

Массив представляет собой заключенный в границы набор элементов одного типа, каждый из которых может быть выбран путем индексирования с использованием целых чисел от 0 до указанного при создании массива предела (значение которого уже не входит в диапазон). Ниже показано, как следует объявлять массив.

Объявление массива
`type arrayid [ ] = new type [limit];` `type arrayid [ ] = {values};`
Первый оператор создает переменную массива, именуемую `arrayid`, которая ссылается на последовательность переменных — элементы массива, проиндексированные целыми числами от `0` до `limit - 1`. `type` — это любой примитивный тип, класс или другой массив. Второй оператор выполняет то же самое, а также инициализирует каждый элемент массива, используя заданные значения. Длина массива равна количеству перечисленных элементов минус 1.

После объявления массивы могут содержать элементы любого типа или класса, но индексом, а следовательно, и длиной массива, всегда должно быть целое число. Следующие примеры относятся к первому оператору.

```
int frequencies [] = new int [20];
Times periods [] = new Times [8];
String countryCode [] = new String [175];
```

Элементами массива `frequencies` являются 20 целых чисел, пронумерованных от 0 до 19. Массив `periods` содержит 8 объектов класса `Times`, которые пронумерованы от 0 до 7. В массиве `countryCode` насчитывается 175 строк (объектов класса `String`) с номерами (индексами) от 0 до 174. Часто, хотя и не всегда, именами массивов являются существительные во множественном числе: frequencies — частоты, periods — периоды.

Второй оператор служит для создания массива исходных значений. Размер массива вычисляется исходя из количества значений, например:

```
char vowels [] = {'a', 'e', 'i', 'o', 'u'};
```

Массив `vowels` (что в переводе означает *гласные*) содержит пять символов, пронумерованных от 0 до 4.

Модель массива

В Java модель массива отражает модель его объекта, так что мы можем начертить диаграммы, чтобы показать, что означает каждое из приведенных выше объявлений. Обозначения для массива даны на рис. 6.1. Отличительным признаком переменной массива служат квадратные скобки. Стрелка, указывающая на массив, направлена вниз. Это означает, что могут быть перечислены как индексы, так и значения массива. Чтобы не перечислять на рисунках все элементы и индексы, мы заменяем часть блоков "троеточиями".

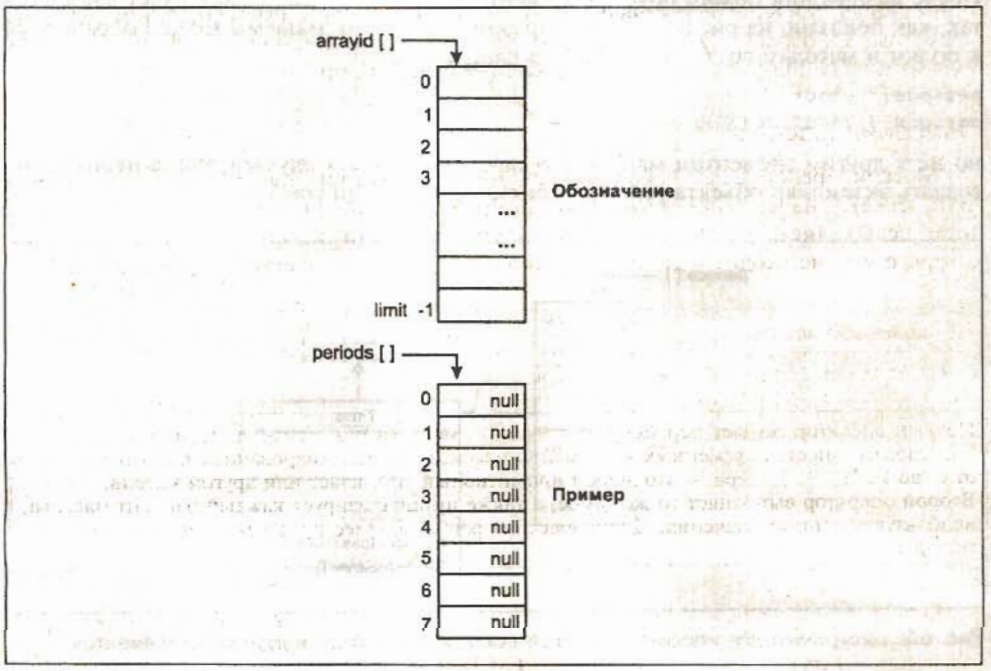


Рис. 6.1. Принятые для массива обозначения, применяемые в схемах

При объявлении массива в памяти выделяется пространство для каждого элемента, и если это элементы примитивного типа, то для них выполняется соответствующая инициализация. В этом случае целым числам присваивается значение 0. В следующем объявлении показывается, что элементами массива `periods` будут объекты. Они еще не являются объектами, а представляют собой объектные переменные. Каждый элемент массива `periods` с помощью оператора `new` должен стать объектом. По умолчанию они имеют значение `null`, как показано на диаграмме.

Доступ к элементам массива

Для получения доступа к элементу массива мы указываем имя переменной массива, а также индекс этого элемента в квадратных скобках. Например, мы можем использовать следующие операторы:

```
periods[3] = new Times (11,0);

for (int i=0; i < 20; i++) {
  frequencies[i] = (int) (Math.random*100);
}

countryCode[27] = "South Africa";

System.out.println(vowels[3]) ;
```

Запомните, что индексация массива всегда начинается с 0, поэтому в последнем примере на печать будет выведен четвертый по счету элемент (т.е. `'o'`, а не `'i'`).

А теперь давайте рассмотрим диаграмму, соответствующую массиву `periods`. После назначения нового объекта элементу `periods[3]` диаграмма будет выглядеть так, как показано на рис 6.2. Таким образом, на данном этапе мы можем обращаться к полям и методам созданного объекта класса `Times`:

```
periods[3].hour;
periods[3].addTime(30);
```

но не к другим элементам массива, за исключением тех случаев, когда необходимо создать экземпляр объекта или проверить, указывает ли ссылка на `null`.

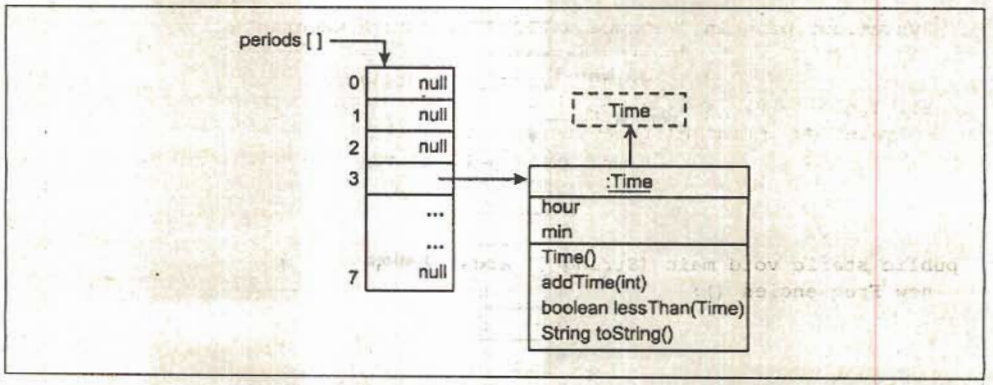


Рис. 6.2. Диаграмма для массива `periods` после инициализации одного из элементов

Пример 6.1 Определение частоты употребления значений

Задача. В наборе, состоящем из нескольких сотен значений из диапазона от 0 до 19, необходимо определить частоту употребления каждого значения.

Решение. В начале главы мы наметили вкратце, как решить эту задачу: создается массив и при вводе очередного числа соответствующий элемент массива увеличивается на единицу. Алгоритм настолько прост, что мы сразу перейдем к рассмотрению программы.

Программа. Для получения ста чисел, каждое из которых входит в диапазон 0—19, программа использует генератор случайных чисел. Обратите внимание на следующую строку:

```
scoreFregs [score]++;
```

Именно она задает увеличение соответствующего элемента массива.

```
import javagently.*;

class Frequencies {
  /* Программа "Подсчет частоты".   Автор - Дж. М. Бишоп, декабрь 1996 г.
   * ----------------------------   Java 1.1
   *                                обновление - май 2000 г.
   * Определяет частоту употребления каждого значения в
   * наборе из нескольких сотен значений от 0 до 19.
   * Демонстрирует работу с простым массивом.
   */

  Frequencies () {

    int maxscore = 20;
    int scoreFreqs [] = new int [maxscore];
    int score;

    for (int i=0; i < 100; i++) {
        score = (int) (Math.random() * maxscore);
        scoreFreqs[score] ++;
    }

    System.out.println("Таблица частоты вхождений чисел\n"+
                       "================================\n\n"+
                       "  Число  встретилось (раз)");
    for (int i = 0; i<maxscore; i++) {
      System.out.println(Stream.format(i, 6) +
                         Stream.format(scoreFreqs[i], 6));
    }
  }

  public static void main (String [] args) {
    new Frequencies ();
  }
}
```

Тестирование. Результат выполнения программы может быть следующим:

```
Таблица частоты вхождений чисел
================================
Число встретилось (раз)
  0       2
  1       6
  2       7
  3       2
  4       4
  5       5
  6       2
  7       3
  8       3
  9       9
  10      8
  11      5
  12      7
  13      5
  14      7
  15      5
  16      6
  17      4
  18      6
  19      4
```

В главе 10 на основе этой таблицы мы создадим гистограмму.

Характеристики массивов

Типы элементов массива

Массивы могут состоять из элементов любого типа или класса — от элементов простых типов до объектов и даже целых массивов. В последнем случае речь уже идет о многомерных массивах, которые подробно рассматриваются в разделе 6.2.

Размер

Размер массива ограничивается только объемом доступной памяти, которой обычно достаточно для подавляющего большинства приложений. Этот параметр **фиксируется** во время создания массива и впоследствии уже не может быть изменен.

Замедленное назначение длины массива

Объявление массива может быть выполнено в два этапа: на первом этапе объявляется имя массива и создается ссылка, а на втором — выделяется область памяти. Эта опция очень полезна в тех случаях, когда размер массива должен быть введен как элемент данных. Так, мы можем сначала объявить:

```
int A [];
```

а затем, когда будет введено или вычислено значение `n`, указать:

```
A = new int [n];
```

что завершит создание массива (далее уже будет следовать инициализация его элементов).

Оператор для массива

Единственным оператором, который можно применить ко всему массиву, является оператор присвоения. В результате назначения одного массива другому не создается копия массива. При выполнении этой операции копируются ссылки, вследствие чего оба массива обращаются к одной и той же области памяти, а изменения, произведенные в одном массиве, отражаются на другом. Предположим, у нас объявлен массив A (см. выше). После выполнения последовательности операторов

```
int B [];
B = A;
B[3] = 81;
```

массивы B и A будут ссылаться на одни и те же элементы, как показано на рис. 6.3, а обращения A[3] и B[3] — на одну и ту же область памяти, в которой хранится значение 81.

Копирование фактических значений массивов может быть выполнено тем же способом, что и копирование объектов, т.е. методом **клонирования**. Этой теме посвящен раздел 8.3. Если же у нас возникнет необходимость скопировать массив на данном этапе, мы можем создать новый массив и, используя цикл, скопировать поочередно все элементы.

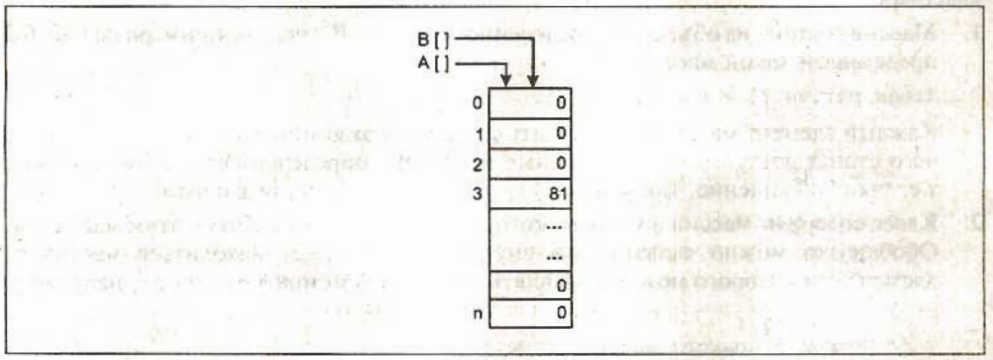


Рис. 6.3. Две переменные обращаются к одному и тому же массиву

Доступ к элементам

В Java строго соблюдается контроль за предоставлением доступа только к существующим элементам массива. Каждый раз при обращении к массиву указанный индекс элемента проверяется на соответствие границам массива, которые определяются при его объявлении. Если индекс не входит в обозначенный диапазон, генерируется объект-исключение `ArrayIndexOutOfBoundsException`. Исключительная ситуация может быть "перехвачена" и обработана (выполняются определенные пользователем методы), иначе программа останавливается. Например, если взять выполненные выше объявления, обращение к `frequencies[100]` и `vowels[5]` приведет к возникновению ошибочной ситуации.

Длина

Каждый массив имеет характеристику, которая позволяет работать с числом, представляющим длину массива. Эта характеристика называется `length`. Так, введя

```
frequencies.length
```

мы получим число 20. Другими словами, `length` возвращает длину массива, указанную при объявлении. Поскольку `length` — это не метод, а характеристика, скобки отсутствуют.

Параметры

Массивы могут передаваться методам в качестве параметров. Очень удобным свойством Java является то, что формальный параметр в методе не должен указывать длину массива, который будет передан этому методу. Метод может принимать массивы корректного типа любой длины и обрабатывать их, используя описанную выше характеристику `length`. В качестве примера можно привести метод `main`, который объявляет*:

```
main (String [] args)
```

При наличии каких-либо аргументов `main` может вывести их, используя цикл `for`:

```
for (int i = 0; i < args.length; i++)
  System.out.println(args[i]);
```

Взаимодействие между массивами и классами

Существует по крайней мере три вида взаимодействия между массивами и классами.

1. Массив состоит из объектов определенного класса. В качестве примера на рис. 6.2 представлен такой массив:

   ```
   Times period [] = new Times [8];
   ```

   Каждый элемент массива может быть создан как экземпляр объекта `Times`, после чего станут доступными переменные и методы, определенные в этом объекте, т.е. такое обращение, как `period[3].hour`, будет действительным.

2. Класс содержит массив и методы, которые обеспечивают работу с этим массивом. Обобщенно можно сказать, что внутри класса может находиться массив `A`, элементами которого можно управлять с помощью методов `get` и `set`, например:

   ```
   class IntArray {
     // Пример закрытого массива с методами get и set.
     private int A [];

       IntArray (int n) {
         A = new int [n];
       }

       int get (int i) {
         return A[i];

       void set (int i, int x) {
         A[i] = x;
       }

       public String toString () {
         String s = "";
         for (int i = 0; i<A.length; i++) {
           s += "  " + A[i];
   ```

* Существует альтернативный синтаксис объявления массива, где скобки ставятся перед идентификатором массива. Он часто использовался в методе `main` ранних программ и идентичен случаю, когда скобки стоят после идентификатора массива.

```
    }
    return s;
  }

}
```

Массив A отмечен как закрытый, потому что в этом классе доступ к массиву можно получить только посредством методов get и set. Диаграмма моделирования для класса IntArray представлена на рис. 6.4.



Рис. 6.4. Класс IntArray

Объекты класса IntArray могут быть объявлены, и каждый из них получит собственный экземпляр массива A. Например, возможен такой блок операторов:

```
IntArray marks = new IntArray(50);
// Ожидается использование массива с 50-ю элементами.
int m;

for (int i = 0; i < 50: i++) {
  m = (int) (Math.random()*100);
  marks.set (i, m);
}
System.out.println(marks);
```

Диаграмма моделирования для этого случая представлена на рис. 6.5. На этой диаграмме массив A размещен в конце объекта marks. Ему соответствует вертикальная секция, состоящая из ячеек. Таким образом демонстрируется то, что массив A является закрытым.

3. Класс содержит методы, которым передаются массивы в качестве параметров (рис. 6.6). Класс может выглядеть следующим образом:

```
class ArrayUtils {

  static int max (int A []) {
    // Находит максимальный элемент массива – A[i] - и возвращает его.
    return A[i];
  }

  static void sort (int A[]) {
    // Сортирует элементы массива A.
  }
}
```

Содержимое методов `max` и `sort` еще не определено. Обратите внимание на то, что методы являются методами класса и что в данном случае мы определили класс утилит. Для его использования мы должны указать следующие операторы:

```
int frequencies [] = new int [20];
int max = ArrayUtils.max(frequencies);
ArrayUtils.sort(frequencies);
```

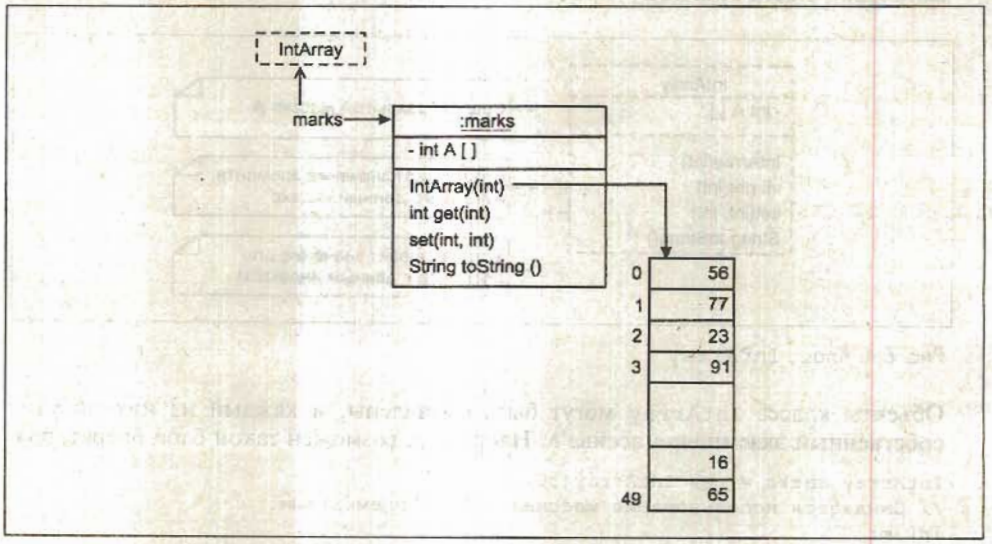


Рис. 6.5. Класс, содержащий массив

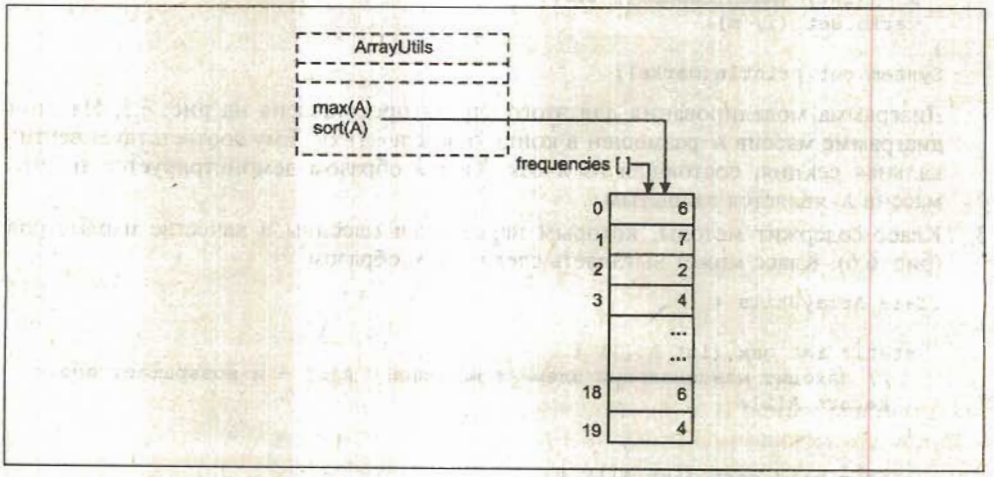


Рис. 6.6. Класс с методами, которые работают с массивами, переданными им в качестве параметров (в данном случае для обоих методов указан один массив A)

На рис. 6.6. показано, что существует только один массив — `frequencies`. Каждый метод класса `ArrayUtils` имеет формальный параметр, который является

массивом. Когда вызывается метод, например `max`, содержимое массива `frequencies` используется в качестве массива `A`, и все действия, выполняемые по отношению к `A`, теперь будут относиться и к массиву `frequencies`.

Далее мы реализуем на практике описанные способы взаимодействия.

Массивы как абстракция

Мы задержались с представлением массивов читателю для того, чтобы акцентировать внимание на операциях управления при обработке данных. Вы не раз столкнетесь с решением задачи, которое состоит в применении массива в тех случаях, когда в этом нет особой необходимости. Это не ошибка, а один из вариантов решения.

Рассмотрим следующую задачу: необходимо ввести 20 значений и найти среди них наибольшее и наименьшее. Существует два способа решения.

- Введите 20 значений и просмотрите их, оставляя в памяти наибольшее и наименьшее, как в примере 4.5.
- Введите 20 значений в массив, после чего дважды выполните анализ его содержимого, чтобы найти наибольшее и наименьшее значения.

Первый способ более привлекателен и не сложнее второго. Но если учесть, что у нас уже есть в запасе определенные методы, то эффективней окажется второй способ. Различие продемонстрировано на рис. 6.7.

Рассматриваемая концепция называется **разделением обязанностей (separation of concerns)**. Предположим, что мы уже имеем методы для определения минимального и максимального значений в данном массиве (как мы уже установили выше для класса `ArrayUtils`). Согласно первому способу все операции скомбинированы в одном процессе, а согласно второму — три различные операции не связаны друг с другом по времени выполнения и идентифицированы как три разных метода. Второй способ эффективен при работе с программами большого объема, поскольку в этом случае легче читать программу и выполнять отладку.

Изложенные идеи воплощены в примере 6.2. Хотя острой необходимости в применении массива не было, мы его использовали, за счет чего программа стала гораздо понятнее. А вот в примере 6.1 массив действительно необходим.

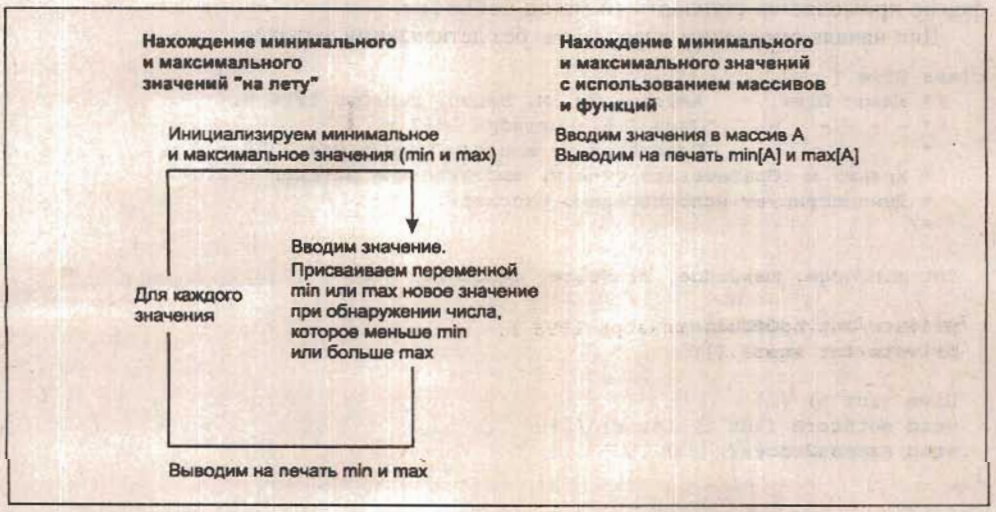


Рис. 6.7. Два алгоритма нахождения минимального и максимального значений

Пример 6.2. Соревнования по прыжкам в воду

Задача. Соревнования по прыжкам в воду судят представители нескольких стран. Чтобы никто не пострадал от предвзятого судейства (например от того, что судья ставит спортсмену своей страны явно завышенную оценку), для каждого прыжка определяется средний балл, причем самый высокий и наиболее низкий результаты не учитываются. Необходимо автоматизировать эти вычисления.

Алгоритм. Существует два способа решения данной задачи. Во-первых, три необходимых значения (сумму, наибольшую и наименьшую оценки) можно вычислить при вводе оценок, а во-вторых, мы можем ввести значения (оценки) в массив и затем их обработать. Если мы выберем второй способ, то снова окажемся перед выбором — вычислять все три значения одновременно в одном цикле или сформировать три различных цикла (возможно, в трех различных методах). Поскольку эти вычисления очень просты, остановимся на первом варианте и выполним все операции в одном цикле.

Конструкция класса. Нам предстоит найти ответы на ряд вопросов. Что взять за основу при разработке класса? Что должна представлять собой концепция, которую стоило бы оформить в виде класса? Есть несколько вариантов.

- Класс `Judge` (в переводе — *судья*) с оценками для каждого прыжка. Если выполнено три прыжка, этот класс должен содержать массив с тремя целыми числами (оценками).
- Класс `Dive` (в переводе — *прыжок*) с оценками, выставленными каждым судьей. Если соревнования судят восемь человек, массив будет состоять из восьми элементов.

По ходу соревнований имеет смысл сохранять результаты каждого прыжка. Мы разовьем эту тему в практикуме, который находится в конце настоящей главы.

При конструировании классов заслуживает рассмотрения еще один момент: мы должны разделить ввод данных и вывод результатов. Класс, моделирующий физический объект, не должен быть привязан к потокам ввода и вывода. Рекомендуем помещать операции ввода/вывода в отдельную часть программы и использовать методы `get` и `set` для изменения данных. Для вывода данных на печать в корректной форме применяется стандартный метод `toString`.

Для начала определим класс `Dive` без детализации методов.

```
class Dive {
   /* Класс Dive.      Автор - Дж. М. Бишоп, декабрь 1996 г.
    * - - - - - -      Java 1.1 - октябрь 1997 г.
    *                  Переделан из класса Judge в мае 2000 г.
    * Хранит и обрабатывает оценки, выставленные за один прыжок.
    * Демонстрирует использование массивов.
    */

  int minJudge, maxJudge, minScore, maxScore, sum;

  private int noOfJudges;
  private int score [];

  Dive (int n) {…}
  void setScore (int i, int s) {…}
  void assessScores() {…}
}
```

Класс Dive имеет конструктор, отвечающий за создание массива корректного размера, количество элементов которого равно количеству судей. Это число (переменная n) передается конструктору в качестве параметра. Следует отметить, что сам массив недоступен для открытого использования. Поэтому ввод оценки осуществляется методом setScore. После ввода всех оценок массив обрабатывается методом assessScores, вследствие чего пять указанных переменных получают свои значения. Они выводятся на печать следующим образом:

```
d.println("Оценки: " + dive.minScore + " выставленная судьей "
      + dive.minJudge + " и " + dive.maxScore +
      " выставленная судьей " + dive.maxJudge + " исключаются.");
```

В соответствии с особенностями взаимодействия между классом Dive и массивом score они относятся ко второй категории из числа описанных выше. Массив оценок содержится в классе, а мы имеем два метода — setScore и assessScore для присваивания значений элементам массива и для их обработки.

Программа. Ниже приведен полный код класса Dive.

```
import java.io.*;

class Dive {
   /* Класс Dive          Автор - Дж. М. Бишоп, декабрь 1996 г.
    * - - - - - -         Java 1.1 - октябрь 1997 г.
    * Хранит и обрабатывает оценки, выставленные за один прыжок.
    * Демонстрирует использование массивов.
    */

    int minJudge, maxJudge, minScore, maxScore, sum;

    private int noOfJudges;
    private int score [];

    Dive (int n) {
      noOfJudges = n;
      score = new int [noOfJudges];
    }

    void setScore (int i, int s) {
      score [i] = s;
    }

    void assessScores() {
        minScore = 10;
        maxScore = 0;
        sum = 0;
        for (int i = 0; i < noOfJudges; i++) {
            sum += score[i];
            if (score[i] <= minScore) {
                minScore = score[i];
                minJudge = i;
            }
            if (score[i] > maxScore) {
                maxScore = score[i];
                maxJudge = i;
            }
        }
        minJudge++;
```

```
        maxJudge++;
    }

}
```

Обратите внимание на то, что мы не можем использовать методы `Math.min` и `Math.max` в методе `assessScores`, поскольку при обнаружении минимальной и максимальной оценок должны зафиксировать индекс элемента массива, а также его значение, чтобы впоследствии вывести на печать номера судей, уличенных в предвзятости.

Рассмотрим теперь текст программы. Обратите внимание на то, что применение метода `format` класса `Stream` гарантирует получение в результате вычислений такого количества цифр в дробной части, сколько необходимо для выставления оценок в данном виде соревнований. Мы воспользовались `Display`, чтобы обеспечить удобный ввод оценок.

```
import java.io.*;
import javagently.*;

class DivingCompetition {

  /* Программа
   * "Соревнования по прыжкам в воду"   Автор - Дж. М. Бишоп, декабрь 1996 г.
   * ------------------                 Java 1.1 - октябрь 1997 г.
   *                                    Обновление - май 2000 г.
   * Класс Dive используется для фиксирования корректных
   * оценок, выставленных за прыжок.
   * Демонстрирует использование вызывающих методов класса,
   * типизированных методов и массивов
   * в отдельном классе. */

  Display d = new Display ("Соревнования по прыжкам в воду ");
  int noOfDives = 3;
  int noOfJudges = 8;

  DivingCompetition () throws IOException {

    headings ();

    double result, total = 0;
    //В объекте Dive фиксируются 8 оценок.
    //Этот объект используется для каждого прыжка.

    Dive dive = new Dive (noOfJudges);
    int score;

    for (int i=0; i < noOfDives; i++) {
      d.println("Прыжок №: "+(i+1));
      d.println("Введите оценки, \"выставленные\" судьями");
      d.ready("Нажмите кнопку \"Готово\" для\n дальнейшей работы программы");
      for (int j = 0; j < noOfJudges; j++) {
        score = d.getInt("Судья "+(j+1));
        dive.setScore(j, score);
      }
      dive.assessScores();
```

```
        result = (double) (dive.sum
               - dive.minScore - dive.maxScore)
               / (double) (noOfJudges - 2);
        total += result;
    d.println("Исключаются оценки: ");
    d.println(dive.minScore + " - выставлена судьей " + dive.minJudge);
    d.println(dive.maxScore + " - выставлена судьей " + dive.maxJudge);
        d.println("Средний балл: " + Text.writeDouble(result,5,1));
      }
    d.println("Средний балл по трем прыжкам: "+
          Text.writeDouble(total/noOfDives,5,3));
    }

    void headings () {
      d.println ("Информационное табло\n" +
                 "====================\n");

      d.println("По каждому из " + noOfDives +
        " прыжков оценки выставляют \n"+noOfJudges+" судей.");
      for (int i = 1; i<=noOfJudges; i++)
        d.prompt ("Судья "+i,0);
    }

    public static void main (String [] args) throws IOException {
      new DivingCompetition ();
    }

}
```

Тестирование. Результат выполнения программы представлен на рис. 6.8.

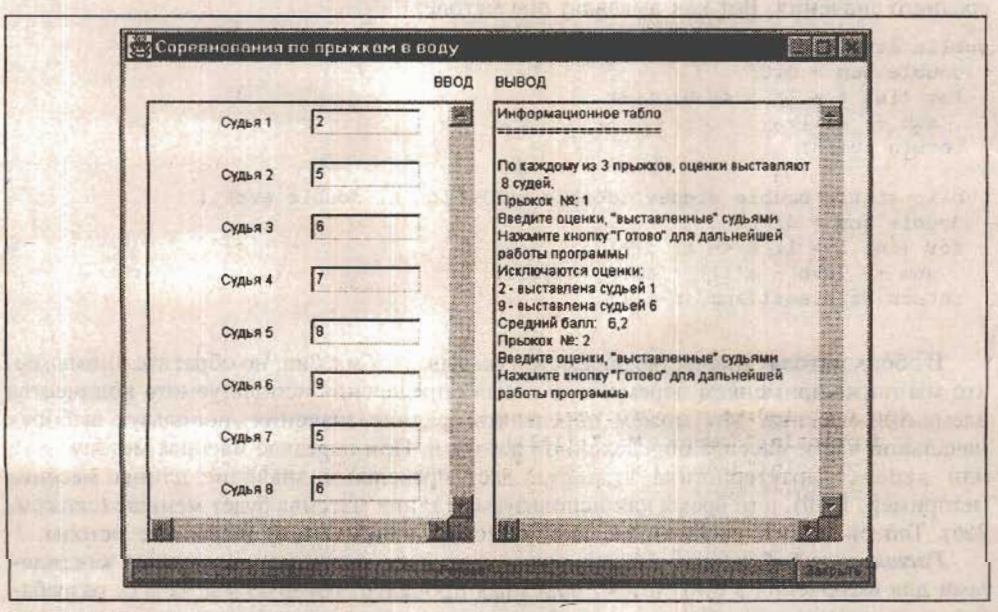


Рис. 6.8. Интерфейс класса `Display` для программы "Соревнования по прыжкам в воду"

Пример 6.3. Статистический анализ

Задача. Под статистическими операциями с элементами массива следует понимать подсчет *среднего значения* (*mean* — *xm*) и стандартного отклонения. Надлежит разработать методы для этих операций и поместить их в библиотеку, с тем чтобы обеспечить к ним легкий доступ (см. раздел 4.3).

Решение. Если мы имеем некоторое множество значений x_i, мы можем проанализировать их и найти среднее значение *xm* и стандартное отклонение *s*, показывающее, на сколько каждый показатель может отклониться от средней величины. Другими словами, стандартное отклонение позволяет определить разброс в значениях.

Теория. Среднее значение x_i, где $i = 1, 2, ..., n$, определяется по формуле:

$$xm = \frac{1}{n}\sum_{i=1}^{n} x_i$$

а формула для вычисления стандартного отклонения имеет следующий вид:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - xm)^2}{n-1}}$$

Принимая во внимание возможные ошибки и разброс значений, мы можем сказать, что результаты множества измерений находятся в рамках определенного стандартного отклонения. Во многих случаях действительный результат находится в диапазоне $xm \pm 2s$, т.е. среднее значение плюс/минус двойное стандартное отклонение.

Конструкция метода. Написание метода для вычисления среднего значения не составляет большого труда. Для этого мы будем использовать цикл `for`. Поскольку в формуле для определения стандартного отклонения используется среднее значение, этот метод должен быть определен после метода, предназначенного для нахождения среднего значения. Вот как выглядят оба метода:

```
public static double mean (double a [], int n) {
  double sum = 0.0;
  for (int i = 1; i <= n; i++)
    sum += a[i];
  return sum/n;
}
public static double stddev (double a [], int n, double ave) {
  double sum = 0;
  for (int i = 1; i <= n; i++)
    sum += (ave - a[i]) * (ave - a[i]);
  return Math.sqrt(sum/(n-1));
}
```

В обоих методах в качестве параметра указывается массив, но обратите внимание, что мы также применяем переменную `n` для определения используемого количества элементов массива. Мы можем определять средние значения, используя выборку начальной части массива определенной длины `n`. При передаче массива методу `mean` или `stddev` характеристика `a.length` даст предельное значение длины массива (например, 1000), в то время как используемая длина массива будет меньше (скажем, 256). Теперь давайте рассмотрим, как поместить в библиотеку указанные методы.

Расширение библиотеки. *Методы* `mean` *и* `stddev` *являются отличными кандидатами* для включения в библиотеку полезных процедур, которую мы начали разрабатывать в примере 4.9. Структура библиотеки такова: методы находятся в классах,

которые, в свою очередь, содержатся в пакете. Мы уже сталкивались с пакетом java.io, а также с очень полезным пакетом javagently, который был специально разработан для данной книги. В пакете javagently предлагаются следующие классы: Stream, Display и Graph.

Методы мы добавим не в пакет javagently, а в пакет myutilities, где уже находится класс Filer. При работе с Java используется следующий способ создания методов и помещения их в новый класс нового пакета:

```
package myutilities;

public class Stats {

  public static double mean (double a [], int n) {
    ...как и прежде
  }

  public static double stddev (double a [], int n, double ave) {
    ... как и прежде
  }
}
```

Указанный выше класс помещается в файл Stats.java. Компилирование этого класса должно быть выполнено в каталоге myutilities. Путь к классу расширяется так, чтобы включать каталог, содержащий пакет myutilities (рис. 4.2). Предварительно туда необходимо поместить файл Stats.java. В результате любая программа, которая захочет использовать эти методы, должна будет импортировать пакет myutilities.* и вызвать нужный метод следующим образом:

```
ave = Stats.mean (myreadings, number);
```

Обратите внимание на то, что класс и его методы объявлены открытыми, поэтому их можно "видеть" из других пакетов.

Пример 6.4. Время службы лампочек

Задача. В Саванне есть фирма, производящая лампочки. Через произвольные промежутки времени или после выпуска определенного объема продукции проверяется срок службы лампочек. Для надлежащего контроля качества требуются регулярные отчеты о средней продолжительности пригодности лампочки и стандартного отклонения от этого значения. Указанные величины измеряются в часах.

Алгоритм. При создании массива мы воспользуемся вторым способом из числа тех, которые рассматривались ранее: объявим большой массив, элементы которого будем использовать в произвольном количестве в зависимости от объема поступивших данных. При этом будет проводиться контроль за тем, чтобы количество вводимых элементов не превышало определенного максимального числа.

```
double hourreadings [] = new double[max];
BufferedReader in = Text.open(System.in);
int n = 0;

try {
  for (n=1; n<max; n++)
    hourreadings[n] = Text.readInt(in);
  System.out.println("Программа обрабатывает только "+(max - 1)+" замеров");
```

```
}
catch (EOFException e) { }

n--;
System.out.println("Данные приняты для обработки");
System.out.println("Было "+n+" замеров");
```

Предполагается, что методы класса `Stats` должны обрабатывать значения от `i = 1` до `i=n`. Поскольку формула для вычисления стандартного отклонения включает деление на `n-1`, мы добавим в программу блок, проверяющий, не равно ли `n` единице, чтобы тем самым избежать ошибки выполнения (деления на 0).

Программа

```
import java.io.*;
import javagently.*;
import myutilities.*;

class LightBulbAnalysis {

  /* Программа "Анализ срока службы лампочек". Автор Дж. М. Бишоп, 1990, 1998 г.
   * ------------------------------          Обновление - май 2000 г.
   *
   * Подсчитывает средний срок службы электролампочек.
   * Применяет методы mean и stddev, определенные в классе
   * Stats пакета myutilities.
   *
   * Демонстрирует использование массива и операций ввода/вывода.
   */

  LightBulbAnalysis () throws IOException {
    int max = 101;
    Display display = new Display("Анализ срока службы лампочек");

    display.println ("***** Анализ срока службы лампочек *****\n");
    display.println("Кол-во вводимых данных должно быть < "+max);
    display.println("После каждого введенного замера \n нажмите кнопку
                    \"Готово\"");
    display.println("После ввода всех замеров введите \"да\"");
    display.prompt("Часы",0);
    display.prompt("Стоп", "нет");

    double hourReadings [] = new double[max];
    int n = 0;
    boolean stop = false;

    for (n=1; n<max & !stop; n++) {
      display.ready("\nВведите число");
      hourReadings[n] = display.getDouble("Часы");
      display.print(Stream.format(hourReadings[n],6,1)+"  ");
      stop = display.getString("Стоп").equalsIgnoreCase("да");
    }
    n--;
    display.println("\n Данные приняты для обработки ");
    display.println("Было "+n+" замеров ");
    if (n == 1) {
```

```
      System.out.println("Средний срок службы = "+
         Stream.format(Stats.mean(hourReadings,n),10,6)+
         " но вычисление стандартного отклонения не определено для  n=1");
    }
    else {
      double ave = Stats.mean(hourReadings,n);
      display.println("Средний срок службы = "+Stream.format(ave,6,2)+
                      " часов");
      display.println("Стандартное отклонение = " +
         Stream.format(
           Stats.stddev(hourReadings,n,ave),6,2) + " часа");
    }
  }

  public static void main (String [] args) throws IOException {
    new LightBulbAnalysis ();
  }
}
```

Тестирование. Количество введенных значений должно быть меньше указанного максимального количества. На рис. 6.9. показано, каким будет результат работы программы при указании шести замеров.

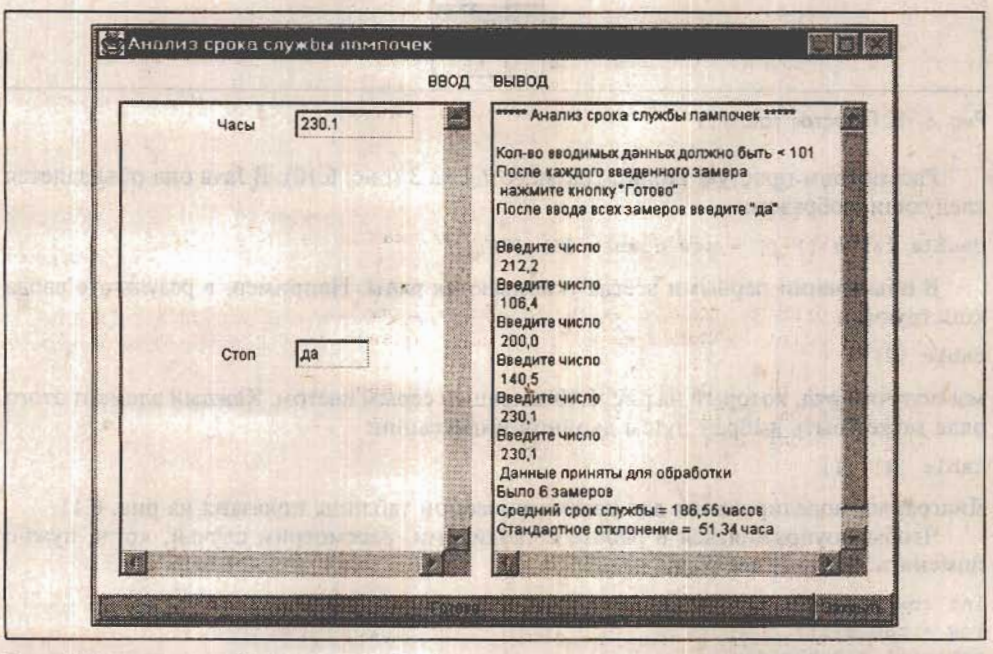


Рис. 6.9. Результат работы программы "Анализ срока службы лампочек"

Для проверки корректности ответов мы можем вычислить искомое значение вручную или с помощью калькулятора. Еще один способ — выполнение программы с использованием контрольных данных. Например, при вводе пяти значений 200 должно получиться среднее значение срока службы равное 200 и стандартное отклонение величиной 0.

6.2. Таблицы

Таблицами часто оперируют в компьютерных программах. Данные в таблице распределены по столбцам и строкам. В Java массив может содержать элементы любого типа, включая сами массивы, что позволяет создавать массивы, имеющие несколько измерений, которые известны как **многомерные массивы**. В большинстве случаев мы будем использовать массивы, которые имеют не более двух измерений их называют **матрицами** или **таблицами**.

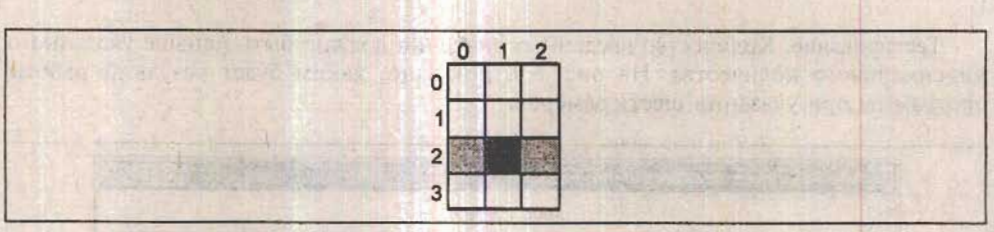


Рис. 6.10. Простая таблица

Рассмотрим простую таблицу размером 4 на 3 (рис. 6.10). В Java она объявляется следующим образом:

```
double table [] [] = new double [4] [3];
```

В объявлении первыми всегда указываются ряды. Например, в результате ввода конструкции

```
table [2]
```

мы получим ряд, который на рис. 6.10 закрашен серым цветом. Каждый элемент этого ряда может быть выбран путем двойной индексации:

```
table [2] [1]
```

Диаграмма моделирования для рассматриваемой таблицы показана на рис. 6.11.

Чтобы поупражняться в работе с таблицами, рассмотрим случай, когда нужно поменять местами два ряда:

```
int row [] = new int [2];
row = table[i];
table[i] = table[j];
table[j] = row;
```

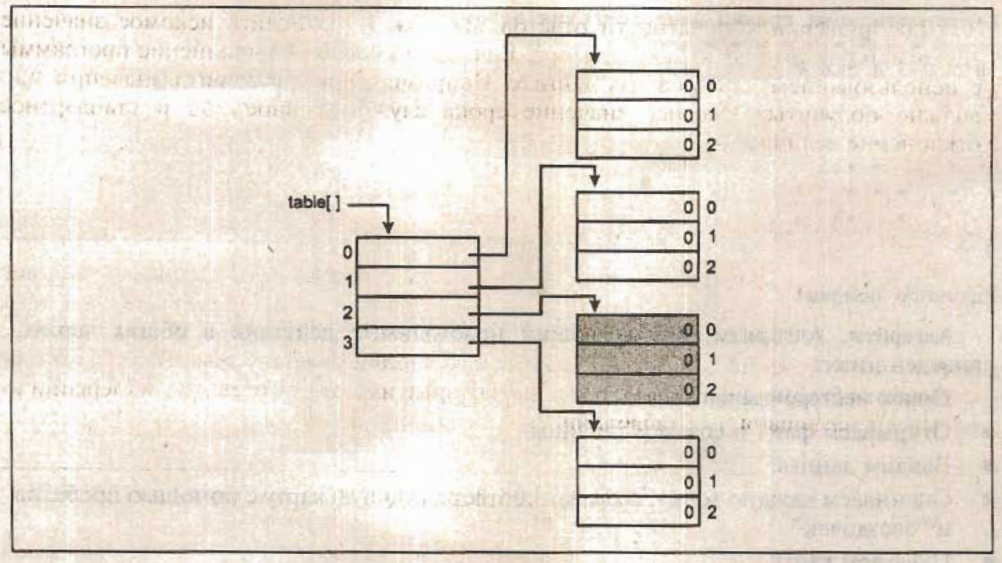


Рис. 6.11. Схема строения таблицы

Пример 6.5. Поиск месторождений золота

Задача. Золотодобывающая компания Саванны получила замеры инфракрасного излучения в определенном районе пустыни, где, как предполагается, может находиться месторождение золота. Эти данные должны помочь установить границы месторождения. Границы определяются на базе замеров, которые имеют большую величину, чем средний показатель, вычисленный на основе четырех точек вокруг проверяемой точки. Компания хотела бы иметь программу, которая определяет границы предполагаемого месторождения.

Решение. Карта замеров может рассматриваться по одному элементу за раз. Одновременно будет выводиться карта тех величин, которые превышают средний уровень инфракрасного излучения. Предположим, мы имеем массив размером 8×8 с такими данными.

```
21 21 22 30 40 21 34 45
21 22 23 30 45 21 37 40
22 23 24 45 46 47 38 39
22 23 24 35 46 47 38 38
23 24 25 36 46 48 37 36
23 24 25 37 39 48 36 35
23 24 25 25 26 25 26 25
23 25 26 27 28 29 30 31
```

Теперь мы можем вывести соответствующую карту:

```
***** Золотодобывающая компания Саванны *****
Если в открытом файле нет положительного результата,
используйте данные файла mali.dat
Файл sam.dat не существует.
Попытайтесь еще раз.
mali.dat
Мы найдем золото!
```

```
Карта возможных границ месторождения золота
= = = = = = = = = = = = = = = = = = = = = = = =
 0 1 2 3 4 5 6 7
0
1         *   *
2  *    * * *
3           * *
4  *    * * *
5         *   *
6
7
Удачного поиска!
```

Алгоритм. Алгоритм, описывающий необходимые действия в общих чертах, приведен ниже:

Поиск месторождений золота

- Открываем файл и создаем таблицы
- Вводим данные
- Оцениваем каждую точку, создавая соответствующую карту с помощью пробелов и "звездочек"
- Печатаем карту

Каждый из этих шагов может быть усовершенствован до уровня двойных циклов, сканирующих всю таблицу. Например, оценка точки может выполняться следующим образом: складываются значения соседних точек, расположенных от нее слева, справа, снизу и сверху, после чего полученная сумма делится на 4. Если значение замера в этой точке окажется больше средней величины для соседних четырех точек, мы отметим ее на карте звездочкой как часть месторождения. В результате у нас будет две таблицы: таблица замеров `data` и таблица символов `map`, соответствующая карте месторождения.

```
point = data[i] [j];
average = (data[i-1][j]+data[i+1][j]+data[i][j-1]+data[i][j+1])/4;
if (point > average) {
  map [i] [j] = cover;
}
```

Следует рассмотреть еще один вопрос: как быть с точками, расположенными по краю карты? Мы будем исходить из того, что у нас достаточно данных для обнаружения месторождения, поэтому мы проигнорируем точки, расположенные на границе карты. Вот почему циклы, оценивающие точки, будут работать на участках массивов, которые ограничены индексами от `1` до `rowmax-1` и от `1` до `colmax-1`.

Конструкция класса. В процессе решения задачи используются два набора данных, которые являются таблицами. Элементы одной таблицы имеют тип `double`, а другой — `char`. По этой причине нет необходимости определять отдельный класс, который бы что-нибудь моделировал и экземпляры которого создавались бы повторно. Однако стоит рассмотреть процедуры создания нескольких полезных методов.

Программа. Вот что представляет собой класс `GoldExsploration`:

```
import java.io.*;
import javagently.*;
import myutilities.*;

class GoldExploration {

  /* Класс GoldExploration.     Автор - Дж. М. Бишоп, январь 1997 г.
```

```
 *                          Java 1.1 октябрь 1997 г.
 * Преобразует вводимые геологические замеры в
 * символьную карту возможного месторождения золота.
 * Демонстрирует использование в классе двух различных
 * многомерных массивов.
 */

int rowMax = 8;
int colMax = 8;
char   map [][]  = new char[rowMax][colMax];
double data [][] = new double[rowMax][colMax];
char blank = ' ';
char cover = '*';

Stream fin;

GoldExploration () throws IOException{
  introduction();
  getData();
  assess();
  report();
}

public void getData() throws IOException {
  for (int i = 0; i < rowMax; i++)
    for (int j = 0; j < colMax; j++)
      data[i][j] = fin.readDouble();
}

public void assess() {
  double point, average;
  for (int i = 0; i < rowMax; i++) {
    for (int j = 0; j < colMax; j++) {
      map[i][j] = blank;
    }
  }
  for (int i = 1; i < rowMax-1; i++) {
    for (int j = 1; j < colMax-1; j++) {
      point = data[i][j];
      average = (data[i-1][j] + data[i+1][j] +
                 data[i][j-1] + data[i][j+1]) / 4;
      if (point > average)
          map[i][j] = cover;
      }
    }
}

public void report() {
  System.out.println("Карта возможных границ месторождения золота ");
  System.out.println("=========================================== ======");
  System.out.println();
  System.out.print("  ");
  for (int j = 0; j < colMax; j++) {
    System.out.print(j+" ");
  }
  System.out.println();
  for (int i = 0; i < rowMax; i++) {
```

```
      System.out.print(i+" ");
      for (int j = 0; j < colMax; j++)
        System.out.print(map[i][j]+" ");
      System.out.println();
    }
    System.out.println("Удачных вам поисков!");
  }

  void introduction () throws IOException {
    System.out.println("******* Золотодобывающая компания Саванны *****");
    System.out.println("Если открытый файл не дал положительного результата \n
                        используйте данные файла mali.dat"+
                        " ");
    fin = Filer.open("");
    System.out.println("Мы найдем золото!\n");
  }

  public static void main(String[] args) throws IOException {
    new GoldExploration ();
  }
}
```

Метод `introduction` использует класс открытия файлов `Filer`*, который разработан нами в примере 4.9 и хранится в пакете `myutilities`. Если файл не найден, загружается карта `mali.dat`, используемая по умолчанию, в существовании которой мы уверены.

Тестирование. Ранее уже был приведен вывод программы, которая обрабатывала файл, содержащий данные, которые позволяли составить карту месторождения. Мы решили дополнительно проверить программу и ввели одинаковые данные, вследствие чего получили следующее:

```
***** Золотодобывающая компания Саванны *****
Если открытый файл не дал положительного результата,
используйте данные файла mali.dat.
Файл sam.dat не существует.
Введите имя файла:
same.dat
Мы найдем золото!

 Карта возможных границ месторождения золота
= = = = = = = = = = = = = = = = = = = = = = = =
 0 1 2 3 4 5 6 7
0
1
2
3
4
5
6
7
Удачного поиска!
```

Другими словами, если между значениями замеров нет различия, значит, можно делать вывод об отсутствии месторождения.

---

* При выполнении программ, в которых используется класс `Filer` и предусмотрено считывание данных из файла, сначала система выдает сообщение об исключительной ситуации, после чего требуется ввести в командной строке имя соответствующего файла — *Прим. ред.*

6.3. Класс Graph

Глубинное понимание проблемы в значительной степени зависит от визуального представления того, что происходит. Таблица дает менее четкое представление об исследуемом процессе, чем, например, график, даже примитивный. К тому же, пропустить ошибку гораздо легче при составлении таблицы, чем графика.

Java предоставляет эффективные средства для черчения на уровне точек и линий. Но конструирование графиков (которые к тому же дополнены осями) из таких примитивных компонентов представляет собой довольно длительный процесс. Чтобы обеспечить необходимую глубину рассмотрения темы и в то же время освободить программиста от монотонной работы, мы создали очень легкий в использовании класс `Graph`. Как и два остальных класса, `Display` и `Stream`, класс `Graph` будет использоваться далее в практикуме.

Методы класса Graph

Изучая класс `Graph`, мы сделаем акцент на десяти методах и четырех константах, описание которых приведено ниже.

Класс Graph

```
// Основные методы
new Graph()              - создание экземпляра класса.
add(x, y)                - добавляет точку к списку; ожидается, что точки
                           будут указываться в порядке, соответствующем
                           направлению оси x.
showGraph()              - вызов этого метода является обязательным и
                           приводит к фактическому черчению графика, а также
                           делает окно видимым.

// Дополнительные методы
new Graph (graphTitle, xAxisTitle, yAxisTitle)
                         - версия создания экземпляра, позволяющая вывести
                           метки рядом с осями. Если не все заголовки
                           используются, применяйте пустые строки.
nextGraph()              - строит новый график, используя те же оси. Для всех
                           графиков применяется единственный вызов showGraph.
setColor (int 0 to 3)    - служит для выбора цвета; возможные варианты -
                           black, magenta, blue и red (вместо цифр можно
                           указывать константы).
setSymbol (boolean)      - определяется исходя из указанного цвета (удобен,
                           если цвета уже назначены).
setSymbol(int 0 to 3)    - предназначен для черчения круга, перевернутого
                           треугольника, треугольника и квадрата
                           (используется, если цвета не назначены).
setline (boolean)        - по умолчанию имеет состояние on (включен), можно
                           отключить (off). Применяется, если линии между
                           точками не нужны.
setTitle (String)        - используется для вывода под графиком текстовой
                           строки.
black, magenta,          - константы для назначения цвета. Эквивалентны
blue, red                  цифрам 0, 1, 2, 3.
```

Чтобы использовать класс Graph, мы должны объявить объект Graph:

```
Graph g = new Graph ();
```

Кроме того, мы можем указать в качестве параметров названия графика и осей, как объяснялось выше. После этого все, что нам нужно будет сделать, — это добавить к графику каждую точку (x, y) и вывести его на экран. Итак, для построения простейшего графика нам нужны только три метода.

Принципы работы класса Graph

Класс Graph принимает точку, предоставленную ему, и добавляет ее в список точек. Когда вызывается метод showGraph, при определении масштаба для выведения осей на экран используются минимальные и максимальные значения x и y существующих точек. Заранее установленных осей, по которым можно было бы ориентироваться при вводе точек, не существует, поэтому каждое окно с графиком будет иметь специфический комплект осей.

Начало и конец каждой оси маркируются соответствующим образом. В классе не предпринимаются попытки разместить другие метки вдоль осей, поскольку невозможно предугадать, что пользователь захочет изобразить на графике.

Дополнительные возможности

Прежде чем мы перейдем к рассмотрению дополнительных методов, предоставляемых классом Graph, мы изучим возможность построения нескольких графиков. Поскольку Graph — это класс, мы можем создать несколько объектов Graph, каждый из которых будет иметь собственное окно. Программа, следовательно, сможет добавлять точки к различным окнам. Эта технология удобна в тех случаях, когда необходимо, чтобы оси различались.

Если же требуется сравнить различные графики на одних и тех же осях, следует применять метод nextGraph. Этот метод создает для нового графика набор точек и переустанавливает параметры следующим образом:

```
d.plotType = black;
d.symbolRequired = false;
d.colorRequired = false;
d.titleRequired = false;
d.lineRequired = true;
```

С помощью рассматривавшихся методов указанные параметры могут быть изменены программой. Полный комплекс изменений представлен ниже:

```
g.setColor(blue);
g.setSymbol(true);
g.setTitle("Осадки");
g.setline(false);
```

Существует взаимосвязь между цветом и символом: им соответствуют значения от 0 до 3. При отсутствии инструкций, в которых указывались бы иные значения (другой цвет или символ), число, используемое для указания символа, применяется и для задания цвета. Таким образом, круги строятся из черных линий, треугольники — из голубых и т.д. Для получения кругов голубого цвета следует в приведенный выше комплект инструкций добавить такую строку:

```
g.setSymbol(0);
```

Конечно, цветные графики эффектно выглядят на экране, но при печати они превращаются в набор слабо различающихся оттенками серых и черных линий. Поэтому в графиках, предназначенных для печати, рекомендуется использовать символы. В нижней части графика принято раскрывать значение каждого символа. Аналогичные пояснения для цветов не приводятся.

Пример 6.6. Определение уровня осадков

Задача. Гидрометцентр Саванны хранит данные о ежемесячном уровне осадков за последние 20 лет. Необходимо вычислить следующее:

- средний уровень осадков для каждого месяца;
- стандартное отклонение данного показателя для каждого месяца.

Решение. Таблица уровней осадков, любезно предоставленная нам клерком управления метеорологии, выглядит так:

Год	янв	фев	мар	апр	май	июн	июл	авг	сен	окт	ноя	дек
1989	20	22	17	14	5	0	0	0	7	12	30	20
1990	22	24	19	12	0	0	3	0	8	15	20	25
1991	17	17	17	15	0	0	0	0	6	17	8	20
1992	10	10	10	5	0	0	0	0	0	12	10	15
1993	10	10	10	5	0	0	0	0	0	12	10	15
1994	20	22	17	14	5	0	0	0	7	12	30	20
1995	22	24	19	12	0	0	3	0	8	15	20	25
1996	17	17	17	15	0	0	0	0	6	17	8	20
1997	25	30	25	15	7	0	0	0	20	15	20	30
1998	25	30	25	15	7	0	0	0	20	15	20	30

Данные должны быть сохранены в файле именно в таком виде. По мере ввода значения сохраняются в матрице, которая проиндексирована по годам и месяцам. Поскольку далее в наших вычислениях ключевыми будут значения уровней осадков за разные месяцы, матрицу нужно организовать так, чтобы обеспечить обработку и перемещение колонок значений целиком. Для этого мы укажем в качестве первого индекса месяцы, а в качестве второго — годы (рис. 6.12).

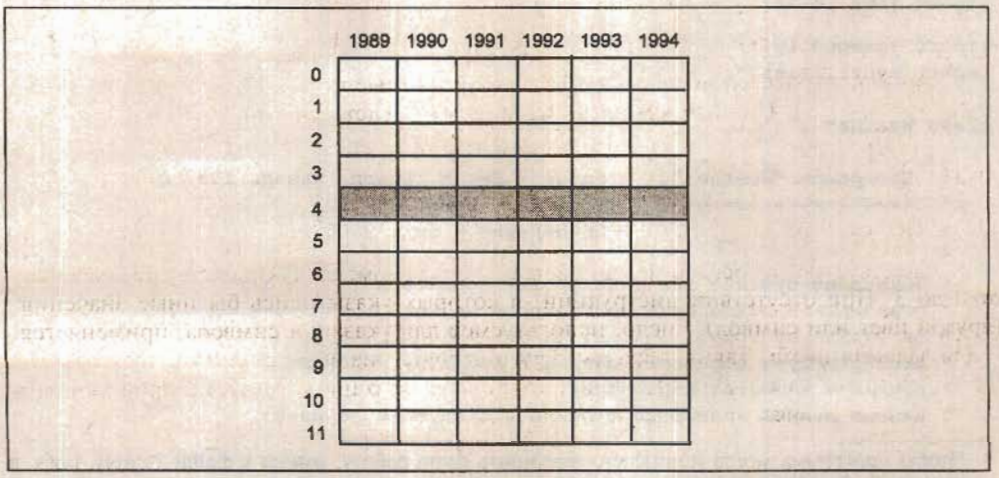


Рис. 6.12. Часть матрицы для хранения данных по уровням осадков

Вот как будут выглядеть необходимые объявления для нашей таблицы:

```
static final int maxyear = 70;
double rainTable [] [] = new double [13] [maxyear];
```

Поскольку в году 12 месяцев, закономерно применять для месяцев индексы от 1 до 12. Мы объявим массив длиной 13, в котором элемент с индексом 0 не употребляется. Теперь в результате ввода `rainTable[4]` мы сможем определить уровень осадков в апреле за все годы, вошедшие в рассматриваемый диапазон. На рис. 6.12 данные за апрель месяц выделены серым цветом.

Разработка программы. В рассматриваемом случае мы имеем дело с классической программой типа считывание/процесс/вывод, где при выполнении процедур ввода/вывода применяются файлы. Поэтому мы создадим один класс для выполнения всех действий и вызовем его из главной программы. Наряду с печатью данных и результатов мы бы хотели начертить график, показывающий средние значения уровней осадков за различные месяцы, а также соответствующие стандартные отклонения. Поскольку при вызове метода `add` класса `Graph` точки добавляются только к одной строящейся линии, мы сохраним значения стандартных отклонений в массиве и в конце программы выполним еще один цикл для создания второго графика.

В процессе обработки значений будут использоваться методы, предназначенные для вычисления среднего значения и стандартного отклонения, которые хранятся в классе `Stats` пакета `myutilities`. Процедура вызова этих методов, описанная в примере 6.3, является довольно простой.

Программа. В программе предусмотрено получение данных из конкретного файла `rain.data`. Можно сделать так, чтобы имя файла являлось переменной и могло быть введено. Поскольку результаты работы программы занимают значительный объем, мы не задавали вывод на экран. Результаты появятся в консольном окне, но могут также быть отправлены в файл. Для этого следует выполнить перенаправление потока вывода, как показано в следующей строке:

```
java Weather>weather.out
```

Ниже дан полный текст программы*.

```
import java.io.*;

import javagently.*;
import myutilities.*;

class Weather {

  /*  Программа "Weather".     Автор - Дж. М. Бишоп, январь 1997 г.
   *  ====================     Графики - июль 1999 г.
   *                           Обновление - июль 2000 г.
   *
   * Вычисляет среднее значение уровня осадков и
   * стандартное отклонение для каждого месяца
   * за указанные годы.
   * Демонстрирует обработку матриц и передачу колонок
   * данных в качестве параметров.
   * Данные должны храниться в файле в следующем формате:
```

* Чтобы программа могла нормально завершить свою работу, нужно в файле `Graph.java` в строке № 93 ввести `System.exit(0);` и выполнить заново компиляцию этого класса. — *Прим. ред.*

```
 * год, за которым следуют 12 значений уровней осадков
 * за каждый месяц этого года.
 */

static class RainBase {

  int    base = 1950;
  int    startyear, endyear = 0; // охватывает период с 1950 года
  // при объявлении всех массивов указывается, что их длина равна 13,
  // поэтому индексы месяцев лежат в диапазоне от 1 до 12.
  double rainTable [] [] = new double [13] [70];
  double averagetable [] = new double [13];
  double stddevTable  [] = new double [13];

  Graph g = new Graph("Уровни осадков", "месяц", "см");

  void readIn () throws IOException {
    Stream fin = new Stream("Rain.dat",Stream.READ);
    int actualYear = 0; // например, 1989
    int yearIndex = 0;  // например, 0

    // Годы могут быть не отсортированы
    // или не упорядочены. Переменная yearIndex начинается с 0 и
    // используется для упорядочения и хранения данных.

    try {
      while (true) {
        actualYear = fin.readInt();
        System.out.print(actualYear+" ");
        if (yearIndex == 0) {
          startyear = actualYear;
        }
        for (int m = 1; m<=12; m++) {
          rainTable[m][yearIndex] = fin.readDouble();
          System.out.print(
              Stream.format(rainTable[m][yearIndex],6,1));
        }
        System.out.println();
        yearIndex++;
      }
    }
    catch (EOFException e) {
      // Фиксирует последний год.
      endyear = actualYear;
      System.out.println("Введены данные за период с "+startyear+
            " по "+ endyear+"\n\n");
    }
  }

  void showResults () {
    System.out.println("Статистика выпавших осадков за период с " +
        startyear + " по " + endyear);
    System.out.println("======================" +
        "===========\n");
```

```
      System.out.println("Месяц\tВ среднем\tСтд. отклонение");
      int nyears = endyear-startyear+1;
      double a;
      g.setTitle("Среднее значение");
      g.setSymbol(true);

      for (int m =1; m<=12; m++) {
        averagetable[m] = Stats.mean (rainTable[m], nyears);
        stddevTable[m] = Stats.stddev
           (rainTable[m], nyears, averagetable[m]);
        System.out.println(Stream.format(m,2)+
           Stream.format(averagetable[m],12,2)+
           Stream.format(stddevTable[m],12,4));
        g.add(m,averagetable[m]);
      }

      g.nextGraph();
      g.setColor(g.blue);
      g.setSymbol(true);
      g.setTitle("Стандартное отклонение");
      for (int m = 1; m <= 12; m++) {
        g.add (m, stddevTable[m]);
      }
      g.showGraph();
    }
  }

  public static void main (String args [])throws IOException {

    RainBase rain = new RainBase();

    rain.readIn ();
    rain.showResults ();
  }

}
```

Тестирование. В результате выполнения программы построен график, изображенный на рис. 6.13, и в файл направлен следующий вывод:

```
1989 20.0 22.0 17.0 14.0 5.0 0.0 0.0 0.0  7.0 12.0 30.0 20.0
1990 22.0 24.0 19.0 12.0 0.0 0.0 3.0 0.0  8.0 15.0 20.0 25.0
1991 17.0 17.0 17.0 15.0 0.0 0.0 0.0 0.0  6.0 17.0 8.0  20.0
1992 10.0 10.0 10.0 5.0  0.0 0.0 0.0 0.0  0.0 12.0 10.0 15.0
1993 20.0 22.0 17.0 14.0 5.0 0.0 0.0 0.0  7.0 12.0 30.0 20.0
1994 22.0 24.0 19.0 12.0 0.0 0.0 3.0 0.0  8.0 15.0 20.0 25.0
1995 17.0 17.0 17.0 15.0 0.0 0.0 0.0 0.0  6.0 17.0 8.0  20.0
1996 25.0 30.0 25.0 15.0 7.0 0.0 0.0 0.0 20.0 15.0 20.0 30.0
1997 25.0 30.0 25.0 15.0 7.0 0.0 0.0 0.0 20.0 15.0 20.0 30.0
1998 10.0 10.0 10.0 5.0  0.0 0.0 0.0 0.0  0.0 12.0 10.0 45.0
Введены данные за период с 1989 по 1998

Статистика выпавших осадков за период с 1989 по 1998
=====================================================
```

```
Месяц    В среднем      Стд. отклонение
 1         16.80          7.9833
 2         18.40          9.5940
 3         15.90          7.5344
 4         10.80          5.4528
 5          1.90          3.1073
 6          0.00          0.0000
 7          0.60          1.2649
 8          0.00          0.0000
 9          7.50          7.3522
 10        13.00          4.9441
 11        14.60          8.7965
 12        23.00         11.5950
```

И вновь мы выбрали реальные данные, хотя это затрудняет проверку программы и нам сложнее убедиться в том, что получены правильные результаты. Но поскольку начерчен график, нетрудно удостовериться, что значения корректны. Кроме того, доподлинно известно, что методы, определяющие среднее значение и стандартное отклонение, работают корректно, поскольку их проверка уже производилась. Следует подчеркнуть, что это преимущество мы получаем при работе с библиотеками.

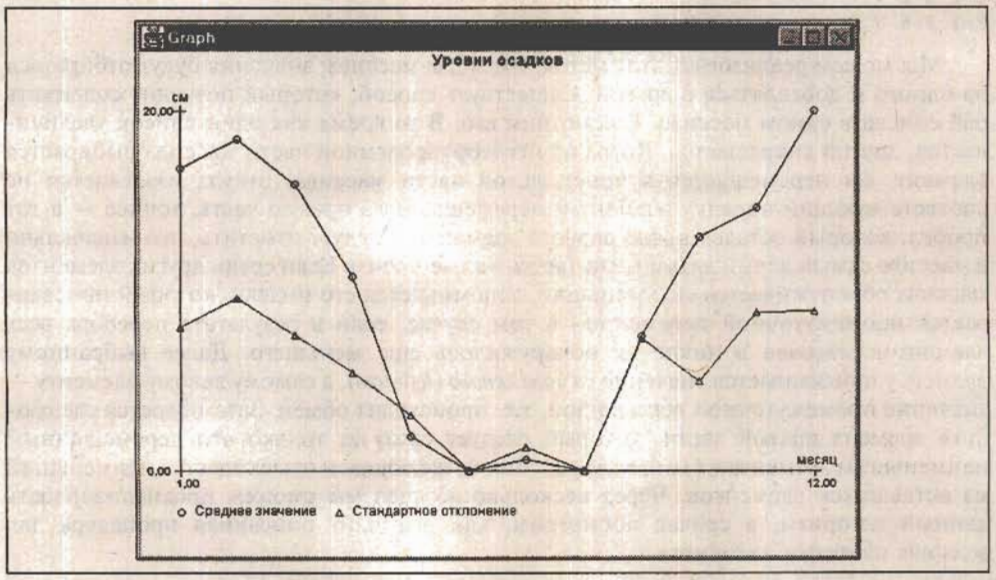


Рис. 6.13. График из программы Weather

6.4. Сортировка и поиск

Сортировка и поиск часто выполняются при вычислениях. Многие системы предоставляют высокоуровневые команды, которые позволяют сортировать данные и выполнять поиск по различным критериям. В первую очередь мы рассмотрим простейший алгоритм сортировки, время работы которого прямо пропорционально квадрату количества элементов, подлежащих сортировке, а также алгоритм линейного поиска, время выполнения которого пропорционально количеству элементов. Другие

алгоритмы (например, Quicksort и двоичный поиск) работают значительно быстрее, но более сложны, поэтому мы изучим их в главе 15.

Сортировка методом отбора

В процессе сортировки элементы перемещают посредством определенного алгоритма до тех пор, пока они не будут упорядочены в соответствии с каким-либо критерием. Метод, используемый некоторыми игроками в карты для их сортировки, заключается в следующем: они держат в правой руке всю колоду, находят наименьшую карту и перекладывают ее в левую руку, затем берут следующую наименьшую из тех, которые находятся в правой руке. В результате такого отбора в правой руке не останется карт, а в левой будет отсортированная колода. Ниже показано, как этот метод работает.

```
Левая рука      Правая рука
                7 3 9 0 2 5
0               7 3 9 - 2 5
0 2             7 3 9 - - 5
0 2 3           7 - 9 - - 5
0 2 3 5         7 - 9 - - -
0 2 3 5 7       - - 9 - - -
0 2 3 5 7 9     - - - - - -
```

Мы можем реализовать этот метод, имея два массива: значения будут отбираться из одного и добавляться в другой. Существует способ, который позволит содержать оба списка в одном массиве. Рассмотрим его. В то время как один список увеличивается, другой сокращается. Когда из неотсортированной части массива выбирается элемент, он перемещается в конец левой части массива, откуда извлекается не соответствующий порядку элемент и перемещается в правую часть, точнее — в тот пробел, который остался после первого элемента. Следует отметить, что изначально в массиве самый левый элемент считается наименьшим. Если среди других элементов массива обнаруживается еще меньший, запоминается его индекс, который присваивается промежуточной переменной в том случае, если в результате перебора всех элементов массива в цикле не обнаружилось еще меньшего. Далее выбранному элементу присваивается значение *самого левого* (*leftmost*), а самому левому элементу — значение промежуточной переменной, т.е. происходит обмен. Затем берется следующий элемент правой части, который следует сразу за только что перемещенным наименьшим элементом (`leftmost+1`). Таким же образом отыскивается наименьший из оставшихся элементов. Через несколько абзацев мы сможем проанализировать данный алгоритм, а сейчас посмотрим, как выглядит описанная процедура для массива из шести элементов.

```
Левая рука      Правая рука
                7 3 9 0 2 5
0                 3 9 7 2 5
0 2                 9 7 3 5
0 2 3                 7 9 5
0 2 3 5                 9 7
0 2 3 5 7                 9
0 2 3 5 7 9
```

Подчеркнуты те цифры, которые были перемещены за один проход цикла. Каждый раз рассматривается список, который становится меньше на один элемент. Так продолжается до тех пор, пока не останется только один элемент. На рис. 6.14 представлена схема этого же алгоритма, которая в большей степени соответствует коду.

Поскольку сортировка — одна из наиболее распространенных операций, целесообразно разработать соответствующий метод. У него должны быть следующие параметры: массив, который нужно отсортировать, и количество элементов, которые подлежат сортировке. Конечно, для формального параметра следует указать, элементы каких типов будут сортироваться. В разделе 9.2 объясняется, как корректно обойти это требование.

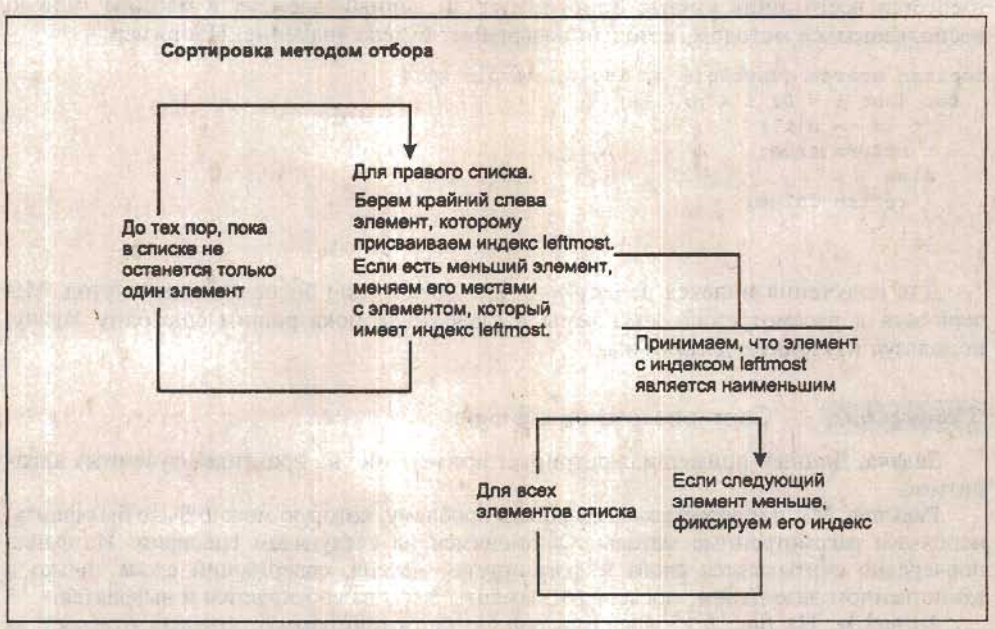


Рис. 6.14. Алгоритм для сортировки методом отбора

```
static void selectionSort (double[ ] a, int n) {
  double temp;
  int chosen;

  for (int leftmost = 0; leftmost < n - 1; leftmost ++) {
    chosen = leftmost;
    for (int j = leftmost + 1; j < n; j ++ )
      if (a[j] < a[chosen]) chosen = j;
    temp = a[chosen];
    a[chosen] = a[leftmost];
    a[leftmost] = temp;
  }
}
```

Сортировку часто приходится выполнять в процессе вычислений, поэтому полезно всегда иметь под рукой надежный алгоритм сортировки. Приведенный выше алгоритм может быть применен к массивам любой длины, содержащим элементы любого типа, а также может использоваться для сортировки в порядке убывания (достаточно в операции сравнения заменить знак < знаком >). Обратите внимание на то, что при сортировке в порядке убывания перемещаемые влево элементы будут наибольшими, а не наименьшими. Рассматриваемый алгоритм сортирует массив элементов, которые имеют тип `double`, для сравнения в нем используется оператор <.

Но если элементами, которые нужно отсортировать, являются объекты, оператор < не подойдет, и мы должны будем заменить его методом `lessThan` или `compareTo`.

Поиск

Простой линейный поиск предусматривает просмотр массива в цикле до тех пор, пока не будет найден нужный элемент или достигнут конец массива. Если же требуется всего лишь узнать, присутствует ли данный элемент в массиве, можно воспользоваться методом, который возвращает булево значение. Например:

```
boolean search (double[] a, int n, double x) {
  for (int i = 0; i < n; i++) {
    if (x == a[i])
      return true;
    else
      return false;
  }
}
```

Для получения индекса найденного элемента нужен более сложный метод. Мы вернемся к рассмотрению этой темы в главе 15, а пока решим еще одну задачу, используя изученные алгоритмы.

Пример 6.7. Сортировка слов в файле

Задача. Данный пример иллюстрирует применение на практике изученных алгоритмов.

Решение. Мы попробуем смоделировать проблему, которую можно было бы решить, используя рассмотренные методы. Остановимся на следующем сценарии. Из файла поочередно считываются слова и формируется массив, содержащий слова только в единственном экземпляре, после чего элементы массива сортируются и выводятся.

Алгоритм. На рис. 6.15 представлена та часть программы, которая отвечает за создание массива уникальных слов. После этого массив выводится, сортируется и выводится еще раз.

Программа. С помощью метода `selectionSort` осуществляется сортировка строк. Как отмечалось в разделе 3.6, оператор < нельзя применять для сравнения объектов (в частности наших объектов `String`). Поэтому мы воспользуемся методом `compareTo` или `compareToIgnoreCase`. Тот же метод будет использован в процессе поиска. Далее следует текст программы.



Рис. 6.15. Алгоритм для создания массива уникальных слов

```
import java.io.*;
import javagently.*;
import myutilities.*;

class SortWords {

    /* Программа "Сортировка слов".  Автор - Дж. М. Бишоп, февраль 1997 г.
     * ============================  Java 1.1 - октябрь 1997 г.
     *                               Обновление - июнь 2000 г.
     * Считывает и сортирует до 100
     * различных слов.
     * Демонстрирует сортировку и поиск в массиве.
     */

  void selectionSort(String[] a, int n) {
    String temp;
    int chosen;

    for (int leftmost = 0; leftmost < n-1; leftmost++) {
      chosen = leftmost;
      for (int j = leftmost+1; j < n; j++)
        if (a[j].compareTo(a[chosen])<0)
              chosen = j;
      temp = a[chosen];
      a[chosen] = a[leftmost];
      a[leftmost] = temp;
    }
  }

  boolean search (String[] a, int n, String x) {
    for (int i = 0; i < n; i++)
      if (x.compareTo(a[i])==0) return true;
    return false;
  }

  void report(String[] a, int n) {
    for (int i = 0; i < n; i++) {
      System.out.print(a[i]+"\t");
      if (i>0 && i % 7 == 0) System.out.println();
    }
  }

  SortWords() throws IOException {

    String group [] = new String[100];
    System.out.print("Где находятся слова? ");
    Stream fin = Filer.open("");

    int count=0;
    try {
      while (count < 100) {
        String word = fin.readString();
        if (! search(group,count,word)) {
          group[count] = word;
```

```
        count ++;
      }
    }
  } catch (EOFException e) {
    System.out.println(count + "слов считано.\n\n");
  }

  System.out.print("Первоначальный массив\n");
  System.out.print("=====================\n");
  report(group,count);

  selectionSort(group, count);

  System.out.print("\n\nОтсортированный массив\n");
      System.out.print("=====================\n");
  report(group,count);
}

public static void main(String[] args) throws IOException {
  new SortWords();
}

}
```

Тестирование. Разработана программа, которая для открытия файла использует класс `Filer`, благодаря чему возможна работа с любыми данными. Применяется метод сортировки `selectionSort`. Чтобы упростить процедуру считывания и максимально увеличить количество слов, которые могут повторяться, мы реорганизовали файл и ввели пробелы между всеми словами и символами. Полученный результат довольно интересен:

```
Где находятся слова? Файл не существует.
Введите имя файла:
sort.java
30 слов считано.
Первоначальный массив
=====================
static  void  selectionSort  (  String  [    ]  a
,       int   n              )  {       temp ;
chosen  for   leftmost       =  0       <    -
1       ++    j              +  if      .    compareTo
}
Отсортированный массив
=====================
(         )       +              ++      ,      -     . 0
 1         ;       <              =       String [     ]
a         chosen  compareTo      for     if     int   j
leftmost  n       selectionSort  static  temp   void  {
}
```

Вывод программы позволяет получить представление об используемых Java принципах сортировки операторов, букв и цифр. Дальнейшее развитие данная тема получит в примере 5.4.

6.5. Таблицы, не зависящие от класса

В этом разделе мы изучим очень полезную структуру — словарь. Словарь имеет следующие характеристики:

- ключи преобразуются в значения;
- ключи и значения могут быть любого типа;
- существует быстрый способ нахождения ключа.

В реальном толковом словаре слова снабжаются пояснениями, а поиск удобен и может осуществляться достаточно быстро потому, что рубрики (слова, значения которых раскрываются) отсортированы в алфавитном порядке. Зная это, мы можем очень быстро отыскать место, где находится нужное слово (см. раздел 15.2).

Проблема состоит в том, что изученные нами массивы не подходят для концепции словаря. Они обладают рядом недостатков:

- имеют фиксированный размер, создаются внутри программы и не могут быть увеличены;
- индекс всегда имеет тип `integer`;
- индекс всегда начинается с нуля.

Ограничение, связанное с фиксированным размером, можно обойти путем создания массива значительного размера (см. пример 6.7). Хотя при объявлении массива и была определена длина в 100 элементов, производился подсчет действительного числа элементов (приблизительно — 30). Что касается второго ограничения, рассмотрим следующее требование: для каждой буквы алфавита необходимо хранить значение. Применение массива было бы идеальным решением, но в таком случае мы не можем при индексировании его элементов употреблять символы (типа `char`), а также с легкостью преобразовывать символы в целые числа и наоборот. Наконец, в примере 6.6 показано, что даже если индекс имеет тип `integer`, в некоторых случаях последовательности данных не начинаются с нуля и приходится прибегать к ухищрениям, чтобы выполнить условие и вписаться в структуру массива, где индексация начинается с нуля.

К счастью, Java предоставляет эффективные средства по работе с массивами, в частности стандартные классы пакета `java.util`. Первый класс — `Vector` — сохраняет идею массива, но снимает ограничения, связанные с фиксированным размером. Второй класс — `Hashtable` (базируется на классе `Dictionary`) — обеспечивает полный комплекс сервисов преобразования, благодаря которому он еще известен как **ассоциативный массив**.

Класс Vector

Класс `Vector` определяет последовательности объектов, длина которых может увеличиваться по мере необходимости. Фактически она увеличивается автоматически без явного запроса. Объекты добавляются в структуру "вектор" и извлекаются из нее с помощью специальных методов. Отдельный метод предусмотрен и для доступа к элементам вектора. В этой главе мы не коснемся больше класса `Vector` по двум причинам: во-первых, следующий класс, который нам предстоит изучить, является более общим и более интересным, а во-вторых, терминология, используемая для описания векторов, несколько отличается от применяемой для других классов Java, *что может нарушить сложившуюся у нас стройную картину программирования на* языке Java. Класс `Vector` входит в состав пакета `java.util`.

Хеш-таблицы

Вот что нам действительно нужно, так это динамически изменяемая структура, которая к тому же имела бы свойства, присущие структурам с индексацией, т.е. была бы универсальной. Хеш-таблицы Java полностью удовлетворяют этим требованиям. **Хеш-таблица** представляет собой коллекцию пар элементов — ключей и значений. Ключом называется индекс к элементу таблицы, а значение — это то, что ассоциируется с ключом. На основании сказанного мы можем сделать заключение относительно того, что массив — это особый вид хеш-таблицы, где ключ имеет тип `integer`. В хеш-таблице ключом может быть объект любого типа, в том числе `Curio`, `String`, `Times` и т.д. Важно, чтобы ключ не был примитивного типа, если только он не помещен в оболочку. Мы поработаем с оболочками чуть позже.

Характерным примером хеш-таблицы может служить та, которая содержится в классе `Display` (рис. 6.16). Она базируется на выводе программы, в которой применялся этот класс и вывод которой был представлен на рис. 4.1. Когда в части программы, отвечающей за ввод данных, мы получаем информацию из поля ввода, которое на экране обозначено меткой и содержит некоторое первоначальное значение (интерфейс класса `Display`), эти данные помещаются в хеш-таблицу с меткой в качестве ключа. Впоследствии, когда программе потребуется это значение, она обратится к хеш-таблице, предоставит метку, которая будет рассматриваться как ключ, и получит требуемое значение.

Если мы сравним изображение на рис. 6.16 с примером, показанным на рис. 6.1, то заметим существенное различие, заключающееся в том, что по отношению к структуре, в основе которой лежит массив, индексы являются внешними, в то время как в данном случае они являются ее частью — ключами. Кроме того, значения после ввода не являются смежными — между ними могут быть пробелы. Это не оказывает негативного влияния на доступ к данным и даже может ускорять его.

В заключение следует сказать несколько слов о термине "хеш-таблица". Термин "хеш" означает, что элементы сохраняются в таблице не последовательно, а таким способом, который позволяет эффективно извлекать их из таблицы. Эффективность работы с таблицей, которая не заполнена полностью, может приблизиться к той, которая характерна для работы с массивом. В результате элементы хеш-таблиц не упорядочиваются по ключу. С другой стороны, ожидается, что в словаре элементы будут размещены в привычном порядке.

Ключ	Значение
Min	45
Hour	13

Рис. 6.16. Пример словаря, или хеш-таблицы

Объявление хеш-таблицы и доступ к ней

Прежде чем мы приступим к рассмотрению процессов объявления хеш-таблицы и доступа к ней, отметим, что в Java есть особый класс, который называется `Object`. В главе 9 мы убедимся в том, что все объекты совместимы с этим классом. Следовательно, класс `Hashtable` имеет ключи и значения, которые являются объектами суперкласса `Object`, и мы можем добавлять в него собственные объекты любого класса (выполнять операцию приведения типа). Ниже показаны некоторые способы объявления и обработки хеш-таблицы:

Объявление хеш-таблицы и доступ к ней

```
Hashtable tableid = new Hashtable ( );

void  put (Object key, Object value);
Object   get (Object key);
boolean containsKey (Object key);
boolean contains (Object value);
void remove (Object key);
```

Конструктор создает объект класса `Hashtable`, именуемый `tableid`, с некоторым назначенным по умолчанию количеством записей, ключи и значения которых являются объектами базового класса `Object`.
Что касается методов доступа к хеш-таблице, то об их назначении легко догадаться из названий.

Предположим, мы хотим ассоциировать названия времени суток с числовыми значениями. Мы можем выполнить следующее:

```
Hashtable timesOfDay = new Hashtable ();
Times time;

timesOfDay.put ("Полдень", new Times (12,00));
timesOfDay.put ("Полночь", new Times (0,00));
timesOfDay.put ("Вечерние новости", new Times (22,00));
time = (Times) timesOfDay.get ("Полдень");
```

В результате выполнения последнего оператора из таблицы будет успешно возвращено значение времени, соответствующее полудню — 12:00. Но почему во время присвоения переменной `time` возвращаемого значения выполняется операция приведения типа в соответствии с объектом класса `Times`? Причина заключается в том, что хеш-таблица потенциально доступна для объектов любого типа. Метод `get` возвращает объект, имеющий тип `Object`. Для того чтобы обеспечить получение объекта нужного типа (`Times`), Java требует, чтобы по отношению к возвращаемому объекту `Object` была произведена **операция приведения типа**.

Установление численности

С хеш-таблицами вплотную связан класс `Enumeration`. Фактически это особый вид класса, называемый **интерфейсом** (см. главу 9). На данном этапе указанный нюанс не имеет особого значения. Класс `Enumeration` предоставляет средства для итерации коллекции объектов. Ниже представлено объявление этого интерфейса и указаны его методы.

Интерфейс Enumeration

```
Enumeration e = table.keys();

Object nextElement ();
boolean hasMoreElements ();
```

`e` — это объект класса `Enumeration`, который базируется на ключах данной таблицы `table`.
Обращение `e.nextElement()` возвращает следующий элемент — `nextElement` — в списке ключей, который представляет собой переменную типа `Object`.
Обращение `e.hasMoreElements()` возвращает значение `true` или `false`, в зависимости от того, исчерпан список ключей или нет.

После объявления интерфейса `Enumeration` для данной последовательности объектов (каковой является последовательность `keys`) метод `nextElement` будет предоставлять следующий элемент в цикле. Можно задать, чтобы этот метод работал до тех пор, пока метод `hasMoreElements` не вернет значение `false`. В качестве примера рассмотрим хеш-таблицу `timesOfDay`. Цикл, обеспечивающий вывод ключей, выглядит так:

```
for (Enumeration e = timesOfDay.keys(); e.hasMoreElements(); i++) {
  String s = (String) e.nextElement();
  display.println(s);
}
```

В результате выполнения данный цикл может вывести следующие ключи (что зависит от того, были ли они введены в таблицу):

```
Полночь
Полдень
Вечерние новости
```

Поскольку требуется приведение типа, та часть цикла, которая отвечает за предоставление нового объекта `String`, выполняется как отдельная операция присвоения в теле цикла, а не одновременно с частями, отвечающими за инициализацию и проверку, как это обычно происходит в циклах `for`.

Доступ к значениям в хеш-таблице

Теперь, когда обеспечен поочередный доступ к каждому из ключей, мы можем получить соответствующие значения из хеш-таблицы. Для этой цели мы воспользуемся методом `get`. Например, приведенный выше цикл можно расширить таким образом:

```
for (Enumeration e = timesOfDay.keys(); e.hasMoreElements(); i++) {
  String s = (String) e.nextElement();
  Times t = (Times) timesOfDay.get(s);
  display.println(s + " " + t);
}
```

Взяв на вооружение все полученные сведения, приступим к решению примера 6.8.

Пример 6.8. Конвертер валют

Задача. Необходимо обеспечить возможность просмотра курса `ZAR` (южноафриканские рэнды) к валютам разных стран и выполнение конвертирования вводимых сумм.

Способ решения. На Web-узле `www.xe.net` ежедневно обновляются курсы валют всего мира. На основе публикуемых там данных мы можем сформировать таблицу, фрагмент которой показан ниже:

```
Денежная единица                    ZAR/единица (валюты)  Единица/ZAR
=================================== ====================  ===========
DZD Алжирские динары                0.777                 12.9
USD Американские доллары            4.556                 0.2195
ARP Аргентинские песо               4.56                  0.2195
AUD Австралийские доллары           3.356                 0.2980
ATS Австрийские шиллинги            0.3614                2.767
BSD Багамские доллары               4.556                 0.2195
BBD Барбадосские доллары            2.265                 0.4415
DEF Бельгийские франки              0.1231                8.121
```

В каждом ряду таблицы содержится по 4 значения — код, название валюты данной страны и два курса обмена. Как сказано в условии задачи, в качестве индексов должны использоваться элементы второго поля — названия стран. Поскольку название страны представляет собой строку, мы не можем хранить данные в массиве, но можем использовать в программе хеш-таблицу.

Мы введем все значения в хеш-таблицу, ряд за рядом, а затем будем обращаться к ней, используя ключ, которым является название страны. Если название страны окажется действительным, мы получим курс обмена и сумму в валюте указанной страны, которая с учетом курса будет равна введенной нами сумме в `ZAR`.

Конструкция класса. В совокупности поля одного ряда образуют объект, для которого мы должны создать класс. В отличие от разработанных раннее классов, этот предназначается не для предоставления каких-либо методов, а для приема и хранения данных. Поэтому класс может в дополнение к своим основным функциям предоставить метод для ввода данных в свои поля из файла. Давайте назовем этот класс `Rates`.

Поскольку мы уже решили, что будет представлять собой класс `Rates`, можно создать хеш-таблицу на основе объектов и обрабатывать ее описанным способом. Диаграмма на рис. 6.17 поможет нам решить, какой метод избрать. Таблица представлена как объект класса `Hashtable`. Поскольку класс `Hashtable` входит в пакет `java.util`, мы не имеем доступа к его структуре, но знаем, что он содержит ключи и значения. Это и отражено на диаграмме. Ключи в таблице — это объекты класса `String`, а значения — объекты класса `Rates`[4 255].

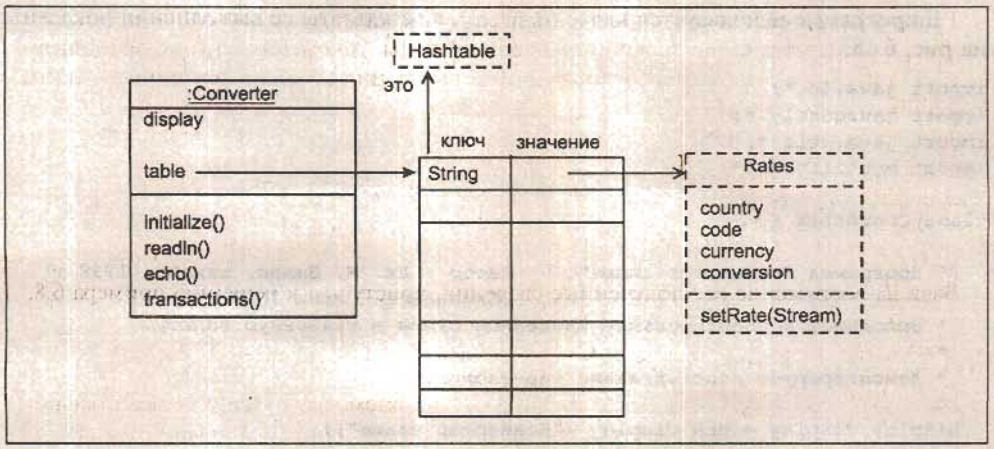


Рис. 6.17. Диаграмма моделирования для программы "Конвертер валют"

Программа. Сначала рассмотрим класс `Rates`.

```
import java.io.*;
import javagently.*;

class Rates {
  /* Класс Rates.          Автор - Дж. М. Бишоп, декабрь 1998 г.
   * -----------
   * Хранит название страны, валюту, код и курс обмена.
   */

  String country;
  String code;
  String currency;
  double conversion;

  void setRate (Stream in) throws IOException {
    code = in.readString();
    country = in.readString();
    currency = in.readString();
    // Нам не нужен первый курс, поэтому мы его
    // проигнорируем, записав следующий поверх него
    conversion = in.readDouble();
    conversion = in.readDouble();
  }

}
```

Программа "Конвертер валют" реализует все те положения, которые мы рассматривали выше. Стоит обратить внимание на строки метода `transactions`, где из хеш-таблицы извлекаются объекты и выполняется приведение типа.

```
c = display.getString("Страна");
if (table.containsKey(c)) {
  amount = display.getDouble("Сумма");
  r = (Rates) table.get(c);
 ... и т.д.
```

В программе используется класс `Display`, а результаты ее выполнения показаны на рис. 6.18.

```
import java.io.*;
import javagently.*;
import java.util.*;
import myutilities.*;

class Converter {

  /* Программа "Конвертер валют".    Автор - Дж. М. Бишоп, декабрь 1998 г.
   * ----------------------------    Обновление - июнь 2000 г.
   * Выполняет конвертирование введенной суммы в указанную валюту.
   *
   * Демонстрирует использование хеш-таблиц.
   */
  Display display = new Display ("Конвертер валют");
  Hashtable table = new Hashtable();
```

```
Converter () throws IOException {
  initialise();

  readIn();
  echo();
 transactions();
}


  // Вводит строку и сохраняет ее в хеш-таблице,
  // указывая в качестве ключа название страны.

  void initialise () {
    display.println("Конвертер валют\n"+
                    "==================");
    display.prompt("Файл с курсами обмена", "rates2.dat");
    display.ready("После вода имени файла,\n нажмите кнопку \"Готово\" ");
  }

  void readIn() throws IOException {
    Rates rate;
    String filename = display.getString("Файл с курсами обмена");
    Stream fin = Filer.open(filename);
    try {
      for (int i = 0; ; i++) {
        rate = new Rates();
        rate.setRate(fin);
        table.put(rate.country, rate);
      }
    }
    catch (EOFException e) {
      display.println("Данные введены и сохранены\n");
    }
  }

  void echo() {
    display.println("Страны, указанные в таблице:");
    int i = 1;
  // while (i < 8) {

  for (Enumeration e = table.keys(); e.hasMoreElements();i++) {
      String country = (String)e.nextElement();
      display.println(country);

   }
  }

  void transactions () {
    display.prompt("Страна", "American");
    display.prompt("Сумма", 1000);
    String country;
    double amount;
    String c;
    Rates r;
```

```
    while (true) {
      display.println("Чтобы просмотреть имеющиеся в таблице \nстраны,");
      display.println("воспользуйтесь полосой прокрутки");
      display.ready("Введите название страны, сумму и нажмите \n кнопку
                    \"Готово\" ");
      c = display.getString("Страна");
      if (table.containsKey(c)) {
        amount = display.getDouble("Сумма");
        r = (Rates) table.get(c);
        display.println( Stream.format(amount,6,2) +" ZAR в "+c+" "
            +r.currency+"\n будут равны "+
            Stream.format(amount * r.conversion,6,2)+"\n\n\n");
      }
      else {
        display.println("Извините, но страны "+c+" нет в таблице\n");
      }
    } // закрывающая скобка цикла while
  } // закрывающая скобка метода transactions

  public static void main(String[] args) throws IOException {
    Converter data = new Converter ();
  }

} // закрывающая скобка класса Converter
```

Тестирование. На рис. 6.18 показан приблизительный результат выполнения программы. Чтобы протестировать программу на должном уровне, следует взять значения, выраженные в широко распространенных валютах. Еще один вариант проверки — конвертировать сумму в ZAR в ту же валюту и удостовериться в том, что значения совпадут. Разработка этого примера будет продолжена в следующих главах, когда мы изучим способы конвертирования одной выбранной валюты в другую.

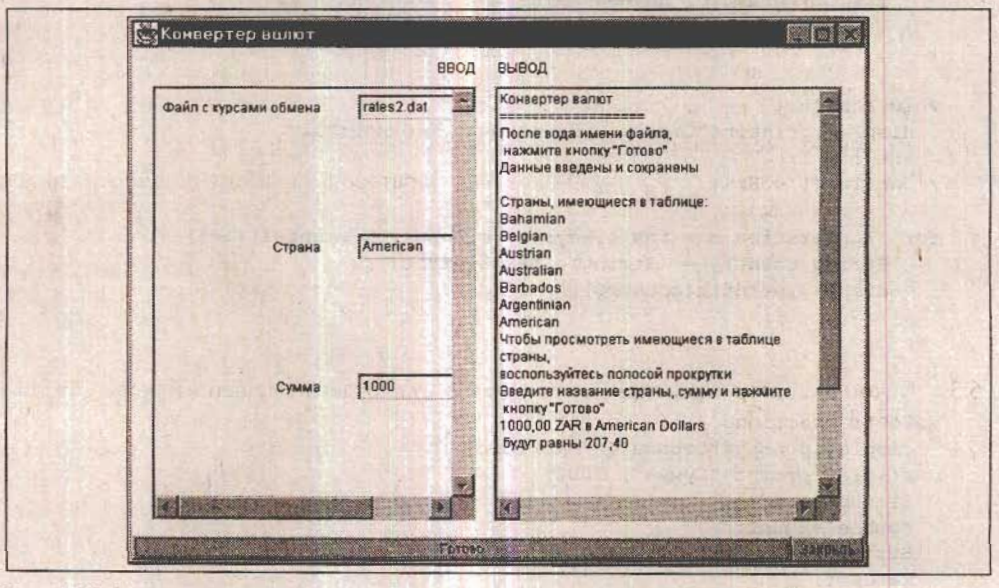


Рис. 6.18. Интерфейс класса `Display` для программы "Конвертер валют"

Если вам не удастся получить доступ к Web-узлу `www.xe.net`, вы можете воспользоваться файлом `curtable`, который находится в каталоге `Other Materials` Web-узла `Java Gently`.

Резюме

Массивы предоставляют многочисленные значения, к которым можно обратиться, используя одно и то же имя (но разные индексы). Для массивов в Java устанавливается фиксированная длина, но в то же время они могут иметь несколько измерений. Существует несколько способов взаимодействия между классами и массивами. Для решения каждой задачи необходимо выбирать соответствующую технологию. Полезной характеристикой Java является возможность объявить метод с указанием в качестве параметра массива, длина которого не конкретизируется. В методах сортировки также используются массивы. В частности, в настоящей главе была подробно описана сортировка методом отбора.

В завершение мы рассмотрели, как можно обойти присущие массивам ограничения, такие как фиксированный размер и применение в качестве индексов только целых чисел. Наиболее эффективным средством в данном случае является хеш-таблица, значениями которой могут быть объекты любого класса, которые, в свою очередь, могут индексироваться объектами любого другого класса. Результатом запросов к хеш-таблице могут быть как значения, так и ключи. Для обработки хеш-таблицы в цикле мы применяли интерфейс `Enumeration`, который работает с ключами таблицы.



Вопросы

6.1. Почему для массива `periods` (рис. 6.2) выражение `periods[7].mins` является недействительным:

а) `mins` не является методом, определенным в массиве `periods`

б) индекс `7` выходит за пределы, установленные для длины массива

в) 7-й объект не был создан

г) выражение должно выглядеть так — `periods[7].mins()`

6.2. Вспомните класс `IntArray`. Какой из операторов надлежит использовать для создания экземпляра этого класса?

а) `IntArray A = new IntArray()`

б) `IntArray A = new IntArray(100)`

в) `IntArray A[] = new IntArray[100]`

г) `IntArray A(100)`

6.3. Какой оператор гарантирует, что класс `Stats` будет помещен в пакет `myutilities`?

а) `imports myutilities`

б) `package myutilities;`

в) `package Stats;`

г) `import myutilities.Stats;`

6.4. Если мы создали таблицу строк размером 10×10, что послужит обращением к строке, находящейся в пятом ряду и в первой колонке?

а) `table [5] [1]`

б) `table [5,11]`

в) `table [4] [0]`

г) `table [4] [1]`

6.5. Какой метод мы должны вызвать при использовании класса `Graph`, чтобы увидеть график?

а) `showGraph()`

б) `nextGraph()`

в) конструктор

г) `SetVisible(true)`

6.6. Метод `selectionSort` имеет в качестве параметра переменную `n`, а также массив `A`, потому что:

а) внутри метода не указана длина массива `A`

б) может потребоваться сортировка только части элементов массива `A`

в) значение переменной `n` равное нулю служит признаком того, что массив еще не инициализирован

г) фактически, `n` не является необходимым параметром

6.7. Ключи и значения в хеш-таблице могут быть (далее "любой тип" означает объекты или примитивные значения):

а) любого типа

б) только объектами

в) ключи могут быть любого типа, значения должны быть объектами

г) значения могут быть любого типа, ключи должны быть объектами

6.8. Какой оператор необходим для получения строкового значения (тип `String`) из хеш-таблицы, которая называется `table`?

а) `String s = t.get();`

б) `String s = (String) t.nextElement();`

в) `String s = (String) t.get();`

г) информации недостаточно

6.9. Если массив объявлен как `int A[] = {2, 4, 6, 8};` и мы попытаемся обратиться к его элементу `A[5]`, что мы получим?

а) значение 8

б) значение 0

в) объект-исключение `arrayOutOfBoundsException`

г) объект-исключение `NullPointerException`

6.10. Какое выражение позволит определить количество элементов в массиве?

а) `A.length()`

б) `length(A)`

в) `A.length`

г) `A.length-1`

Задачи

6.1. **Увеличивающиеся массивы.** Массивы имеют фиксированную длину, но мы можем заменить существующий массив большим, если его размер окажется недостаточным. Предположим, объявлен массив `A`, имеющий определенную длину. После попытки добавить в него очередное значение возникла исключительная ситуация `arrayOutOfBoundsException`. Мы можем вызвать метод `grow`, который создаст новый массив на 50 % большим, скопирует все значения и вернет ссылку на новый массив. Напишите такой метод и протестируйте его в небольшой программе.

6.2. **Таблица расстояний.** В атласе часто публикуются таблицы расстояний между важнейшими городами, названия которых служат заголовками столбцов и колонок. В Java мы не можем создать такую таблицу в виде двухмерного массива, поскольку ряды и колонки в ней являются строками, а не целыми числами. Решение заключается в применении хеш-таблицы, где ключами являются названия городов, а значениями — числа, которые послужат индексами для двухмерной матрицы расстояний между городами (т.е. каждому городу будет присвоен номер, а по двум номерам-индексам можно будет обращаться к двухмерному массиву). Напишите программу, которая:

- охватывает до 20 городов и случайным образом генерирует значения расстояний в диапазоне от 50 до 1000 км;
- использует интерфейс класса `Display` для задания расстояний между любыми двумя городами.

6.3. **Кто в какой группе?** Модернизируйте программу 4.8 (в которой определялся средний возраст) так, чтобы в объект вводились такие значения, как имена детей, их возраст и название группы (буквы от А до Г), а сами объекты хранились в массиве. Отсортируйте элементы массива по значениям возраста и задайте вывод информации о детях. Сможете ли вы отсортировать элементы массива по именам и по обозначениям групп?

6.4. **Оценка преподавательских качеств лектора.** В университете Саванны разработана схема определения уровня преподавательского мастерства лекторов. Преподаватели оцениваются по нескольким критериям: объем знаний, манера вести лекции, умение увлечь аудиторию, использование дополнительных источников (видео, пособий, экспонатов, устройств и т.д.). При оценивании студенты используют пятибалльную шкалу, причем каждый балл соответствует определенному значению. Например, фрагмент анкеты может выглядеть следующим образом:

Голос	**Объем подаваемого**
живой и разнообразный	удовлетворительный
достаточно живой	чрезмерный
удовлетворительный	хотелось бы больше
скорее скучный	слишком много
монотонный	недостаточный

Пять баллов дается за первый ответ, четыре — за второй и т.д. Ответы представлены в файле, где для каждого вопроса уже подсчитано количество студентов, выставивших свою оценку. Например:

```
Р.Н. Смит
6 ноября 2000 г.
COS101 Принципы программирования
80 студентов
```

```
Голос 16 35 20 9 0
Материал 37 20 11 10 2
```

Вот возможный вариант вывода:

```
Р.Н. Смит
6 ноября 2000 г.
COS101 Принципы программирования
Голос
А  живой и разнообразный   20% ********************
Б  довольно живой          44% ********************************************
В  удовлетворительный      25% *************************
Г  скорее скучный          11% ***********
Д  монотонный               0%
```

Требуется разработать программу, позволяющую подобным образом оценить преподавателей.

— Решите, какие файлы потребуются для данных и в каком формате выводить данные.

— Определите, какой должна быть конструкция класса для хранения вводимой информации во время выполнения программы.

— Решите, что вам потребуется в программе: массивы, хеш-таблицы, обе структуры или ни одна из них.

— Напишите программу и протестируйте ее надлежащим образом.

6.5. **Главы книг.** Расширьте класс, разработка которого была начата в задачах 2.10 и 3.19, чтобы он включал больше информации о каждой главе, например список из семи параграфов, количество задач, десять букв, которые являются ответами на вопросы для самоконтроля и т.д. Разработайте интерактивную программу, в которой была бы предусмотрена возможность ввода новых данных о главах по мере написания книги и вывода сведений о ее текущем состоянии. Обратите внимание на то, что в первоначальных версиях программы уже имеются три массива: массив глав, массив параграфов (которые являются объектами своего собственного класса, в котором есть названия строк и целые числа — номера страниц) и массив ответов на вопросы для самоконтроля, элементы которого имеют тип `char`.

6.6. **Всенародные праздники.** Используя хеш-таблицу, создайте программу, позволяющую вводить и сохранять даты всенародных праздников вашей страны для текущего года. С помощью класса `Display` реализуйте следующую функцию: пользователь вводит название праздника и получает взамен его дату. Рассмотрите возможность создания большой хеш-таблицы, в которую были бы внесены все дни года, чтобы при вводе даты можно было узнать, выпадает ли на этот день какой-либо праздник.

6.7. **Средняя величина напряжения.** Для определения среднего напряжения в электросети показания вольтметра фиксируются через каждый час в течение 72 часов. На основе этих данных и вычисляется искомая величина. Напишите программу, в которой предусмотрен ввод 72 показаний вольтметра, и определите:

— среднее напряжение в сети;

— часы, когда отклонение вольтажа от среднего значения превысило 10%;

— любые смежные часы, когда разница между показаниями вольтметра превысила 15 % от средней величины напряжения.

Программа должна выводить на экран график вольтажа за 72 часа. Используйте для этого класс `Graph`.

6.8. **Полезные методы.** Напишите класс, методы которого принимали бы в качестве одного из параметров массив целых чисел и могли бы выполнить следующее:

- найти максимальный элемент;
- определить, равны ли значения элементов;
- определить, сколько раз возникает ситуация, когда значение элемента оказывается больше некоторого заданного уровня.

В этот класс мы хотели бы также добавить методы, позволяющие:

- найти максимальный элемент и, если таких элементов несколько, зафиксировать их позиции;
- определить диапазон значений массива.

Подумайте, почему труднее создать эти два метода и какой вариант взаимодействия между массивом и классом подойдет для реализации поставленной задачи?

6.9. **Максимальный "непроходной" балл.** Ученый совет университета Саванны принял решение, согласно которому, по крайней мере, 75% студентов каждого курса должны переводиться на следующий курс. Это означает, что сумма баллов, позволяющая студенту перейти на следующий курс, будет специфической для каждого курса. Напишите программу, которая считывает из файла оценки студентов и определяет такой максимальный "непроходной" балл, который не нарушал бы директиву сената.

6.10. **Из какой страны судья?** В примере 6.2 вместо номеров судей мы хотели бы выводить названия стран, из которых они прибыли. Используйте массив для создания перечня стран, представители которых занимаются судейством, и модифицируйте программу так, чтобы ее вывод был более информативным. Подумайте, что лучше — иметь в классе `Dive` массив названий стран вместе с массивом выставленных баллов или создать другой класс, включающий названия и баллы?

6.11. **Рост населения.** С 1980 года статистические данные о численности населения Саванны хранятся в файле, где в каждой строке указан год и количество жителей Саванны за текущий год. Напишите программу, которая считывает эти данные из файла и находит два идущих подряд года, когда процент увеличения населения был наивысшим. Возможно, данные придется отсортировать.

6.12. **Календарь на двух языках.** Разработайте программу, которая выводит на экран календарь, где названия месяцев размещаются одно под другим. Используя хеш-таблицу, создайте версию, в которой все названия месяцев и дней недели указаны на иностранном языке (например, французском или немецком). Предоставьте пользователю возможность выбирать, на каком языке должен быть выведен календарь.

6.13. **Множество лампочек.** Модифицируйте программу 6.4 таким образом, чтобы она могла считывать некоторое количество значений из файла и использовала класс `Display`. Затем расширьте ее так, чтобы можно было вести данные, полученные от десяти различных производителей лампочек, и вычислить для каждого из них среднее значение и стандартное отклонение. Начертите соответствующие гистограммы. При желании используйте генератор случайных чисел.

6.14. **Сравнение показателей.** Большинство преподавателей хотели бы сравнить свои показатели с результатами коллег. Соответствующий алгоритм мог бы выглядеть так: создается файл для хранения текущих средних баллов по каждому вопросу;

после проведения нового опроса относительно отдельного преподавателя считывается текущее среднее значение и с учетом количества преподавателей, прошедших аттестацию, определяется новое среднее значение, которое заносится в файл. Реализуйте такое расширение программы. Если у преподавателей будет возможность выбрать вопрос, окажет ли это влияние на выбор структуры для хранения средних баллов (массива или хеш-таблицы)?

6.15. **График дней рождений.** Воспользуйтесь классом `Graph` для создания графика дней рождений (см. задачу 3.14).

6.16. **Сумма квадратов.** Напишите типизированный метод, который принимает в качестве параметров два целых числа и возвращает сумму квадратов целых чисел, находящихся на числовой прямой между ними, включая граничные значения. Как вызвать метод для вывода значения суммы квадратов целых чисел из диапазона от 1 до 10? Каким образом должна измениться программа, если данные сначала нужно ввести в массив?

6.17. **Онлайновое бронирование номеров.** Модифицируйте программы 4.12 и 4.13 так, чтобы во время их работы забронированные номера хранились в массиве или хеш-таблице.

ГЛАВА 7



Форматирование

В этой главе мы рассмотрим различные способы форматирования текста средствами Java. Изначально возможности форматирования этого языка не были разнообразными, но разработчики Java 2 значительно расширили их для представления данных с учетом локальных настроек системы. Это означает, что программы могут отображать данные в соответствии с соглашениями, принятыми в данной стране. Эти детали могут показаться несущественными, но зато они ярко демонстрируют, насколько Java отличается от других языков — его предшественников.

7.1. Строки и их обработка

Строки были первым элементом пакета JavaGently, с которым мы познакомились, и широко применялись в примерах. Мы продолжим развивать эту тему и поговорим о способах их обработки. Строки являются в Java особыми элементами по нескольким причинам:

- `String` — это класс, поэтому строки являются объектами.
- Для строковых констант имеется специальная запись — сокращенный вариант, на основе которого создаются строковые объекты с известным содержанием (например, "America").
- В отличие от других объектов, строки имеют специальный оператор, служащий для их конкатенации.
- Содержимое объекта `String` после создания не может быть изменено.
- Другой класс — `StringBuffer` — позволяет вносить изменения в содержимое строк.
- Существует различие между пустой (но инициализированной) и нулевой (не инициализированной) строками.

Например, в результате объявления

```
String big = "hippopotamus";
```

создается и инициализируется объект (без использования оператора `new`), обращаться к которому можно будет с помощью переменной `big` (рис. 7.1).

Объект `big` показан с необработанными данными, но без привычного указания переменных и методов. Причина в том, что класс `String` является классом Java, внутрь которого мы еще не заглядывали. Однако мы можем получить доступ к некоторым из его методов, о чем рассказывается ниже.

Сравните это объявление с тем, что выполнено для класса `Time` (см. Практикум 1 главы 3):

```
Time noon = new Time (12,0);
```

Здесь константа заключена в скобки и применяется оператор `new`.

Сравнивая пустую и нулевую строки, рассмотрим следующие два объявления:

```
String s1;
String s2 = " ";
```

Строка `s1` будет иметь специальное значение, называемое нулевым, пока не будет присвоено некоторое другое значение. Такая строка не является действительной для методов обработки строк, поэтому попытка получения доступа к ней приведет к возникновению исключительной ситуации `NullPointerException`. Строка `s2` может использоваться методами обработки строк: она просто не содержит ни одного символа. Различия между указанными строками показаны в примере 7.1.

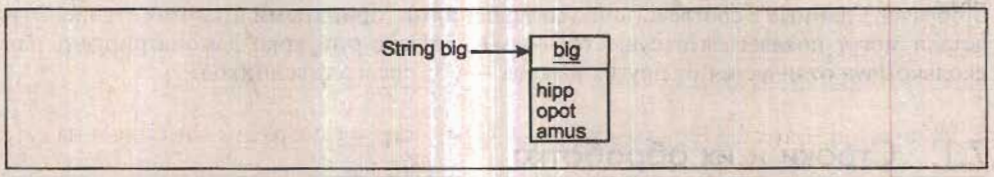


Рис. 7.1. Пример сохранения строки

Строки, заключенные в кавычки или добавленные в объекты класса `String`, не могут быть изменены. Такую возможность предоставляет класс `StringBuffer`, но мы не будем рассматривать его в рамках настоящей книги.

Операции со строками

Класс `String` имеет несколько конструкторов, некоторые из них представлены ниже:

Создание строки

`String s = "chars";`
`String ();`
`String (String value);`
`String (char[] value);`
`String (char[] value, int offset, int count)`
`String (StringBuffer buffer)`

Первая форма объявления с назначением строковой константы является наиболее распространенной. Следующие две конструкции создают пустую строку и предоставляют конструктор копирования соответственно. Строки также могут быть созданы на основе массивов символов и содержимого переменной класса `StringBuffer`.

В классе `String` доступно множество методов. Рассмотрим некоторые из них:

Работа со строками

```
// методы класса
 String  valueOf  (int i)
 String  valueOf  (double d)
 ... и 9 других

// методы экземпляра
char      charAt (int index)
int       compareTo (String s)
boolean   equals (Object obj)
int       indexOf (String s)
int       indexOf (String s, int fromindex)
int       length ()
String    substring (int begin, int end)
Boolean   startsWith (String prefix)
char []   toCharArray ()
String    toLowerCase ()
String    trim ()
... и 33 других
```

Применив метод `length`, мы получим длину строки. Обратите внимание на то, что `length` в массиве является константой, а в данном случае — вызовом метода, в связи с чем требуется наличие скобок. Хотя строка не может индексироваться непосредственно, она состоит из символов, а символ, находящийся на любой проиндексированной (имеющей номер) позиции, может быть получен с помощью метода `charAt`. В случае необходимости вся строка может быть трансформирована в массив символов. Например, в результате ввода конструкций

```
String big = "hippopotamus";
char study [] = big.toCharArray();
```

мы получим массив длиной 12 символов, где каждый символ взят из исходной строки. Метод `indexOf` играет роль поискового средства, предоставляя возможность найти в строке подстроку. Так, например, конструкции

```
String big = "hippopotamus";
int bang = big.indexOf("pop");
int fizz = big.indexOf("up");
```

возвращают значение 3 для переменной `bang` и -1 для переменной `fizz`. Идентичным способом может быть извлечена подстрока, как показано в следующем примере:

```
String small = big.substring(3,5);
```

В результате строке `small` будет присвоено значение `"pop"`.

Методы `compareTo` и `equals` предоставляют возможность сравнивать значения строк, чего не позволяют булевы операторы (см. раздел 3.6). В этом разделе объяснялось также различие между присвоением значений объектам и созданием ссылок на них, что применимо и к строкам. Поэтому сейчас мы отметим, как будут использоваться методы сравнения. Метод `equals` возвращает булево значение. Например, в результате ввода конструкций

```
String big = "hippopotamus";
String small = big.substring(3,5);
System.out.println (big.equals(small));
```

возвращается значение "ложь". Метод compareTo производит сравнение по алфавиту и выдает значения −1, 0 или +1, что соответствует "меньше чем", "равно" или "больше чем". Так, если ввести

```
String big = "hippopotamus";
String small = big.substring(3,5);
System.out.println (big.compareTo(small));
```

получим −1. Другими словами, при использовании алфавитного порядка значение "hippopotamus" предшествует значению "pop".

Одна из распространенных в программировании операций — преобразование строк в числа и наоборот. Методы valueOf могут, используя любой из примитивных типов, создать строку. Они применяются в тех случаях, когда требуется сохранять строку в программе в течение некоторого времени вместо того, чтобы вывести ее немедленно.

Пример 7.1. Удаление двойных пробелов

Задача. Согласно правилам оформления в абзацах не рекомендуется употреблять двойные пробелы. Перед нами стоит задача обработать предложенный текст и заменить двойные пробелы одинарными.

Решение. Для поиска двойных пробелов мы воспользуемся строковым методом indexOf. Содержимое файла будет считано по одной строке за один проход.

Алгоритм. Алгоритмы обработки строк могут быть довольно мудреными. Настоящий лучше применять поэтапно. Давайте начнем с того, что считаем одну строку текста и выведем ее вместе с указанием позиции первого обнаруженного двойного пробела. Если исходить из того, что fin — это класс BufferedReader, фрагмент программы на Java будет выглядеть так:

```
s = fin.readLine();
System.out.println(s.indexOf("  ",s) + "  " + s);
```

Если включить этот фрагмент в программу для обработки целого файла, полученный результат может быть таким:

```
0  Алгоритмы обработки строк могут быть довольно мудреными. Настоящий
25 лучше применять поэтапно.  Давайте начнем с того, что считаем
-1 одну строку текста и выведем ее вместе с указанием
47 позиции первого обнаруженного двойного пробела.  Если исходить из того, что
15 fin — это класс  BufferedReader, фрагмент программы на Java будет выглядеть
4  так:
```

Как мы видим, в первой строке двойной пробел расположен в самом начале. Вторая, четвертая и пятая строки имеют двойные пробелы в средине. Третья строка не содержит двойного пробела, поэтому метод indexOf возвращает −1. Наконец, в последней строке двойной пробел находится в самом конце — после двоеточия.

Для нахождения более одного двойного пробела в строке необходимо создать цикл, который начинает проверку с позиции обнаруженного двойного пробела плюс 2 символа. Общий цикл выглядит следующим образом:

```
startingFrom = 0;
while (true) {
spaceAt = s.indexOf("  ",startingFrom);
if (spaceAt==-1) break;
s = s.substring(0,spaceAt)+" "
```

```
    + s.substring(spaceAt+2,s.length());
  startingFrom = spaceAt+2;
}
 System.out.println(s);
```

Другими словами, мы воссоздаем строку `s` каждый раз с позиции 0 до двойного пробела плюс одинарный пробел и оставшаяся часть строки. Признаком того, что обработка строки завершена, послужит значение −1, возвращенное методом `indexOf`.

Программа. Полная версия программы включает приведенные части, а также выполняет проверку файла до конца. Это достигается путем поиска строки нулевой длины. С целью тестирования программа настраивается на ввод данных с клавиатуры, но поток `fin` с помощью метода `Filter.open` может быть легко перенаправлен в файл по выбору пользователя.

```
import java.io.*;
import javagently.*;

class Spaces {
  /* Программа для удаления двойных пробелов.  Дж.М. Бишоп, декабрь 1997 г.
   * - - - - - - - - - - - - - - - - - - - -   Java 1.1
   *
   * Заменяет двойные пробелы одинарными.
   * Иллюстрирует использование методов обработки строк.
   */
 static int count;
  public static void main (String args []) throws IOException {
     System.out.print("Программа замены двойных пробелов ");
     System.out.println("одинарными.");
     Stream fin = new Stream(System.in);

     count = 0;
     String s = "";
     int spaceAt, startingFrom;
     try {
        while (++count <5) {
          System.out.println("Введите предложение");
          s = fin.readLine();
          if (s==null) throw new EOFException();

          startingFrom = 0;
          while (true) {

          spaceAt = s.indexOf("  ",startingFrom);
            if (spaceAt==-1) break;
            else if (spaceAt==0)
                 s = s.substring(1,s.length());
            else s = s.substring(0,spaceAt)+" "
                        + s.substring(spaceAt+2,s.length());
            startingFrom = spaceAt+2;
          }
          System.out.println(s);

        }
     }
     catch (EOFException e) {}
  }
}
```

Тестирование. Следующие простые строки охватывают многие из тех случаев, которые представляют для нас интерес с точки зрения обнаружения программой.

```
Программа для замены двойных пробелов одинарными.
Введите предложение
Строка один  с одним двойным пробелом.
Строка один с одним двойным пробелом.
Введите предложение
Строка два  с  двумя.
Строка два с двумя.
Введите предложение
Строка три, заканчивающаяся двойным пробелом.
Строка три, заканчивающаяся двойным пробелом.
Введите предложение
  Строка четыре, начинающаяся с двойного пробела.
Строка четыре, начинающаяся с двойного пробела.
Введите предложение
Строка пять содержит   тройной пробел.
Строка пять содержит тройной пробел.
```

Способы усовершенствования рассмотренной программы описаны в приложении к данной главе.

Класс StringBuffer

Как указывалось, строки являются неизменными. Изменить длину или содержимое строки можно за счет разделения ее на части и их объединения требуемым способом (см. пример 7.1). Класс `StringBuffer` предоставляет методы `append`, `insert` и `setcharAt`, которые являются эффективными средствами изменения строк.

Строки и массивы символов

В Java существует различие между строками и массивами символов. Следует отметить, что допустимым является их взаимное преобразование. В следующем примере показано, в каких случаях такое преобразование может быть полезно. Основная идея состоит в следующем: чтобы ввести отдельные символы, мы сначала считываем строку и преобразовываем ее в массив, а затем поочередно считываем по одному символу. Если просто считывать символы, нам придется разделить их пробелами, что довольно неестественно при многих обстоятельствах.

Пример 7.2. Поддельные чеки

Задача. В оборот попали поддельные чеки. Установлено, что все они имеют одинаковые отличительные черты. Например, если в номере чека из 10 цифр имеются:

- три и более последовательных нулей;
- последовательности из четырех и более цифр, отличных от нуля

то этот чек может быть поддельным. Проверку и обнаружение поддельных чеков требуется автоматизировать.

Решение. При обработке чеков банковскими компьютерами в первую очередь считывается номер. Мы напишем программу, в которой можно вводить номера чеков и анализировать их на наличие указанных характеристик.

Алгоритм. Алгоритм (рис. 7.2) предусматривает ввод номера (цифра за цифрой) и подсчет количества нулей и не нулей. Его работа направлена на поиск определенных последовательностей. Если последовательность прерывается, механизм подсчета

запускается заново с этого места. Следовательно, возникнет необходимость зафиксировать, что на некотором этапе достигнут критический счет. Лучше всего это сделать посредством булевой переменной.



Рис. 7.2. Алгоритм проверки чеков

Программа. Первая версия программы работает в соответствии с рассмотренным алгоритмом. В ней для ясности применяются два метода. Цифры считываются как символы. Одна из особенностей класса `Stream` заключается в том, что любые элементы должны разделяться непечатаемыми символами — пробелами, символами окончания строки, символами табуляции. Поэтому при вводе цифр, составляющих номер чека, за каждой из них должны следовать разделители. В этом случае можно будет считать цифры с использованием метода `readChar`.

```
import java.io.*;
import javagently.*;

public class ChequeDetector {

   /* Детектор поддельных чеков. Дж.М. Бишоп, сентябрь 1997 г.
    * - - - - - - - - - - - - - - -  Java 1.1 октябрь 1997 г.
    *                                обновлена в июле 2000 г.
    * Проверяет номер чека на наличие
    * последовательно идущих >= 3 нулей
    * или >= 4 не нулей.
    * Цифры номера должны разделяться пробелами.
    *
    * Демонстрирует применение if-then-else и символов.
    */

   static final int noOfDigits = 10;

   boolean counterfeit = false;
   int countOfZeros = 0;
   int countOfNonzeros = 0;

   ChequeDetector () throws IOException {

      Stream in = new Stream (System.in);

      System.out.println("***** Проверка чеков *****");
      System.out.println("Введите номер чека из десяти цифр, "+
```

```
            "разделенных пробелами.");
        System.oun.ptintln("В конце ввода нажмите клавишу Enter.");

        char digit;

        for (int i = 0; i < noOfDigits; i++) {
           digit = in.readChar();
           if (digit == '0')
             recordZero();
           else
             recordNonzero();
        }
        if (counterfeit)
           System.out.print("\tПОДДЕЛКА");
        else
           System.out.print("\tДЕЙСТВИТЕЛЬНЫЙ ЧЕК");
        System.out.println();
     }

     void recordZero() {
        countOfZeros ++;
        countOfNonzeros = 0;
        if (countOfZeros == 3)
           counterfeit = true;
     }
     void recordNonzero() {
        countOfNonzeros ++;
        countOfZeros = 0;
        if (countOfNonzeros == 4)
           counterfeit = true;
     }
     public static void main(String[] args) throws IOException {
        new ChequeDetector();
     }

}
```

Тестирование. Номера чеков должны быть подобраны так, чтобы можно было проверить особые (желательно все) случаи, вызывающие опасения. Вот несколько примеров, в которых приведены вводимые данные и данные, полученные на выходе.

```
***** Проверка чеков *****
Введите номер чека из десяти цифр, разделенных пробелами.
В конце ввода нажмите клавишу Enter.
0 0 0 3 3 0 0 4 4 0
    ПОДДЕЛКА
***** Проверка чеков *****
Введите номер чека из десяти цифр, разделенных пробелами.
В конце ввода нажмите клавишу Enter.
0 0 3 3 3 0 0 3 3 3
    ДЕЙСТВИТЕЛЬНЫЙ ЧЕК
***** Проверка чеков *****
Введите номер чека из десяти цифр, разделенных пробелами.
В конце ввода нажмите клавишу Enter.
4 4 4 4 0 0 5 5 0 0
    ПОДДЕЛКА
```

Усовершенствование. Теперь мы рассмотрим, как избавиться от ограничения, связанного с употреблением пробелов в качестве разделителей. Вместо того, чтобы использовать метод `readChar`, мы считаем строку, преобразуем ее в массив и проверим символы по одному. Главный цикл изменится с:

```
char digit;

for (int i = 0; i < noOfDigits; i++) {
  digit = in.readChar();
  if (digit == '0')
    recordZero();
      else
  recordNonzero();
}
```

где вводимые данные — 1 0 0 6 7 0 2 2 1 5, на

```
String s = in.readString();
char digits [ ] = s.toCharArray();

for (int i = 0; i < digits.length; i++) {
  if (digits[i] == '0')
     recordZero();
  else
     recordNonzero();
}
```

с гораздо более традиционным представлением данных, таким как 1006702215.

7.2. Разделители и оболочки

В главе 4 отмечалось, что считывание чисел в Java было организовано несколько неуклюже. В качестве средства, позволяющего выполнять эти простые операции, был предложен класс `Stream`. В следующем разделе мы изучим структуру этого класса, но прежде остановимся на более важных вопросах, связанных с употреблением разделителей и оболочек.

Разделители

Итак, в Java в сфере ввода данных наблюдался ряд несовершенств. Пробелы эти были восполнены за счет класса `Stream`. Метод, применяемый в Java, состоит в следующем: строки считываются и с помощью очень эффективных средств разбиваются на отдельные участки — лексемы (tokens), которые впоследствии приводятся к требуемому типу. Лексемы представляют собой подстроки, разделенные такими символами, как запятая, пробел или табуляция. Посредством этого метода мы можем выбрать подстроки из вводимых данных и передать их процедурам класса-оболочки для обработки. Например, в следующем предложении имеется 10 лексем, которые соответствуют общеупотребительным словам.

```
Where e're you walk, cool gales shall fan the glade[17].
```

Класс StringTokenizer, который находится в java.util, предоставляет следующие методы:

Объявление методов класса StringTokenizer и доступ к ним

```
StringTokenizer (String s);
StringTokenizer (String s, String delimiters);
StringTokenizer (String s, String delimiters,
                 boolean returnasTokens);
String nextToken ();
String nextToken (String delimiter);
boolean hasMoreTokens ();
int countTokens ();
```

После ввода строки мы объявляем для нее разделитель и указываем ее в качестве параметра, как в следующем примере:

```
StringTokenizer T = new StringTokenizer (S);
```

Затем просматриваем лексемы, проверяя, является ли данная лексема последней:

```
while (T.hasMoreTokens()) {
  System.out.println(T.nextToken());
}
```

В результате ввода предыдущей конструкции может быть получен следующий результат:

```
Where
e're
you
walk,
cool
gales
shall
fan
the
glade[17]
```

Изменение комплекта разделителей

Набор предлагаемых по умолчанию разделителей (пробел, маркер конца строки и табуляция) можно изменить. Например, в процессе выполнения анализа программы может возникнуть необходимость использовать в качестве разделителей точку с запятой, скобки и т.д., которые в противном случае воспринимаются как часть лексем. Предположим, что в качестве данных мы имеем следующее выражение:

```
spaceAt = s.indexOf("  ",startingFrom);
```

При использовании разделителей, предлагаемых по умолчанию, фрагменты будут выглядеть следующим образом:

```
spaceAt
=
s.indexOf("  ",startingFrom);
```

Возможно, это не совсем то, что нам нужно. Если нам нужно больше деталей, мы получим лучший эффект, объявив:

```
StringTokenizer T = new StringTokenizer (s," .(,); =:", false);
```

Здесь увеличен набор разделителей, а соответствующие параметры исключены из размеченных участков. Результат будет следующим:

```
spaceAt
s
indexOf
"  "
startingFrom
```

что в большей степени соответствует нашим требованиям.

Оболочки

Одно из строгих правил языка Java гласит, что значения примитивных типов и объекты не должны смешиваться. В большинстве случаев это вызывает значительные неудобства, но иногда требуется направить стандартному пакету объект, а мы передаем ему, например, значение типа `int`. Чтобы сделать это, мы сначала помещаем значение в конструкцию, известную как **класс-оболочка**, вследствие чего оно становится объектом. Класс-оболочка обеспечивает доступ к значению и поддерживает разнообразные методы преобразования.

Пакет Java `lang` имеет классы-оболочки, ассоциированные с каждым из примитивных типов. Вот перечень этих классов: `Boolean`, `Character`, `Double`, `Float`, `Integer` и `Long` (обратите внимание на то, что имена классов начинаются с большой буквы). Как следует из имен, эти классы предоставляют версии соответствующих примитивных типов данных.

В качестве примера рассмотрим важнейшие методы класса `Integer`:

Преобразования между `Integer` и `int`

```
Integer (int value);       // конструктор

Integer valueOf (String s);

int         intValue ();
int         parseInt (String s)
String      toString (int i);
// Далее следуют 15 других методов класса Integer.
```

Конструктор принимает значение `int`, помещает его в оболочку и представляет как объект класса `Integer`.
Метод `valueOf` создает целочисленное значение на основе строки и помещает его в объект `Integer`.
Метод `parseInt` идентичен предыдущему с той разницей, что помещает результат в объект примитивного типа `int`.
Метод `intValue` извлекает из оболочки значение `int` (преобразует объект `Integer` в `int`).
Метод `toString` преобразует примитивное целое число в `String`.

Конструктор и методы экземпляра предоставляют возможность свободно перемещаться между типами и классами. Поэтому, если `i` — это `int`, то

```
Integer Iobj = new Integer (i);
```

делает из i объект. Чтобы получить int назад для вывода на печать, мы используем:

```
System.out.println(Iobj.intValue());
```

Остальные отобранные методы преобразовывают строки в числовые значения и наоборот. Другие классы-оболочки также имеют соответствующие методы (вспомните класс Character из раздела 5.4).

7.3. Структура класса Stream

Небольшой класс Stream использует разделители и оболочки. Методы, предоставляемые классом, уже указывались в главе 4, но мы повторим их здесь еще раз:

Класс Stream

```
Константы
- - - - -
READ = 0,
WRITE = 1;

Конструкторы
- - - - - - -
Stream (InputStream filename)
Stream (String filename, int how)

Ввод
- - -
int       readInt ()
double    readDouble ()
String    readString ()
char      readChar ()

Вывод
- - -
void println - для Objects, String, int, double, char
void print - для Objects, String, int, double, char
void close ()

Вывод -  методы класса
- - - - - - - - - - -
String format (int number, int align)
String format (double number, int align, int frac)
```

Первый конструктор используется, когда известен поток InputStream (например, при соединении с клавиатурой).
Второй конструктор применяется для соединения файлов. Параметр how позволяет указать, предназначен файл для чтения или для записи.
Четыре метода ввода работают с объектом Stream и включают в поток новый элемент, интерпретируя его в соответствии с заданным типом. Если элемент имеет некорректный тип, то метод обращается к другому элементу. Все элементы должны быть разделены пробелами или другими знаками пунктуации.
Методы println и print работают так же, как и при использовании объекта System.out.
Метод close должен вызываться для любого файла, открытого для записи. В этом случае его содержимое будет сохранено для последующего использования.
Методы форматирования являются методами класса и вызываются соответствующим образом — Stream.format. Подробно их работа описана ниже.

Класс `Stream` предназначен для создания экземпляров потоковых объектов. Будучи созданными, объекты `Stream` могут использовать методы ввода. Однако два метода `format` не зависят от объектов. Каждый из них представляет собой комплект методов, которые могут быть вызваны в любом порядке и на любой стадии для представления числа в виде строки с определенным количеством цифр. Предварительная инициализация не производится, поэтому методы должны инициализировать себя.

Алгоритмы

Главный алгоритм для каждого из методов считывания проиллюстрирован на примере считывания целого числа. Процедура может инициализировать себя, обнаружив нулевую лексему. Если такая имеется, то процедура считывает строку и инициирует разделитель. После этого она вызывает метод `nextToken` для получения лексемы и пытается преобразовать ее в число. Если в процессе выполнения указанных операций возникнут ошибки, отобразится сообщение об исключительной ситуации.

- Вызов метода `nextToken` может остаться невыполненным, если больше нет лексем. Следовательно, нужно попробовать считать следующую строку.
- Если подстрока, образующая лексему, не представляет число надлежащего вида, возможен сбой в процессе преобразования. В таком случае мы просто выводим сообщение (система генерирует объект-исключение) и повторяем попытку считывания.

При успешном выполнении обеих операций работа метода завершается. Вот почему данный метод является удачным примером использования сообщений об исключительных ситуациях. Соответствующий код приведен ниже:

```
public int readInt () throws IOException {
  if (T == null) refresh();
  while (true) {
    try {
      return Integer.parseInt(T.nextToken());
    }
    catch (NoSuchElementException e1) {
      refresh ();
    }
    catch (NumberFormatException e2) {
      System.out.println("Error in number, try again.");
    }
  }
}
```

Класс содержит интересные операторы в каждом методе числовых преобразований. При анализе введенного целого числа выполняется синтаксический разбор с помощью метода `parseInt` из класса-оболочки `Integer`:

```
return Integer.parseInt(T.nextToken());
```

Однако класс `Double` не имеет соответствующего метода для синтаксического разбора, в связи с чем преобразование усложняется:

```
return Double.valueOf(item.trim()).doubleValue();
```

Объект `item` — это возвращаемая строка. Сначала мы освободим ее от пробелов посредством метода `trim` класса `String`. Затем вызовем метод `Double.valueOf` для преобразования строки в объект класса `Double`. В завершение мы извлечем объект

из оболочки и преобразуем его в тип double. Очевидно, что эта операция стоит того, чтобы ее определили как типизированный метод!

Методы format используют пакет java.text, который рассматривается в следующем разделе. Рассмотрим полный текст класса Stream:

Класс Stream

```
package javagently;

import java.io.*;
import java.util.*;
import java.text.*;


public class Stream {

  /* Класс Stream.     Авторы - Дж. М. Бишоп и Б. Воролл. Написан в мае 2000 г.
   * ============      при содействии Дж. Каасболла.
   *                   Основан на классе Text, август  1996 г.
   *                   Авторы - Дж. М. Бишоп и А. Мулмен.
   *
   * Обеспечивает простой ввод с клавиатуры и файлов
   * и форматированный вывод на монитор и в файлы.
   *
   * Конструкторы
   * ------------
   * public Stream (InputStream in)
   * public Stream (String filename, int why)
   *
   * Ввод
   * -----
   * public int    readInt ()
   * public double readDouble ()
   * public String readString ()
   * public char   readChar ()
   *
   * Вывод
   * ------
   * public void println  - для Objects, String, int, double, char
   * public void print    - для Objects, String, int, double, char
   * public void close()
   *
   * Вывод — методы класса
   * ---------------------
   * public String format (int number, int align)
   * public String format (double number, int align, int frac)
   */


  private BufferedReader in;
  private PrintWriter out;
  private StringTokenizer T;
  private String S;
```

```
public static final int
    READ = 0,
    WRITE = 1;

public Stream(InputStream i) {
    in = open(i);
}
public Stream(String filename, int how)
      throws FileNotFoundException, IOException {
  switch(how) {
    case READ: in = open(filename); break;
    case WRITE: out = create(filename); break;
  }
}

private BufferedReader open(InputStream in) {
  return new BufferedReader(new InputStreamReader(in));
}

private BufferedReader open(String filename) throws FileNotFoundException {
  return new BufferedReader(new FileReader(filename));
}

private PrintWriter create(String filename) throws IOException {
  return new PrintWriter(new FileWriter(filename));
}

public String readLine () throws IOException {
  refresh();
  return S;
}

public int readInt () throws IOException {
  if (T==null) refresh();
  while (true) {
    try {
      return Integer.parseInt(T.nextToken());
    }
    catch (NoSuchElementException e1) {
      refresh ();
    }
    catch (NumberFormatException e2) {
      System.out.println("Error in number, try again.");
    }
  }
}

public char readChar () throws IOException {
  if (T==null) refresh();
  while (true) {
    try {
      return T.nextToken().trim().charAt(0);
    }
    catch (NoSuchElementException e1) {
```

```
        refresh ();
      }
    }
  }

  public double readDouble () throws IOException {
    if (T==null) refresh();
    while (true) {
      try {
        String item = T.nextToken();
        return Double.valueOf(item.trim()).doubleValue();
      }
      catch (NoSuchElementException e1) {
        refresh ();
      }
      catch (NumberFormatException e2) {
        System.out.println("Error in number, try again.");
      }
    }
  }

  public String readString () throws IOException {
    if (T==null) refresh ();
    while (true) {
      try {
        return T.nextToken();
      }
      catch (NoSuchElementException e1) {
        refresh ();
      }
    }
  }

  private void refresh () throws IOException {
    S = in.readLine ();
    if (S==null) throw new EOFException();
    T = new StringTokenizer (S);
  }

  private static DecimalFormat N = new DecimalFormat();
  private static final String spaces = "                    ";

  public static String format(double number, int align, int frac) {
    N.setGroupingUsed(false);
    N.setMaximumFractionDigits(frac);
    N.setMinimumFractionDigits(frac);
    String num = N.format(number);
    if (num.length() < align)
       num = spaces.substring(0,align-num.length()) + num;
    return num;
  }

  public static String format(int number, int align) {
    N.setGroupingUsed(false);
```

```
    N.setMaximumFractionDigits(0);
    String num = N.format(number);
    if (num.length() < align)
       num = spaces.substring(0,align-num.length()) + num;
    return num;
}

    public void println(Object s) {
        out.println(String.valueOf(s));
        out.flush();
    }

    public void println(String s) {
        out.println(s);
        out.flush();
    }

    public void println(int s) {
        out.println(s);
        out.flush();
    }

    public void println(double s) {
        out.println(s);
        out.flush();
    }

    public void println(char s) {
        out.println(s);
        out.flush();
    }

    public void print(Object s) {
        out.print(String.valueOf(s));
        out.flush();
    }
    public void print(String s) {
        out.print(s);
        out.flush();
    }

    public void print(int s) {
        out.print(s);
        out.flush();
    }

    public void print(double s) {
        out.print(s);
        out.flush();
    }

    public void print(char s) {
        out.print(s);
        out.flush();
```

```
    }

    public void close() throws IOException {
        if (out != null)
            out.close();
        if (in != null)
            in.close();
    }

    public void flush() {
        out.flush();
    }


}
```

На рис. 7.3 структура класса `Stream` показана в виде диаграммы. Четко обозначены те части, которые являются членами класса, и те, которые создаются как экземпляры. Для каждого члена указан режим доступа (public или private) с помощью знаков + и −. Класс `Stream` предоставляет для каждого объекта пять методов экземпляра и два метода класса, которые представлены как открытые (public). Методы `open`, `create` и `refresh` используются открытыми методами и являются закрытыми (private). Аналогично: параметры конструктора копируются в закрытые переменные ввода-вывода, а две закрытые переменные, `T` и `S`, применяются для разделения.

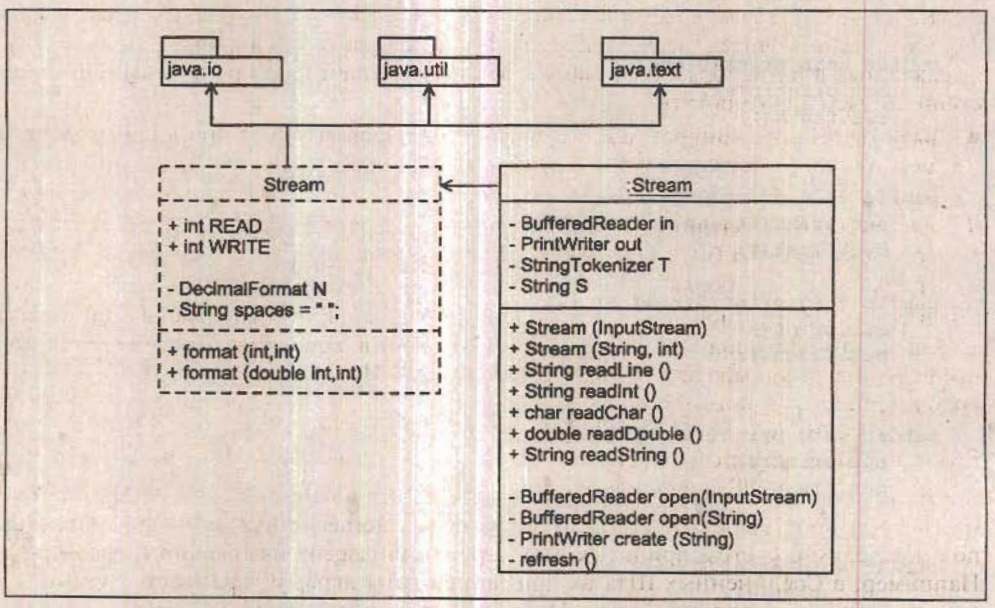


Рис. 7.3. Структура класса `Stream` и типичного объекта `Stream`

7.4. Форматирование чисел

При рассмотрении методов `format` класса `Stream` мы обращались к характеристикам пакета `java.text`. Настоящий раздел посвящен описанию этих встроенных способов записи чисел.

Функции форматирования

Java отличается от других языков тем, что форматирование чисел задается не в операторах вывода на печать, а в отдельном объекте. Затем в операторе вывода на печать соответствующий объект, определяющий формат, присоединяется к элементу, который должен быть выведен. В любой момент времени в программе могут одновременно существовать несколько объектов форматирования, доступные для неоднократного применения.

В пакете `java.text` для форматирования предназначены такие классы:

```
DateFormat
SimpleDateFormat
NumberFormat
DecimalFormat
MessageFormat
```

Обзор методов, которые служат для форматирования дат, вы найдете в следующем разделе, а на данном этапе мы изучим те, что предназначены для форматирования чисел.

Класс `NumberFormat` и его подкласс `DecimalFormat` предоставляют средства для форматирования чисел всех видов, включая процентные значения и значения, выраженные в разнообразных валютах. Мы разработаем формат, который впоследствии сможем адаптировать:

- в соответствии с конкретными требованиями к форматированию, такими, например, как количество разрядов в дробной части вещественного (действительного) числа;
- для различных регионов, в которых действуют разные стандарты относительно того, как должны быть представлены десятичные доли чисел, символы валюты, группы разрядов больших чисел и т.д.

Вторая из перечисленных возможностей является очень важной. Для языка определенного региона выделяется класс, который в дальнейшем применяется при вычислениях и обработке во всех точках мира. Итак, наша следующая тема — локализация с помощью Java.

Региональные настройки компьютера

Язык Java без преувеличения можно назвать интернациональным языком. Программы, распространяемые по всему миру через Internet, могут быть предоставлены пользователям с учетом принятых в их стране стандартов изложения информации. Например, в Соединенных Штатах при записи даты первым указывается месяц, а в других странах — год или день. Пользователям операционных систем известны региональные настройки, которые часто по умолчанию являются следующими: язык — English, страна — United States. При выборе другой страны остальные параметры (символ валюты, формат даты) изменяются автоматически.

Java предоставляет в пакете `java.util` класс, который позволяет программе исследовать и вывести настройки компьютера. Класс этот называется `Locale`. Некоторые из его средств описаны ниже.

Класс Locale

```
Locale (String language, String country, String variant);
Locale (String language, String country);
// Константы
   Locale CANADA;
   Locale CANADA_FRENCH;
   Locale CHINA;
   Locale CHINESE;
   Locale ENGLISH;
   Locale FRANCE;
   Locale FRENCH;
   Locale GERMAN;
   Locale GERMANY;
   Locale ITALIAN;
   Locale ITALY;
   Locale JAPAN;
   Locale JAPANESE;
   Locale KOREA;
   Locale KOREAN;
   Locale PRC;
   Locale SIMPLIFIED_CHINESE;
   Locale TAIWAN;
   Locale TRADITIONAL_CHINESE;
   Locale UK;
   Locale US;

// методы класса
void Locale  getDefault();
void         setDefault (Locale newLocale);

// методы экземпляра
String getCountry();
String getDisplayCountry(Locale inLocale);
String getDisplayLanguage(Locale inLocale);
String getDisplayLanguage();
String getLanguage();
String getName();
// и 10 других
```

Вы, наверное, заметили, что описание класса `Locale` начинается с двух конструкторов. Мы можем создать новый объект, содержащий региональные стандарты, при условии, что знаем страну и государственный язык. Кроме того, есть два метода класса, один из которых предоставляет следующую возможность: объект с региональными стандартами может быть создан с учетом установок по умолчанию системы. Второй метод класса используется в стандартных объявлениях объектов для создания одного из объектов класса `Locale` в виде константы. Константы предусмотрены не для всех регионов, но из системы можно получить дополнительную информацию, как показано в следующем примере.

Пример 7.3. Где я?

Задача. Требуется выяснить, какой регион и какой язык заданы на компьютере, где выполняется программа.

Решение. Для решения поставленной задачи мы вызовем класс `Locale`. Затем мы расширим спецификацию и создадим другой объект класса `Locale`, чтобы определить, как это отразится на получаемых данных.

Программа. Программа ищет объект класса `Locale`, соответствующий установкам на данной машине, а затем создает объект класса `Locale` для Германии.

```
import java.util.*;

class WhereAmI {

  /* Программа, опрашивающая
  /* Региональные стандарты.   Автор - Дж. М. Бишоп, декабрь 1997 г.
  /* - - - - - - - - - - - - -
  /*                           Обновлена в июле 2000 г.
  /* Демонстрирует использование классов из пакетов Java
  /* и средств для изменения региональных стандартов.
  */

  public static void main (String [ ] args) {
    new WhereAmI ();
  }

  WhereAmI () {
    Locale here = Locale.getDefault();

    System.out.println("Мои региональные стандарты - " + here);
    System.out.println("Страна: " + here.getCountry());
    System.out.println("Язык: " + here.getLanguage());
    System.out.println("Страна: " + here.getDisplayCountry());
    System.out.println("Язык: " + here.getDisplayLanguage());
    System.out.println();

    Locale there = new Locale("GERMAN","GERMANY");
    there.setDefault(Locale.GERMANY);
    System.out.println("Новые региональные установки - " + there);
    System.out.println("Страна: " + there.getCountry());
    System.out.println("Язык: " + there.getLanguage());
    System.out.println("Страна: " + there.getDisplayCountry());
    System.out.println("Язык: " + there.getDisplayLanguage());

  }
}
```

Тестирование. Вот какой результат был получен после выполнения программы на моем компьютере с установленной операционной системой Windows:

```
Мои региональные стандарты - en_SOUTH AFRICA
Страна: SOUTH AFRICA
Язык: en
Страна: United States
Язык: English
```

```
Новые региональные установки - german_GERMANY
Страна: GERMANY
Язык: german
Страна: Deutschland
Язык: Deutsch
```

Итак, компьютер обнаружил, что он находится в Южно-Африканской Республике (South Africa), а государственным языком является английский (en). Однако при выводе на экран используются установки, рассчитанные на United States, поскольку Южно-Африканская Республика не учтена в региональных стандартах компьютера. Если мы затем создадим новый объект класса `Locale` и объявим его как `GERMANY`, то увидим, что страна и язык практически полностью локализованы. Интересный результат я получила, когда запустила программу на моем компьютере iMac:

```
Мои региональные стандарты - en_US
Страна: US
Язык: en
Страна: United States
Язык: English

Новые региональные установки - german_GERMANY
Страна: GERMANY
Язык: german
Страна: GERMANY
Язык: german
```

Эти данные свидетельствуют о том, что установки компьютера некорректны и/или MacOS не поддерживает региональные стандарты в такой степени, как Windows.

Класс NumberFormat

Класс `NumberFormat` является абстрактным. Следовательно, мы не сможем создать объекты, определив их, но зато у нас есть возможность получить экземпляры объекта непосредственно из класса (подробно абстрактные классы описаны в главе 9). Методы класса, позволяющие получить объект форматирования, перечислены ниже:

Создание объектов для форматирования чисел

`NumberFormat getInstance ()`
`NumberFormat getCurrencyInstance ()`
`NumberFormat getNumberInstance ()`
`NumberFormat getPercentInstance ()`

Каждый из них возвращает объект, различные методы которого решают ту или иную задачу форматирования.

Каждый из четырех методов имеет версии, соответствующие региональным стандартам, отличным от заданных по умолчанию. Региональные стандарты представляют собой совокупность соглашений, касающихся языка, денежных единиц, времени и других параметров, свойственных определенному региону. Типичными объектами класса `Locale` являются `Locale.US` и `Locale.GERMANY`.

Чтобы использовать объект форматирования, необходимо задействовать методы экземпляра, перечисленные ниже. Поскольку все остальные числовые типы могут быть представлены как `long` или `double`, нужны только два метода.

Использование объектов для форматирования чисел
`String format (double number);` `String format (long number);`
При использовании объектов для форматирования чисел эти методы позволяют выполнять более компактное и разнообразное форматирование, чем обеспечивают операторы `println`.

Сравним, как выводятся числа по умолчанию и в случае применения объекта форматирования с заданными по умолчанию методами:

```
NumberFormat N = NumberFormat.getInstance ();

System.out.println(Math.PI + " " + 10000);
System.out.println(N.format(Math.PI) + " " + N.format(10000);
```

Вывод будет таким:

```
3.141592653589793 10000
3.142 10,000
```

Следовательно, объект, форматирующий числа, осуществляет структурирование чисел, придавая данным на выходе удобочитаемый вид. Правда, запись чисел с запятыми не получила широкого распространения и используется только в финансовой сфере ряда стран, например в США. Поэтому в большинстве случаев потребуется привести формат вывода в соответствие с нормой. Для этого мы будем использовать следующие методы экземпляра:

Настройка объектов форматирования чисел

`void setMaximumIntegerDigits (int newvalue);`
`void setMinimumIntegerDigits (int newvalue);`
`void setMaximumFractionDigits (int newvalue);`
`void setMinimumFractionDigits (int newvalue);`
`void setGroupingUsed (boolean newvalue);`

В результате применения методов `setMaximum` значения усекаются. Методы `setMinimum` добавляют пробелы для целой части числа или нули для дробной части числа. Методы группирования определяют, должны ли тысячи быть разделены запятой, пробелом или другим символом.

Для каждого метода `set` имеется соответствующий метод `get`, который может при необходимости предоставить текущие установки. Использование кратких идентификаторов — удачное решение, поскольку употребление слова `format`, являющегося составной частью имени метода, приведет к чрезмерному увеличению длины операторов вывода.

Форматирование действительных чисел

Посмотрим, какой эффект дает изменение различных установок для действительных чисел. С этой целью будет выполнен анализ метода `Stream.format`:

```
private static String format (double number, int align, int frac) {
    N.setGroupingUsed(false);
    N.setMaximumFractionDigits(frac);
    N.setMinimumFractionDigits(frac);
    String num = N.format(number);
```

```
    if (num.length() < align)
      num = spaces.substring(0,align-num.length()) + num;
    return num;
  }
```

Три вызова `set` задают количество цифр в дробной части числа и отменяют группирование тысяч. Если установить минимум, а не максимум, в качестве заполнителей будут использоваться пробелы, хотя обычно в дробной части пишутся нули. Затем число форматируется и помещается в локальную строку, поскольку должна быть сделана одна настройка. В объекте, форматирующем число, по умолчанию задано, что минимальное количество цифр в целой части числа равно 1, поэтому при выводе мы всегда будем иметь цифру перед дробной частью (например, 0.567, а не .567). Если же мы зададим, что максимальное значение равно (`align-frac`), все числа будут иметь фиксированную длину, при этом они будут начинаться с нулей, а не с пробелов. Вот почему вместо указанной операции мы добавляем дополнительные пробелы с помощью средств обработки строки.

Использование максимального количества цифр в целой части числа имеет еще один недостаток: старшие разряды отсекаются. Так, если задано, что максимальное количество цифр равно 3, число 1000 будет выведено как 0. Однако в целом класс `Stream` является полезным инструментом.

В классе `DecimalFormat` имеются средства для формирования модели, которая может использоваться при форматировании. Однако, создавая такую модель, как `###0.###`, надлежит учитывать, что при изменении количества символов `#` потребуются функции обработки строки. Мы вернемся к разговору о моделях во время настройки формата для вывода даты.

Денежные единицы

Мы можем задать вывод любой денежной единицы в удобной для нас форме. Для представления числа как денежной единицы предназначены следующие методы:

```
NumberFormat C = NumberFormat.getCurrencyInstance();
System.out.println (C.format(10000));
```

Они возвращают такой результат:

```
$10,000.0
```

Пример 7.4. Таблица конвертирования валют

Задача. Требуется исследовать, какими будут результаты при поочередном обращении к разным объектам класса `Locale` для вывода денежных единиц.

Решение. Мы запустим небольшую программу, которая извлекает курсы обмена для пяти валют и выводит таблицу конвертирования в вымышленную денежную единицу грац.

Программа. Программа очень простая. Объект для форматирования валюты создается для каждого из используемых объектов класса `Locale`. Указываются курсы, а затем выводятся суммы, полученные в результате конвертирования.

```
import java.text.*;
import javagently.*;
import java.util.*;
import java.io.*;
```

```
class Currency {

  public static void main (String args []) throws IOException {
    Stream in = new Stream ("rates.dat",Stream.READ);

    System.out.println("Таблица соотношения валют");
    System.out.println("== = = = = = = = = = = = ");
    System.out.println();
    System.out.println("Курсы обмена:");
    System.out.println("graz\tдоллары\tфунты\tйены\tмарки\tфранки");
    double d = in.readDouble();
    double p = in.readDouble();
    double y = in.readDouble();
    double m = in.readDouble();
    double f = in.readDouble();
    System.out.println("1\t"+d+"\t"+p+"\t"+y+"\t"+m+"\t"+f);
    System.out.println();

    NumberFormat Nd =
          NumberFormat.getCurrencyInstance(Locale.US);
    NumberFormat Np =
          NumberFormat.getCurrencyInstance(Locale.UK);
    NumberFormat Ny =
          NumberFormat.getCurrencyInstance(Locale.JAPAN);
    NumberFormat Nm =
          NumberFormat.getCurrencyInstance(Locale.GERMANY);
    NumberFormat Nf =
          NumberFormat.getCurrencyInstance(Locale.FRANCE);
    for (int graz = 1000; graz < 10000; graz+=1000)
      System.out.println('G'+Text.writeInt(graz,3) + '\t' +
        Nd.format(graz/d) + '\t' +
        Np.format(graz/p) + '\t' +
        Ny.format(graz/y) + '\t' +
        Nm.format(graz/m) + '\t' +
        Nf.format(graz/f));
  }
}
```

Тестирование. Результат выполнения программы таков:

```
Таблица соотношения валют
= = = = = = = = = = = = = =
 Курсы обмена:
Graz    доллары   фунты    йены       марки        франки
1       $4.8845   8.047    0.0378     2.7361       0.8174

G1000   $204.73   124.27   26,455.03  365,48 DM    1 223,39 F
G2000   $409.46   248.54   52,910.05  730,97 DM    2 446,78 F
G3000   $614.19   372.81   79,365.08  1.096,45 DM  3 670,17 F
G4000   $1,023.65 497.08   105,820.11 1.461,93 DM  4 893,56 F
G5000   $1,023.65 621.35   132,275.13 1.827,42 DM  6 116,96 F
G6000   $1,228.38 745.62   158,730.16 2.192,90 DM  7 340,35 F
G7000   $1,433.10 869.89   185,185.19 2.558,39 DM  8 563,74 F
G8000   $1,637.83 994.16   211,640.21 2.923,87 DM  9 787,13 F
G9000   $1,842.56 1,118.43 238,095.24 3.289,35 DM  11 010,52 F
```

Вывод довольно интересен, особенно если учесть, что почти все было сделано без нашего участия.

Ввод форматированных данных

В пакете `java.text` предоставляются средства для ввода форматированных данных в каждом из классов. При работе с форматированными данными не требуется следить за употреблением пробелов в качестве заполнителей и т.д. Эти значения необходимо вводить в специально отведенных местах строки. Они могут даже примыкать друг к другу, как в следующем примере:

```
76 89123 45 10 6100
```

Если эти значения представить в формате с максимальным и минимальным количеством цифр равным 3, результат получится таким: 768, 912, 345, 106 и 100.

Главной функцией при обработке вводимых данных в Java является синтаксический анализ, который аналогичен форматированию выводимых данных. Однако в настоящее время данные редко выводятся без разделителей. Недостаток синтаксического анализа с помощью объектов форматирования состоит в том, что результаты выводятся как объекты, поэтому приходится выполнять преобразование в примитивные типы.

7.5. Даты, календари и время

Класс `Date`, который входит в состав пакета `java.util`, представляет даты и время в виде, независимом от системных установок. Ниже приведена его полная спецификация:

Класс `Date`

```
Date ();
Date (long date);
boolean after (Date when);
boolean before (Date when);
boolean compareTo (Date d);
boolean equals (Object obj);
long    getTime ();
long    setTime (long time);
String  toString ();
```

Описание класса `Date` начинается с двух конструкторов. Первый, который не имеет параметров, помещает в новый объект текущие значения даты и времени, которые установлены на компьютере. Поэтому при использовании следующей конструкции:

```
Date today = new Date ();
```

будет создан объект, содержащий сегодняшнюю дату и системное время. Второй конструктор создает дату, основываясь на количестве миллисекунд, прошедших с некоторой произвольной даты. В классе `Date` нет более легкого пути для получения конкретной даты, для этой цели нужен другой класс `Calendar`.

За конструкторами следуют методы. На данном этапе интерес для нас представляют `getTime` и `toString`. Имея объект класса `Date`, мы можем вывести его с помощью `println` следующим образом:

```
System.out.println(today);
```

Объект `today` будет преобразован в строку, после чего произойдет его вывод. Преобразование выполняет метод `toString`, который должен присутствовать во всех классах, где следует обеспечить возможность вывода объектов. В соответствии с форматом преобразования дата после вывода всегда будет иметь следующий вид:

```
Day Month Date Hours:Mins:Secs Zone Year
Tue Jul 04 17:11:54 EST 2000
```

Если часовой пояс региона, в котором находится компьютер, не принадлежит к числу распознаваемых Java, дается значение смещения от GMT (Greenwich Mean Time — время по Гринвичу):

```
Tue Jul 04 17:11:54 GMT+00:00 2000
```

В пакете `java.text` для дат предусмотрены и другие форматы.

Изначально в Java 1.0, класс `Date` был значительно больше и включал также методы получения и назначения частей даты. В настоящее время эти функции переданы классу `Calendar`. Ниже перечислены те его части, которые понадобятся для выполнения следующего примера:

- константы, соответствующие различным полям (`YEAR` или `MINUTE`);
- константы для дней и месяцев (`MONDAY` или `DECEMBER`);
- метод класса, создающий объект, т.е. `getInstance`;
- методы экземпляра для получения и задания частей даты (`get` и `set`).

Методы `get` и `set` требуют, чтобы части, которые соответствуют полям, были специфицированы, например:

```
int y = get(Calendar.YEAR);
set (Calendar.YEAR, 2001);
```

Методу `set` могут быть переданы три значения — год, месяц и день, но лучше использовать имена-константы, соответствующие месяцам (например, `Calendar.May`, а не 5).

Пример 7.5. Дни для покупок перед Рождеством

Задача. Требуется подсчитать количество дней для покупок перед Рождеством, исходя из того, что уже декабрь.

Алгоритм. Алгоритм очень прост, поскольку нужно только вычесть значения двух объектов `Date`. Для разработки алгоритма, учитывающего другие месяцы года, потребуется больше навыков.

Программа

```
import java.util.*;

public class Shopping {
   /* Программа Shopping.  Автор - Дж. М. Бишоп, декабрь 1996 г.
    * - - - - - - - - - -   Java 1.1 - Дж. М. Бишоп, декабрь 1997 г.
    *                       Обновлена в июле 2000 г.
    * Подсчитывает количество дней для покупок перед
    * Рождеством в декабре.
    */
  public static void main(String[] args) {
    new Shopping ();
  }
```

```
Shopping () {
  Calendar today = Calendar.getInstance() ;
  Calendar christmas = Calendar.getInstance() ;

  christmas.set(
    today.get(Calendar.YEAR), Calendar.DECEMBER, 25);

  System.out.println("Программа работает только в декабре!");
  System.out.println("Рождество наступит в "+
      christmas.getTime());
  System.out.println("Сегодня " + today.getTime());

  System.out.println("Осталось "
    + (christmas.get(Calendar.DATE)-
       today.get(Calendar.DATE))
    + " дней для покупок перед Рождеством.");
}
}
```

В строках 1 и 2 метода `main` две даты объявлены как объекты `Calendar`, для создания которых использовался метод класса `getInstance`. В строках 3 и 4 значение текущего года для объекта `christmas` указывается с помощью объекта `today`. Это делает программу независимой от года использования.

В строках 6, 7 и 8 задан вывод двух дат, а в строках 10 и 11 осуществляется вычитание числовых значений дней для заключительного сообщения. Обратите внимание на то, что нам пришлось поместить выражение, вычисляющее разность, в скобки, благодаря чему мы получаем целое число, которое преобразуется в строку для вывода. Если этого не сделать, компилятор выдаст сообщение об ошибке.

Тестирование. Вот какой результат был получен в результате прогона программы:

```
Программа работает только в декабре!
Рождество наступит в Mon Dec 25 10:17:01 GMT+02:00 2000
Сегодня Tue Dec 05 10:17:01 GMT+03:00 2000
Осталось 20 дней для покупок перед Рождеством.
```

Форматирование даты

Форматирование дат осуществляется таким же способом, как форматирование чисел и денежных значений. Используя класс `DateFormat`, который находится в пакете `java.text`, можно создать форматирующий объект, имеющий определенный стиль. Возможны такие варианты: `DEFAULT`, `FULL`, `LONG`, `MEDIUM` и `SHORT`. Поэтому вместо простого вывода

```
System.out.println("Сегодня "+ today.getTime());
```

который использовался в предыдущей программе, мы можем указать более конкретно:

```
DateFormat D = DateFormat.getDateInstance(DateFormat.FULL);
System.out.println("Сегодня "+ D.format(today));
```

В результате будет выведена дата в более легком для восприятия формате:

```
Tuesday, July 4, 2000
```

В следующем примере исследуются различные варианты форматирования. Мы можем подобрать объект для форматирования значений времени и выводить их различными способами. В этом случае будет использоваться метод `getTimeInstance`.

Синтаксический анализ даты

В классе `DateFormat` имеется метод получения даты из строки. Он предназначен для считывания дат. Например, если объявить обычный объект для форматирования даты как:

```
DateFormat DF = DateFormat.getDateInstance();
```

и взять дату

```
15-Jul-2000
```

мы сможем прочитать ее с помощью следующей конструкции:

```
Date D = DF.parse(in.readString());
```

В случае применения метода `parse` может возникнуть исключительная ситуация. По этой причине рекомендуется использовать метод `catch` или употреблять пару `throws IOException - catch` во всех методах, ассоциированных с вызовом. Как обрабатывается исключительная ситуация, показано в следующей программе:

```
import java.text.*;
import javagently.*;
import java.util.*;
import java.io.*;

class TestDates {
  public static void main(String args []) throws IOException {
    new TestDates ();
  }

  TestDates() throws IOException {
    System.out.println("Введите дату по шаблону – dd.mm.yyyy");
    System.out.println();
    Stream in = new Stream(System.in);
    String s = in.readString();
    System.out.println("Дана строка: "+s);
    DateFormat DF = DateFormat.getDateInstance();
    Date d = new Date();
    try {
       d = DF.parse(s);
       System.out.println("Новый формат даты: "+d);
    } catch (ParseException e) {
       System.out.println("Ошибка в дате "+s);
    }
  }
}
```

```
Введите дату по шаблону – dd.mm.yyyy

15.07.2000
Дана строка: 15.07.2000
Новый формат даты: Sat Jul 15 00:00:00 GMT+03:00 2000

Введите дату по шаблону – dd.mm.yyyy

15.Mei.2000
```

```
Дана строка: 15.Mei.2000
Ошибка в дате 15.Mei.2000
```

Представлен простой формат вывода даты, применяемый по умолчанию. Конечно, могут быть заданы другие форматы за счет изменения соответствующего форматирующего объекта, что отражено в следующем примере.

Пример 7.6. Форматирование и синтаксический анализ дат

Задача. Необходимо продемонстрировать различные варианты ввода и вывода дат в Java.

Решение. Мы разработаем тестирующую программу, которая продемонстрирует специфический вывод на печать. Будут затронуты также некоторые интересные аспекты форматирования дат в Java, а именно:

- форматы дат могут быть переданы в качестве параметров;
- метод синтаксического анализа выдает сообщение об исключительной ситуации `ParseException`, в котором указывается, что дата подана в неправильной форме.

Программа. Обратите внимание на то, что при написании программы использовался объект класса `Locale` с региональными стандартами Великобритании (`Locale.UK`), вследствие чего день будет указан перед месяцем. Эту операцию легко изменить. По умолчанию задан объект с региональными стандартами США (`Locale.US`), в случае применения которого месяц указывается перед днем. Из примера 7.7 вы сможете получить больше информации о датах и объектах класса `Locale`.

```
import java.text.*;
import javagently.*;
import java.util.*;
import java.io.*;

class CustomDates {

   /* Программа для проверки
    * форматирования даты.      Автор - Дж. М. Бишоп, декабрь 1997 г.
    * - - - - - - - - - - - -   Java 1.1
    *                           Java 2 - июль 2000 г.
    *
    * Выводит даты в различных форматах,
    * обеспечивает ввод даты по заданному шаблону.
    */

   Stream in = new Stream (System.in);
   Date d = new Date ();
   Date my = new Date();

   void echoDate(String style, DateFormat Din, DateFormat Dout)
             throws IOException {
     System.out.println(style+"\t"+Din.format(d));
     String s = in.readLine();
     System.out.println("\t\t\t"+s);
     try {
       my = Din.parse(s);
     }
     catch (ParseException e) {
```

```
            System.out.println("Неправильная дата "+s);
        }
        System.out.println("\t\t\t"+Dout.format(my));
    }

    CustomDates() throws IOException {

        Locale.setDefault(Locale.UK);
        DateFormat DS = DateFormat.getInstance(DateFormat.SHORT);
        DateFormat DM = DateFormat.getInstance(DateFormat.MEDIUM);
        DateFormat DL = DateFormat.getInstance(DateFormat.LONG);
        DateFormat DF = DateFormat.getInstance(DateFormat.FULL);
        DateFormat DD = DateFormat.getInstance(DateFormat.DEFAULT);

        echoDate("SHORT",DS,DF);
        echoDate("MEDIUM",DM,DF);
        echoDate("LONG",DL,DF);
        echoDate("FULL",DF,DF);
        echoDate("DEFAULT",DD,DF);
    }

    public static void main(String args []) throws IOException {
        new CustomDates ();
    }
}
```

Тестирование

```
SHORT       04/07/00
30/04/00
            30/04/00
            30 April 2000
MEDIUM      04-Jul-00
15-Oct-00
            15-Oct-00
            15 October 2000
LONG 04 July 2000
25 September 2000
            25 September 2000
            25 September 2000
FULL 04 July 2000
7 January 2000
            7 January 2000
            07 January 2000
DEFAULT     04-Jul-00
Jul-04-00
            Jul-04-00
Неправильная дата Jul-04-00
            07 January 2000
```

Анализируя выведенные на печать данные, мы отметили два важных момента.

1. Кажется, Java тоже подвержена проблеме 2000-го года! Это не проявляется при *использовании стиля форматирования* даты LONG и FULL, в которых предусмотрен ввод четырехзначного значения года.
2. Согласно стилю FULL следует указывать также день (например, Tuesday).

Часовые пояса

Java предоставляет возможность определения часового пояса для компьютера, на котором выполняется программа. Зная часовой пояс, можно установить код пояса и смещение относительно Всемирного времени* (UTC), а также получить информацию о переходе на летнее время и наоборот.

Пример 7.7. Тестирование часовых поясов

Следующая программа позволяет получить информацию о часовом поясе того региона, в котором находится компьютер.

```
import java.util.*;

class WorldTime {
   /* Программа, определяющая часовые пояса.
    * Автор - Дж. М. Бишоп, декабрь 1997 г.
    * - - - - - - - - - - - - - - - Java 1.1
    *                               Обновлена в июле 2000 г.
    * Использует библиотеки Java для вывода значений
    * времени любого часового пояса.
    */
   public static void main(String [ ] args) {
     new WorldTime ();
   }

   WorldTime() {
     TimeZone here = TimeZone.getDefault();
     Date today = new Date();
     System.out.println(today);
     System.out.println("Мы находимся в "+here.getID() +
                 " часовом поясе ");
     System.out.println("со смещением в "+here.getRawOffset()/3600000 +
                 " часа от UTC");
     System.out.println("Здесь задается летнее время, "+
                 here.useDaylightTime());
     System.out.println("а сейчас часы переведены на зимнее время "
               + here.inDaylightTime(today));
   }
}
```

Тестирование

```
Tue Nov 20 15:36:19 IST 2001
Мы находимся в Asia/Jerusalem часовом поясе
со смещением в 2 часа от UTC
Здесь задается летнее время true,
а сейчас часы переведены на зимнее время true
```

Интересно, что произойдет, если изменить ваш часовой пояс после выполнения программы `WorldTime` и запустить ее вновь?

---

* Всемирным считается время на долготе 00 (Гринвичский меридиан). Для его обозначения употребляется аббревиатура GMT (Greenwich Mean Time — время по Гринвичу).

Форматирование локализованной даты

Объекты класса `Locale` обеспечивают вывод дат на различных языках и в разных форматах. Если используемый язык не известен классу `Locale` (полный список объектов этого класса опубликован в таблице "Класс `Locale`"), вы можете создать собственный объект этого класса.

Пример 7.8. Многоязычные даты

Задача. Цель данной задачи — продемонстрировать на практике возможности Java по выводу дат на разных языках.

Решение. Для решения задачи мы воспользуемся классами `DateFormat` и `Locale`.

Алгоритм. Вывод дат может быть выполнен с помощью метода `printDates`:

```
void printDates (Locale L) {

  System.out.println("Мы находимся в " + L.getCountry() +
     " и разговариваем на языке " + L.getLanguage());
  DateFormat D = DateFormat.getDateInstance();
  System.out.println(D.format(today));

  D = DateFormat.getDateInstance(DateFormat.FULL);
  System.out.println(D.format(today));
  System.out.println();
}
```

В результате выполнения этого метода будут выведены название страны, язык и дата в формате, установленном по умолчанию (краткое представление), а также в полной форме (формат `FULL`), но без значения времени. Вызов этого метода осуществляется следующим образом:

```
Locale.setDefault(Locale.GERMANY);
printDates(Locale.getDefault());
```

Другими словами, объект класса `Locale`, заданный по умолчанию, переопределяется и передается методу `printDates`. В результате мы должны получить:

```
Мы находимся в DE и разговариваем на языке de
14.12.2000
Donnerstag, 14. Dezember 2000
```

Программа. Программа включает измененные объекты класса `Locale`.

```
import java.util.*;
import java.io.*;
import javagently.*;
import java.text.*;

class VaryDates {

   /* Программа "Изменяющиеся даты". Автор - Дж. М. Бишоп, декабрь 1997 г.
    * - - - - - - - - - - - - - -       Java 1.1
    *                                   Обновлена в июле 2000 г.
    * Использует библиотеки Java для вывода дат
    * в формате различных объектов класса Locale.
    */
   Date today = new Date();
```

```
  void printDates (Locale L) {

    System.out.println("Мы находимся в " + L.getCountry() +
         " и разговариваем на языке " + L.getLanguage());
    DateFormat D = DateFormat.getDateInstance();
    System.out.println(D.format(today));

    D = DateFormat.getDateInstance(DateFormat.FULL);
    System.out.println(D.format(today));
    System.out.println();
  }

  VaryDates () {
    System.out.println(today);

    Locale genuine = Locale.getDefault();
    printDates(genuine);

    Locale.setDefault(Locale.GERMANY);
    printDates(Locale.getDefault());

    Locale.setDefault(Locale.UK);
    printDates(Locale.getDefault());

    Locale.setDefault(Locale.FRANCE);
    printDates(Locale.getDefault());

    Locale.setDefault(Locale.US);
    printDates(Locale.getDefault());

    Locale.setDefault(Locale.JAPAN);
    printDates(Locale.getDefault());
  }

  public static void main (String  [ ] args) throws IOException {
    new VaryDates ();
  }

}
```

Тестирование. Ниже вашему вниманию предлагается результат выполнения программы.

```
Tue Nov 20 16:01:59 IST 2001
Мы находимся в RU и разговариваем на языке ru
20.11.2001
20 Ноябрь 2001

Мы находимся в DE и разговариваем на языке de
20.11.2001
Dienstag, 20. November 2001

Мы находимся в GB и разговариваем на языке en
20-Nov-01
20 November 2001
```

```
Мы находимся в FR и разговариваем на языке fr
20 nov. 01
mardi 20 november 2001

Мы находимся в US и разговариваем на языке en
Nov 20, 2001
 Tuesday, November 20, 2001

Мы находимся в JP и разговариваем на языке ja
2001/11/21
2001?11?21?
```

Интересен следующий факт. Если задействовать объект класса `Locale` с региональными стандартами Японии, в которой на настоящий момент уже наступили новые сутки, дата изменится и в ней будет указано 21 ноября. Java не предоставляет средств для воспроизведения японских иероглифов, по этой причине дата выводится в виде числа.

Расширение. Мы можем расширить набор региональных стандартов, используемых на компьютере. С этой целью объявим новый объект класса `Locale`:

```
Locale Afr = new Locale("Afrikaans","Suid-Afrika");
```

Затем мы задействуем другой класс, `DateFormatSymbols`, в котором назначим названия для дней и месяцев на новом языке. Например, месяцы будут выглядеть так:

```
String AfrMonths [] = {"Januarie", "Februarie", "Maart", "April",
       "Mei", "Junie", "Julie", "Augustus", 'September",
      "Oktober", "November", "Desember"};
DateFormatSymbols DAfr  = new DateFormatSymbols(Afr);
DAfr.setMonths (AfrMonths);
```

Затем мы зададим шаблоны формата для страны, обратившись к средствам подкласса `SimpleDateFormat` класса `DateFormat`. Типичные объявления формата будут такими:

```
SimpleDateFormat Dshort = new SimpleDateFormat("yyy/MM/dd");

SimpleDateFormat Dfull = new SimpleDateFormat
       ("EEEE dd MMMM yyyy", DAfr);
```

Объект `Dshort` задает шаблон краткого представления даты, а объект `Dfull` — шаблон полного ее представления (с указанием месяца и т.д.). Объединив перечисленные средства, мы получим следующие операторы, которые можно поместить в конец программы `VaryDates`.

```
String AfrMonths [] = {"Januarie", "Februarie", "Maart", "April",
       "Mei", "Junie", "Julie", "Augustus", "September",
       "Oktober", "November", "Desember"};

String AfrDays [] = {"Saterdag", "Sondag", "Maandag", "Dinsdag",
       "Woensdag", "Donderdag", "Vrydag"};

Locale Afr = new Locale("Afrikaans","Suid-Afrika");
Locale.setDefault(genuine);

DateFormatSymbols DAfr  = new DateFormatSymbols(Afr);
DAfr.setMonths (AfrMonths);
DAfr.setWeekdays (AfrDays);
```

```
System.out.println("Мы находимся в " + Afr.getCountry() +
         " и разговариваем на языке " + Afr.getLanguage());

SimpleDateFormat Dshort = new SimpleDateFormat("yyy/MM/dd");
 System.out.println(Dshort.format(today));

SimpleDateFormat Dfull = new SimpleDateFormat
        ("EEEE dd MMMM yyyy", DAfr);
System.out.println(Dfull.format(today));
```

Выходные данные должны быть следующими:

```
Мы находимся в SUID-AFRIKA и разговариваем на языке afrikaans
01/11/20
Dinsdag 20 November 2001
```

В рассмотренных классах доступны еще многие средства. Вы найдете их в Java API с помощью опции **Help**.

Резюме

В Java строки не являются массивами. Это объекты, экземпляры которых создаются на основе специального класса `String`. Они обладают собственными свойствами и характеристиками. Существуют методы, которые выполняют взаимное преобразование строк и массивов символов, а также строк и чисел. В процессе преобразования иногда требуется помощь классов-оболочек, которые трансформируют переменные в объекты. В процессе обработки строк могут употребляться разделители. Все эти средства поддерживаются классом `Stream`, который используется при вводе данных в программах этой книги.

В Java предусмотрены мощные средства форматирования вывода. Числа, значения в разнообразных национальных валютах, процентные значения, а также даты могут быть представлены в различных форматах и адаптированы для использования в объектах класса `Locale`. В результате выбора объекта класса `Locale` язык и формат вывода приспосабливаются к соглашениям, принятым в данной стране. Часовые пояса также могут быть определены и заданы.

В главе дан обзор таких API:

```
String              Numberformat
  valueOf             getInstance
  charAt              getCurrencyInstance
  indexOf             getNumberInstance
  substring           getPercentInstance
  startsWith          format
  toCharAray          setMaximumIntegerDigits
  toLowerCase         setMinimumIntegerDigits
  trim                setMaximumFractionDigits
StringBuffer          setMinimumFractionDigits
StringTokenizer       setGroupingUsed
  nextToken         Date
  hasMoreTokens       after
  countTokens         before
Integer               getTime
  valueOf             setTime
  intValue
  parseInt
```

Описаны следующие объекты-исключения:

```
NoSuchElementException   ParseException
NumberFormatException
```

Следующий перечень включает темы, на которых в главе обращено особое внимание:

- создание строки;
- работа со строками;
- объявление и доступ к методам класса `StringTokenizer`;
- взаимные преобразования `Integer` и `int`;
- класс `Stream`;
- класс `Locale`;
- создание объектов для форматирования чисел;
- использование объектов для форматирования чисел;
- настройка объектов форматирования чисел;
- класс `Date`.

Вопросы



7.1. Если `name` — это переменная класса `String`, какое выражение позволит получить корректную позицию первой буквы "e", если таковая существует?

а) `name.indexOf('e')`

б) `name.startsWith('e')`

в) `indexOf(name,'e')`

г) `name.charAt('e')`

7.2. Если в строке `price` присутствуют данные '$567.89', как получить центы в виде строки (подстроки)?

а) `price.substring(price.indexOf('.'), price.length()-1);`

б) `price.startsWith('.', price.endsWith('9'));`

в) `price.substring(price.indexOf('.')+1,price.length());`

г) `price.substring('.',price.length);`

7.3. Если `tragic` — это объект `Calendar`, как мы должны установить в нем дату 31 August 1997?

а) `tragic.set(1997,Calendar.August,31);`

б) `tragic.set(31, 8, 1997);`

в) `tragic.set (1997, 8, 31);`

г) `tragic.set (1997, Calendar.AUGUST, 31);`

7.4. Чтобы ввести надлежащим образом 10-цифровые номера чеков, нужно сначала считать строку и преобразовать ее в массив символов. Почему?

а) в классе `Stream` нет метода `readChar`

б) чтобы символы можно было считать, за ними должны следовать пробелы

в) число из 10 цифр слишком длинное, чтобы его можно было считать как `integer`

г) нельзя сравнить строки и символы

7.5. Имеются следующие строка и разделитель:

```
String s = "A678230056 7";
StringTokenizer T = new Stringtokenizer(s, "0123456789", false);
```

Какие лексемы будут получены на выходе, если к строке применить цикл разбиения на лексемы?

а) `A67230056` и `7`

б) `7`

в) `A` и `7`

г) `A` и пробел

7.6. Если цена `price` имеет тип `int` и мы хотим включить ее в хеш-таблицу, она должна быть помещена в класс-оболочку как объект. Для этой цели предназначено выражение:

а) `(Integer) price`

б) `new Integer (price)`

в) `price.intValue()`

г) `Integer (price)`

7.7. В классе `Stream` методы форматирования объявляются статическими, потому что:

а) они работают с выходными данными, а не входными

б) они не относятся ни к одному объекту `Stream`

в) два из них имеют одинаковые имена

г) в результате их работы система генерирует сообщение об исключительной ситуации `IOException`

7.8. Что будет выведено в результате выполнения следующей последовательности операторов?

```
NumberFormat N = NumberFormat.getInstance();
N.setGroupingUsed(false);
N.setMaximumFractionDigits(4);
N.setMinimumIntegerDigits(4);
System.out.println(N.format(Math.Pi)+" "+N.format(10000));
```

а) `3.1416 10000`

б) `0003.1416 10000`

в) `3.1416 10000.0000`

г) `3 10000`

7.9. Региональные стандарты могут быть назначены:

а) программой Java

б) пользователем через операционную систему

в) двумя указанными выше способами

г) ни одним из них

7.10. Если в объекте `Locale` указаны региональные стандарты Германии и мы вводим такую дату, как 07-May-2000, что произойдет при ее синтаксическом анализе с помощью метода `parse` класса `DateFormat`?

а) компьютер интерпретирует эту дату надлежащим образом, т.е. 7-5-2000

б) отобразится сообщение `parseException`

в) элемент `May` будет преобразован в установленное по умолчанию значение `0`, т.е. `January`

г) система попросит повторить ввод даты

Задачи

По теме "Обработка строк"

7.1. **Проверка кодов.** Курсовые коды в Savanna University имеют фиксированную структуру: три прописные буквы, за которыми следуют три цифры. Напишите класс для таких кодов и включите в него методы для их создания, преобразования в строку и проверки на достоверность. Проверьте класс с помощью небольшой тестовой программы.

7.2. **Палиндромы.** Палиндромы — это слова или предложения, которые одинаково читаются в обоих направлениях (слева направо и справа налево), если не учитывать знаки препинания. Известными палиндромами являются:

Madam, I'm Adam
Able was I ere I saw Elba

Напишите метод, проверяющий, является строка палиндромом или нет. Вызовите метод из тестовой программы.

7.3. **Преобразование имен.** При создании файла с именами людей использовались исключительно прописные буквы. Необходимо представить имена в обычной форме: первая буква — прописная, остальные — строчные. Следует учитывать инициалы, а также то, что префиксы к фамилиям, такие как Du или Von, начинаются с прописной буквы. Например:

M A STORE	M A Store
J.FOX-ROBINSON	J.Fox-Robinson
P DU PLESSIS	P Du Plessis

По теме "Разделители"

7.4. Программа **индексирования.** В процессе программирования полезно знать, где встречается каждый из идентификаторов. Напишите программу, которая индексирует код надлежащим образом, т.е. создает список идентификаторов, отсортированных по алфавиту, с указанием номеров строк, где присутствуют эти идентификаторы.

7.5. **Алфавитный указатель.** Под алфавитным указателем следует понимать список слов, в котором для каждого элемента приводится номер страницы и строки. Различие между алфавитным указателем и индексированным списком состоит в том, что в первом случае учитываются все слова, за исключением часто употребляемых (таких как "and", "the" и "is"). Напишите программу, формирующую алфавитный указатель для произвольного текста небольшого объема. Используйте разделители и хеш-таблицы. (Подсказка: в хеш-таблице слова должны быть ключами, а номера страниц/строк — значениями. Поскольку слова могут встречаться более одного раза, значением должен быть массив

чисел. Кроме того, значение должно иметь верхний предел, например десять случаев употребления данного слова.)

7.6. **Эгалитарные сообщения.** Так называют сообщения, в которых все лица упоминаются одинаково, независимо от титулов или чина. Совет университета Саванны постановил, что на собраниях во всех документах надлежит опускать титулы выступающих. Так, вместо проф. Браун следует употреблять один из следующих вариантов: Джон Браун, Браун. ДжБ или Джон. Важно также соблюдать унификацию (не использовать варианты одного имени). Ваша задача — обработать документ, найти титулы и префиксы (проф., др., м-р, мисс, миссис и т.д.), удалить их и представить имя в выбранной форме.
Программе понадобится информация об именах людей. Например, если выбран вариант "Джон Браун", а программа встретила словосочетание "проф. Браун", ей потребуется где-то найти недостающее имя "Джон". Возможны такие пути:
   — познакомиться с секретаршей или иным лицом, которое будет применять эту программу, и получить соответствующую информацию;
   — на основе списков приглашенных и списков отказавшихся от участия составить хеш-таблицу с необходимой информацией. Следует исходить из того, что списки содержат все данные, касающиеся имени человека и обращения к нему (например, проф. Джон Браун (ДжБ)). Для нахождения списков используйте метод `startWith` класса `String`.

   Вы должны решить, как будет организована хеш-таблица (что послужит ключом). Помните, что искать можно и ключи, и значения.

По форматированию

7.7. **Имена из нескольких слов.** В программе "Олимпийские медали" из примера 4.10 мы столкнулись с проблемой, которая заключалась в отсутствии возможности обрабатывать имена, содержащие пробелы. Дело в том, что пробел относится к разделителям. Мы решили эту проблему, изменив данные так, что, например, название страны United.States записывалось с точкой посредине. Теперь мы знаем больше о форматировании. Какой же способ будет лучшим для применения в методе, осуществляющем чтение таких имен? Учтите, что имена распределены по колонкам фиксированного размера, в которых отсутствуют другие элементы. Напишите подходящий метод для считывания имен, состоящих из нескольких слов, включите его в программу как часть класса `Formatters` из пакета `myutilities`* и используйте, чтобы избавиться от ограничения, характерного для программы "Олимпийские медали".

7.8. **Улучшенное форматирование цены.** Вернемся к задаче 5.9. Используйте форматирующий числа объект, чтобы вывести цены на автомобили в соответствующем формате для валюты страны, с которой вы хорошо знакомы.

7.9. **Изменяемое форматирование.** Напишите программу, которая выводит действительные числа (генерируемые случайно) в формате, указываемом через интерфейс класса `Display`. Это означает, что нужно использовать шаблон `###0.###`, где количество знаков `#` до и после точки может изменяться.

---

* Для этого нужно выполнить компиляцию класса `Formatters` в пакете `myutilities`. — *Прим. ред.*

По теме "Даты, календари и время"

7.10. **Дни рождения.** Методы `after` и `before` класса `Calendar`, который находится в пакете `java.util`, возвращают значение `true` или `false` в зависимости от отношений между экземпляром объекта и параметром. Например, выполнение сравнения (метода `before`) в выражении

```
new Calendar (1951, Calendar. DECEMBER, 14).before
                (new Calendar (1951, Calendar. December, 1));
```

даст в результате `false`. Напишите программу, в которой производится ввод даты вашего рождения и даты Пасхи в текущем году, а также вывод результата каждого из трех отношений — `before`, `equals` и `after` — между двумя датами.

7.11. **Настройка региональных стандартов.** Выберите любую страну, для которой еще не создан объект класса `Locale`, и произведите для нее настройку языка и региональных стандартов, как в примере 7.8. Исследуйте интерфейсы API посредством команды **help** языка Java и попробуйте отрегулировать форматирование валюты.

7.12. **Всенародные праздники.** Вспомните задачу 6.6, в которой применялся класс `Date`. Найдите в справочниках праздники одной из стран, перечисленных в описании класса `Locale` (см. раздел 7.4), и напишите программу, выводящую даты в соответствующем этой стране формате. Информацию о всенародных праздниках можно найти по такому адресу:

`http://www.globalsources.com/TNTLIST/TRAVEL/HOLIDAYS.HTM`.

ГЛАВА 8



Объекты: применение на практике

Java является объектно-ориентированным языком. В предыдущих главах мы познакомились со многими объектами и классами и приобрели навыки их практического применения. Теперь, когда базовые сведения уже изложены, настало время раскрыть всю мощь объектов и продемонстрировать их работу в программах большего объема. В этой главе мы проанализируем типичную программу по инвентаризации, а в следующей продолжим обзор функциональных возможностей объектов и поговорим о наследовании, интерфейсах и абстрактных классах.

8.1. Разработка объектно-ориентированной программы

В примере, предлагаемом вашему вниманию, интенсивно используются классы и создается множество объектов — экземпляров этих классов.

Пример 8.1. Кофейный магазин Нельсона

Задача. Администрация кофейного магазина Нельсона решила автоматизировать процесс инвентаризации. Требуется исследовать, какая информация о запасах кофе необходима и какие операции обработки придется выполнить.

Решение. На данном этапе мы не будем писать всю программу инвентаризации, а только определим объекты. В процессе работы следует учитывать, что, хотя реализация кофе является спецификой данного магазина, в будущем он может расширить свой ассортимент и заняться продажей макаронных изделий, мороженого и т.д.

Определение. В магазине продается несколько сортов кофе: Ява, Колумбийский, VIP и Кенийский. Поэтому мы создадим класс для кофе и различные объекты этого класса. Во время анализа запасов кофе мы можем сформировать список элементов данных, которые нужно принимать во внимание при записи информации о конкретном сорте. Такой список является начальной точкой в определении объекта. Кроме того, мы укажем возможный тип каждого элемента данных. Первый вариант списка представлен в табл. 8.1. Строки, действительные числа, целые числа и даты имеют эквиваленты в Java.

Таблица 8.1. Элементы данных для класса Coffee

Элемент	Тип
Имя (название сорта)	`string`
Цена	`double` (грацы)

Элемент	Тип
Объем запасов	`integer` (килограммы)
Минимальный объем поставки	`integer` (килограммы)
Штрих-код	`string`
Крайний срок реализации	`date`

Предложенный вашему вниманию список довольно прост. Первые пять характеристик относятся ко всем партиям кофе, а срок реализации специфичен для каждой партии. Это означает, что `batch` следует сделать другим классом. В нем необходимо указывать не только дату, но и количество кофе, в килограммах, имеющееся в наличии с данным сроком реализации. Главный объект `Coffee` должен содержать итоговые данные по партии. Связь между классами показана на рис. 8.1.

Следующая часть определения касается операций. Мы разделили их на две группы: операции для сорта кофе (табл. 8.2) и операции для конкретных партий (табл. 8.3). Мы начнем с определения операций, относящихся к сорту кофе. Некоторые из них касаются также и партий.



Рис. 8.1. Набор классов и объектов для магазина кофе

Мы получили иерархию: объекты кофе вызывают операции партий, но не наоборот. Метод по удалению старого запаса должен рассмотреть каждую партию и аннулировать ее, если обнаружится, что срок ее хранения закончился. Кроме того, следует учесть, что вся партия может быть продана.

Как может быть удален объект? Все зависит от того, как организованы объекты `Batch`. Единственное знакомое нам средство группировки — это массив, который не очень подходит для тех случаев, когда требуется удалить элемент: индексы всех элементов, следующих за удаленным, нужно уменьшить на единицу, иначе придется обрабатывать массив с удаленными элементами.

Конечно, существует более совершенный способ. Мы познакомимся с ним в разделе 8.3 и вернемся к кофейному магазину в разделе 8.4. Ведь пока что мы не будем применять методы, относящиеся к продаже кофе или удалению старых фондов, а ограничимся тем, что поместим партии в массив.

Таблица 8.2. Операции с сортами кофе

Операции с кофе	Влияние на партии
Подготовка к приему товара (stock)	
Рассмотрение данных о сорте кофе (display)	

Операции с кофе	Влияние на партии
Назначение объема уровня поставки (reorder)	
Изменение цены	
Увеличение фондов (stock)	Добавление новой партии (batch)
Продажа кофе (sell)	Уменьшение фондов (stock) соответствующей партии (batch)
Удаление старых фондов (stock)	Проверка крайнего срока реализации (date)

Таблица 8.3. Операции с партиями кофе

Операции с партиями	Влияние на операции с кофе
Добавление новой партии	
Рассмотрение данных о партии (display)	
Определение доступного количества продукции	
Проверка крайнего срока реализации	
Продажа (sell)	

Классы. Теперь попробуем описать структуру программы на языке Java. Начнем со спецификации класса `Batch`

```
class Batch {

  /* Класс Batch  Автор - Дж. М. Бишоп, октябрь 1996 г.
   * - - - - - -  Java 1.1 - октябрь 1997 г.
   * Предназначен для продукции, имеющей крайний
   * срок реализации.
   * Может быть проверен с помощью класса Coffee.
   */
  private double inStock;
  private Date sellByDate;
  private DateFormat DF = DateFormat.getDateInstance();

  Bath    (double k, Date d) { }
  double  available ( ) { }
  boolean sell (double k) { }
  void    display () { }
}
```

Детали методов еще не указаны. Давайте сначала рассмотрим, как они используются. Для создания новой партии класс `Coffee` должен включать следующий оператор:

```
freshBatch = new Batch (kilos, roastDay);
```

Новая партия должна быть каким-то образом связана с другими партиями, но пока эта тема не будет освещаться. Для продажи соответствующий метод выбирает наиболее старую партию кофе. Если в этой партии товара недостаточно, предлагается следующая партия. Давайте будем исходить из того, что в массиве наиболее старые *поставки кофе относятся к партии* `B[i]`. Оператор, выполняющий обращение к наиболее старой партии для продажи `k1` килограммов кофе, имеет такой вид:

```
B[i].sell(k1);
```

Если нужно уменьшить фонды другой партии, идентифицированной как B[j], воспользуйтесь конструкцией

```
B[j].sell(k2);
```

которая позволит обратиться к другим фондам и дате. Эти две партии не будут перепутаны, поскольку они относятся к разным объектам.

Что касается доступности, объект Batch всегда будет использоваться объектом Coffee, поэтому не нужно объявлять его как public (открытый) на этой стадии. Для четырех методов, которые он предоставляет классу Coffee, также можно задать вид доступа, применяемый по умолчанию. Поля, которые использует объект Batch, не должны быть доступными извне, поэтому они объявляются как private (*закрытые*).

Определение класса Coffee осуществляется аналогичным образом:

```
class Coffee {
   /* Класс Coffee  Автор - Дж. М. Бишоп, октябрь 1996 г.
    * - - - - - - - - Java 1.1 октябрь 1997 г.
    *                 обновление - август 2000 г.
    * Сохраняет результаты инвентаризации партий кофе
    * конкретного сорта.
    */

  Coffee () { }

  Coffee (String s) { };

  void prepareToStock (Stream in) { }
  boolean newBatchIn (Stream in) { }

  void display (0 { }
  double sell (double k) { }
  boolean stocksLow () { }
  void   changePrice (double p) { }
  void  checkSellByDate () { }

  private String name;
  private double price;  //кг
  private double stock; //кг
  private double reorder; //кг
  private batch B [ ] = new Batch [10];
  private int batches = 0;
  private DateFormat DF = DateFormat.getDateInstance(DateFormat.
          DEFAULT, Locale.UK);
}
```

Как и в случае с классом Batch, все переменные объявляются как закрытые, благодаря чему они не могут быть изменены извне, а только с помощью методов этого класса. Конструктор не имеет параметров и, фактически, ничего не делает: все установки выполняются внутри метода prepareToStock. Почему мы поступаем таким образом? Причина заключается в том, что данные о кофе будут считываться из файла, они недоступны непосредственно в программе. В следующей версии программы будет добавлен второй конструктор.

Методы prepareToStock и newBatchIn будут считывать данные, а также открывать и закрывать свои потоки данных (рис. 8.2). Все переменные в обоих классах объявляются закрытыми (см. главу 5), что обозначено на рисунке знаками "минус".

Массив партий, относящихся к классу `Coffee`, объявляется как содержащий элементы класса `Batch`. Графически это показано посредством стрелки. Один из элементов массива определен как объект `Batch`.

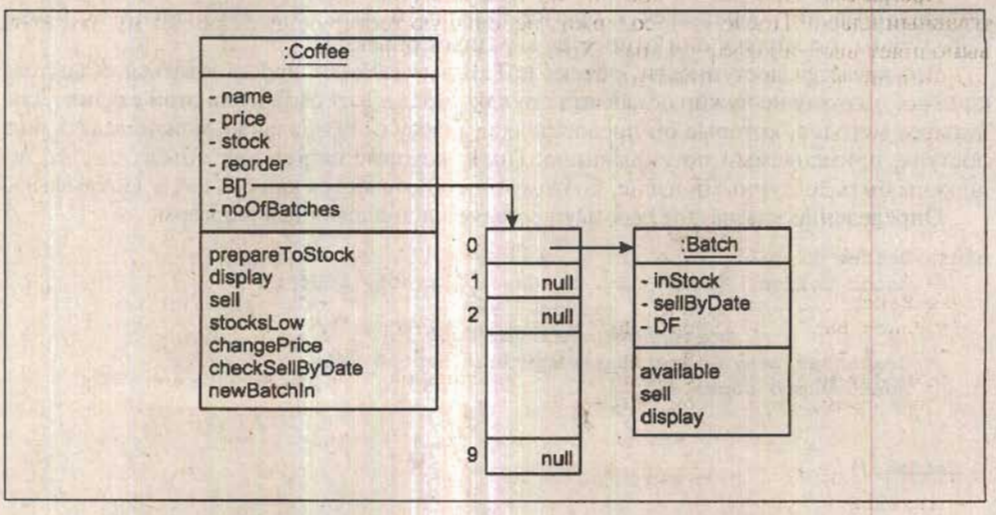


Рис. 8.2. Схема организации классов `Coffee` и `Batch`

Данные. При вводе данных о кофе важным является вопрос о том, как они должны быть организованы. На этой стадии мы предпочли содержать их в файле, поскольку так проще протестировать программу. Альтернативный вариант — построчный ввод данных — окажется скучным и утомительным.

Данные могут быть упорядочены в файле следующим образом:

```
название
цена    минимальный объем поставки
фонды   крайний срок реализации (повторяется для каждой партии)
0
```

Если взять за основу схему, приведенную на рис. 8.1, они будут выглядеть так:

```
Colombian
59.95 5
45.5 23-Jun-00
5 30-Jul-00
0
Kenyan
60.75 10
15 19-May-00
0
Java
63.50 3
61.88 23-May-00
20 12-Jul-00
3 1-Aug-00
0
```

Ноль, введенный в файле после данных по каждому комплекту партий, необходим для завершения цикла (см. раздел 5.2). Дата вводится в виде строки и преобразуется в объект `Date`. Как это происходит, описано в разделе 7.5.

Программа. Программа состоит из трех классов: `Batch`, `Coffee` и `CoffeeShop` (главный класс). Последний содержит небольшую тестирующую программу, которая выполняет ввод и вывод данных. Хотя на этом этапе ничего больше не происходит, мы многого достигли: написали самую большую (на текущий момент) объектно-ориентированную программу, которая содержит объекты двух видов, определенные пользователем. Ради интереса можете сосчитать, сколько в ней объектов других видов.

```
import java.io.*;
import java.util.*;
import java.text.*;
import javagently.*;

class Batch {
  /* Класс Batch     Автор - Дж. М. Бишоп, октябрь 1996 г.
   * - - - - - - -    Java 1.1 - октябрь 1997 г.
   * Хранит информацию о продукции с указанными сроками реализации.
   * Используется классом Coffee.
   */

  Batch (double k, Date d) {
    inStock = k;
    sellByDate = d;
  }
  double available (  ) {
    return inStock;
  }

  boolean sell (double k) {
    inStock -= k;
    System.out.println("Продано "+ Stream.format(k, 1, 1)
        +" кг из партии "+ DF.format(sellByDate));
    return inStock <= 0; // продано
  }

  void display () {
    System.out.println("\t\t"+DF.format(sellByDate) +
        " : " + Stream.format(inStock, 4, 1) + " кг");
  }

  private double inStock;
  private Date sellByDate;
  private DateFormat DF = DateFormat.getDateInstance(DateFormat.
          DEFAULT, Locale.UK);
}

// = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
import java.io.*;
import java.util.*;
import javagently.*;
import java.text.*;

class Coffee {
```

```
/* Класс Coffee  Автор - Дж. М. Бишоп, октябрь 1996 г.
 * - - - - - -   Java 1.1 - октябрь 1997 г.
 *               Обновление - август 2000 г.
 * Сохраняет данные об инвентаризации партий кофе
 * конкретного сорта.
 */

Coffee () {
}

Coffee (String s) {
  name = new String (s);
};

void prepareToStock (Stream in)
          throws IOException {
  name = in.readString();
  price = in.readDouble();
  reorder = in.readInt();
  batches = 0;
  stock = 0;
}

boolean newBatchIn (Stream in) throws IOException {
  double amount;
  Date sellby;
  String s;

  amount = in.readDouble();
  if (amount == 0) // партия продана
    return false;
  else {
    stock += amount;
    sellby = new Date ();
    s = in.readString();
    try {
       sellby = DF.parse(s);
    } catch (ParseException e) {
       System.out.println("Ошибка в дате " + s);
    }
       B[batches] = new Batch (amount,sellby);
       batches ++;
       return true;
  }
}

void display () {
  System.out.println(name+" @ G"+
        Stream.format(price,1,2)+" за кг");
  for (int i = 0; i < batches; i++) {
    B[i].display ();
  }
  System.out.println("\t" + Stream.format(stock, 1, 1) +
     " кг в фондах. Минимальный объем поставки равен "
     +reorder+" кг\n");
}
```

```
  private String name;
  private double price;  //за кг
  private double stock; //кг
  private double reorder; //кг
  private Batch B [ ] = new Batch [10];
  private int batches = 0;
  private DateFormat DF = DateFormat.getDateInstance(DateFormat.
          DEFAULT, Locale.UK);
}

// = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
import java.io.*;
import java.util.*;
import javagently.*;
import java.text.*;
import myutilities.*;

class CoffeeShop {
  /* Класс CoffeeShop  Автор - Дж. М. Бишоп, октябрь 1996 г.
   * - - - - - - - -   Java 1.1 - октябрь 1997 г.
   *                   Обновление - август 2000 г.
   * Проверяет, как данные могут быть включены в классы
   * Coffee и Batch и затем выведены.
   * Использует массив для хранения нескольких объектов.
   */

 public static void main (String [] args) throws IOException {

   Stream in = Filer.open("coffee.dat");

   Coffee C [ ] = new Coffee [10];
   int kind;

   // Общая информация.
   System.out.println("Кофейный магазин Нельсона");
   System.out.println("Инвентаризационный контроль");
   System.out.println("Данные по фондам из файла coffee.dat");

   // Ввод данных о кофе.
fs24    for (kind=0; kind<10; kind++) {
     try {
       C[kind] = new Coffee ();
       C[kind].prepareToStock (in);
       boolean morebatches = true;
       while (morebatches)
         morebatches = C[kind].newBatchIn (in);
     }
     catch (EOFException e) {
       System.out.println("Фонды магазина на сегодня\n");
       break;
     }
   }

   // Выводит подробную информацию.
   DateFormat DF = DateFormat.getDateInstance(DateFormat.FULL);
   System.out.println("Имеется в наличии на " +
        DF.format(new Date()));
```

```
    for (int i = 0; i<kind; i++)
      C[i].display();
  }
}
```

Тестирование. С учетом указанных выше данных результат работы программы будет следующим:

```
Кофейный магазин Нельсона
Инвентаризационный контроль
Данные по фондам из файла coffee.dat
Фонды магазина на сегодня

Имеется в наличии на 23 ноября 2001 г.
Colombian @ G59,95 за кг
                23-Jun-00 : 45,5 кг
                30-Jul-00 :  5,0 кг
        50,5 кг в фондах. Минимальный объем поставки равен 5.0 кг

Kenyan @ G60,75 за кг
                19-May-00 : 15,0 кг
        15,0 кг в фондах. Минимальный объем поставки равен 10.0 кг

Java @ G63,50 за кг
                23-May-00 : 61,8 кг
                12-Jul-00 : 20,0кг
                01-Aug-00 :  3,0 кг
        84,8 кг в фондах. Минимальный объем поставки равен 3.0 кг
```

8.2. Свойства объектов

В этом разделе подробно рассматриваются свойства объектов, а также способы манипуляции ими в программе.

Применение ссылок

Начиная с главы 3 мы наблюдали, как ссылки используются для передачи параметров, а объектные переменные — для формирования части объявлений классов естественным образом. Еще один способ применения ссылок состоит в составлении **списка** объектов одинакового вида. Такой список может быть использован для преодоления проблемы, с которой мы столкнулись в предыдущем разделе, когда были вынуждены установить верхний предел для количества сортов кофе, поскольку использовали массив. Далее мы изучим встроенные классы Java, предназначенные для формирования списков и работы сними, а пока вкратце обрисуем, как с "нуля" создать список.

Для составления списка сортов кофе мы добавим в класс `Coffee` переменную `Coffee`, которая будет связана со следующей переменной `Coffee`:

```
Coffee link;
```

Конструктор внутри класса `Coffee` должен быть изменен так, чтобы добавляемый в список объект ссылался на уже существующий, например:

```
Coffee (String s, Coffee c) {
  name = s;
  link = c;
}
```

В результате объявления переменной типа `Coffee` получаем переменную со значением. Определяя первый сорт кофе (первый объект списка), мы используем такую запись:

```
Coffee coffees = new Coffee ("Java", null);
```

и получаем объект, который способен устанавливать связь с другими объектами `coffees` посредством ссылок (рис. 8.3).

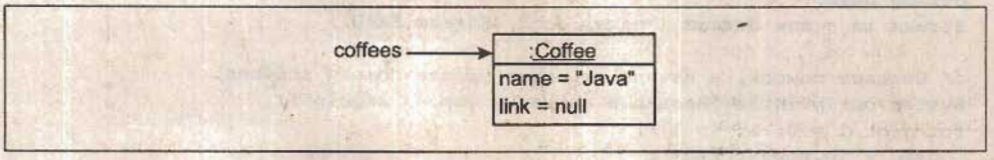


Рис. 8.3. Первый элемент списка

Если теперь мы повторим процедуру и создадим экземпляр другого объекта, используя предыдущий объект `coffees` как ссылку и присваивая результат создаваемому объекту `coffees`:

```
Coffee coffees = new Coffee ("Colombian", coffees);
```

то получим схему, представленную на рис. 8.4. Описанную последовательность действий можно продолжить. Список будет расширяться за счет добавления в его начало новых записей о сортах кофе. Последним всегда будет указываться сорт "Ява". Фактическое имя переменной `coffees` после каждого назначения, а значит и создания экземпляра, будет ассоциироваться с другим объектом. Другие объекты, представляющие сорта кофе, не будут иметь идентификаторов, а доступ к ним будет возможен только через ссылку.

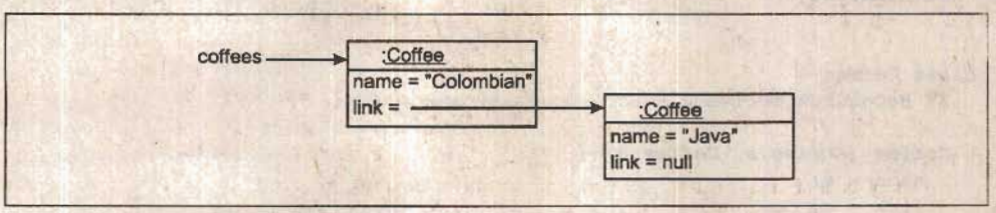


Рис. 8.4. Два элемента списка

Пример 8.2. Составление списка

Задача. Созданная программа проиллюстрирует составление списка объектов.

Решение. Мы уже знаем, как решается данная задача. В этом примере будет написана небольшая программа для проверки используемой технологии, которая создает список из пяти названий сортов кофе, считанных из файла, а затем выводит их.

Программа

```
import java.io.*;
import javagently.*;

class LinkTest {
```

```
 /* Программа LinkTest    Автор - Дж. М. Бишоп, август 2000 г.
  * ==================
  * Небольшая программа-тест для демонстрации связей
  * между объектами, реализуемых посредством ссылок.
  */

LinkTest () throws IOException {
Coffee coffees = null;
 String name;
 Stream in = new Stream ("names.dat", Stream.READ);

 // Создает список, в начале которого указан объект coffees.
 System.out.println("Вводимые названия сортов кофе\n");
 for (int i = 0; i<=4; i++) {
   name = in.readString();
   System.out.println(name);
   coffees = new Coffee (name, coffees);
 }

 System.out.println("\nСорта кофе в списке\n");
 // Копирует первый элемент списка.
 Coffee list = coffees;
 // Выводит на печать список.
 for (int i = 0; i<=4; i++) {
   System.out.println(list.name);
   list = list.link;
 }
}

public static void main (String args []) throws IOException {
 new LinkTest();
}

class Coffee {
  /* Небольшой тестовый класс, использующий ссылки. */

  Coffee (String s, Coffee c) {
    name = s;
    link = c;
  }

  String name;
  Coffee link;
}
}
```

Обратите внимание на следующую особенность программы: когда были введены все названия сортов и сформирован список, мы создали копию первого элемента списка в объекте `list` и использовали его для обратной связи (посредством ссылок) со всеми объектами. На рис. 8.5 воспроизведена ситуация с переменными, возникшая после того, как было выведено название первого сорта и фактическое имя переменной стало ассоциироваться со вторым элементом списка.

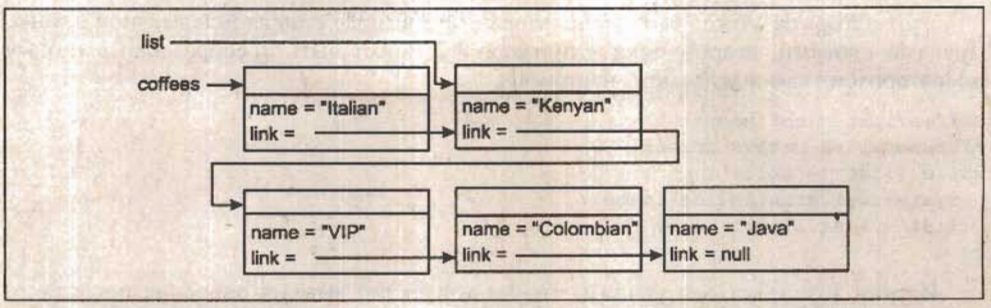


Рис. 8.5. Связный список с двумя объектами в качестве первых элементов списка

Следовательно, список `list` постепенно уменьшается по мере того, как выводится очередное название сорта. В конечном счете количество элементов в нем будет сведено к нулю. В процессе создания списка таким способом данные как бы помещаются в стек, откуда извлекаются при выводе в обратном порядке, что и продемонстрировано при тестировании*.

Тестирование

```
Вводимые названия сортов кофе

Ява
Колумбийский
VIP
Кенийский
Итальянский

Сорта кофе в списке

Итальянский
Кенийский
VIP
Колумбийский
Ява
```

Мы также должны проверить, как будет работать программа, если файл окажется пустым. Чтобы сделать это надлежащим образом, нам придется переписать цикл ввода так, чтобы в нем использовался обработчик исключительных ситуаций. В цикле вывода надлежит обеспечить применение оператора `while`. Именно эта тема будет освещена следующей.

Использование значения null

Как вы уже знаете, набор значений для объектных переменных начинается со значения **null**. Если мы попытаемся получить доступ к элементам нулевого объекта, Java выдаст сообщение об исключительной ситуации `NullPointerException`**. Прежде чем продолжить считывание значений полей `name`, мы можем проверить, является ли объект нулевым.

---

* Использовался файл `names.dat`, в котором сорта кофе переведены на русский язык. — *Прим. ред.*

** Должен быть объект `NullObjectException`, но в Java проявляется влияние C++.

Этот алгоритм может быть использован при выводе списка неизвестной длины. Другими словами, второй цикл в примере 8.2 может быть преобразован в цикл с оператором `while` и условием, например:

```
Coffee list = coffees;
// Выводит на печать список.
while (list != null) {
  System.out.println(list.name);
  list = list.link;
}
```

Изучив этот цикл, мы увидим, что он корректно выведет значение последнего поля `name` (строку "Ява"), а затем подойдет к нулевой ссылке и прекратит свою работу. В качестве примера создадим цикл с обработкой исключительной ситуации:

```
Coffee list = coffees;
// Выводит на печать список.
try {
  while (true) {
    System.out.println(list.name);
    list = list.link;
  }
catch (NullPointerException e) { // достигнут конец списка
}
```

Идентификатор this

Каждый объект может получить доступ ко всем своим членам по имени. Полным именем элемента является `this.member`, где `this` — это идентификатор, указывающий на текущий объект. Если имеется двусмысленность в именах элементов, использование `this` не обязательно. Мы будем применять `this` в случае, когда локальное поле или параметр метода имеет то же имя, что и поле или параметр класса: в такой ситуации мы сможем обращаться к полю класса, используя полное имя поля с приставкой `this`.

Например, мы можем представить конструктор `Coffee` из примере 8.2 следующим образом:

```
Coffee (String name, Coffee c) {
  this.name = name;
  link = c;
}
```

В этой книге мы предпочли присваивать параметрам конструктора однобуквенные идентификаторы, поскольку их смысл понятен из названий переменных, в которые они копируются. Как здесь показано, мы можем использовать одинаковые имена для параметра и его переменной, применяя `this` во избежание двусмысленности.

Суперкласс Object

В Java существует суперкласс, стоящий над всеми классами, который называется `Object` (см. раздел 6.5). Благодаря ему объекты различных классов можно использовать в одних и тех же частях программы под одним именем — `Object`. Этот принцип важен в клонировании, которое будет описано ниже.

Равенство объектов

Тот факт, что объекты имеют в качестве своих значений ссылки, имеет два следствия. Первое касается **равенства** (см. раздел 7.1). Конструкция

```
B[i] == B[j]
```

представляет собой прямое сравнение *ссылок* двух объектов (в данном случае — `Batch`) и позволяет определить, относятся ли они к одному и тому же объекту. Сравнение значений объектов не производится. Для сравнения значений полей (как группы) надлежит использовать метод `equals`, который следует определить для каждого нового класса объектов. Поскольку метод `equals` определяется пользователем, последний должен решить, на основе чего устанавливать равенство. Принимаем, что две партии являются равными, если у них одинаковая дата реализации. При этом не имеет значения, сколько килограммов кофе в каждой из них. Следовательно, метод, определяющий равенство, будет выглядеть следующим образом:

```
boolean equals (Batch b) {
  return (sellByDate.equals(b.sellByDate));
}
```

Поскольку даты тоже являются объектами, для их сравнения мы должны использовать метод `equals` класса `Date`, поэтому фактически сравнение выполняется на двух уровнях.

Назначение объекта

Вторым следствием хранения объектов в памяти является **назначение** одного объекта другому. Например, при вводе конструкции

```
tempBatch = B[i];
```

будет скопирована *ссылка*. Для копирования значений мы должны клонировать* объект, используя метод, который вновь должен быть определен для каждого нового класса, как показано ниже.

Объявление клона

```
class Classid implements Cloneable {

  Объявления переменных класса Classid

  public Object clone ( ) {
    Classid obj = new classname (parameters);
    Операторы копирования
    return obj;
  }
}
```

Прототип метода `clone` создает новый объект этого класса, а затем использует операторы для присвоения новым переменным объекта всех текущих значений переменных. Новый объект возвращается как значение метода `clone`.

* Что касается термина "клонирование", вы можете воспринимать его как "копирование", если вам так больше нравится.

В дополнение отметим, что клонирование является специальной операцией в Java, следовательно, любой класс, в котором предполагается клонирование, должен применять интерфейс `Cloneable`, что указывается в объявлении класса. Интерфейсам посвящена следующая глава, а пока нам достаточно знать, что интерфейс `Cloneable` имеет метод `clone`, который мы применяем.

На рис. 8.6 показано различие между операцией назначения одного объекта другому и клонированием объектов.

В процессе присвоения ссылка объекта `temp1` становится такой же, как и ссылка объекта `B[0]`. Любые изменения, произведенные над объектом `temp1`, отражаются на объекте `B[0]`. При клонировании создается копия объекта, на которую ссылается объект `temp2`. Теперь объекты `temp2` и `B[0]` могут изменяться независимо.

Как упоминалось ранее, каждый класс отвечает за определение того, что подразумевается под `equals` и `clone`. В нашем примере клоном для объекта `Batch` может быть

```
public Object clone () {
  Batch x = new Batch (inStock,null);
  x.sellByDate = new Date(sellByDate.getTime());
  return x;
}
```

Клонирование партии предполагает копирование таких элементов данных, как объем запасов и даты. Переменная `inStock` является примитивной, поэтому она может быть назначена или помещена в конструктор нового объекта, как в данном случае. Копирование даты заключается в создании нового объекта даты (`Date`) и помещении его в соответствующий конструктор.

Преобразования классов

В разделе 3.1 указывалось, что между примитивными переменными различных типов допускаются взаимные преобразования. То же относится и к объектам. В частности, в Java есть методы, определенные для суперкласса `Object`, которые могут быть использованы другими объектами. Метод `clone` является одним из них, однако прежде чем будут назначаться результаты выполнения этого метода, должно быть сделано явное преобразование типа. Поэтому для клонирования и вывода объекта `Batch` потребуется выполнение таких операторов:

```
Batch tempBatch = (Batch) B[0].clone();
System.out.println(tempBatch);
```

В результате вызова метода `clone` для объекта `B[0]` возвращается объект, который является экземпляром суперкласса `Object`. Он *должен* вернуть объект `Object`, поскольку именно этого требует интерфейс `Cloneable`. Но мы можем преобразовать его в объект класса `Batch`, указав имя класса. Вы можете внедрить метод клонирования в объект `Batch`, а указанный выше оператор — в объект `Coffee`. В завершение проверьте, что все это работает.

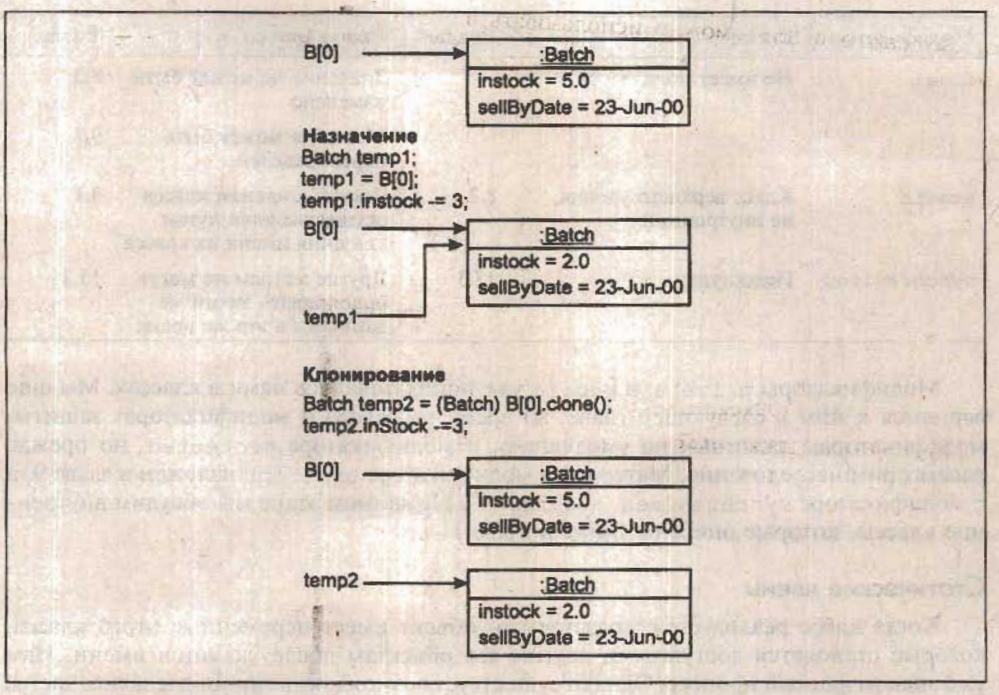


Рис. 8.6. Различие между операциями присвоения и клонирования

Модификаторы

Классам и членам классов могут быть присвоены модификаторы, которые указывают, как их надлежит использовать. Не все модификаторы применимы к классам: некоторые предназначены только для членов (методов, переменных). В табл. 8.4 описана область применения главных модификаторов Java, а также указываются разделы этой книги, в которых соответствующие темы рассмотрены в полном объеме.

Таблица 8.4. Назначение основных модификаторов, применяемых по отношению к классам и членам, с указанием разделов, содержащих информацию о них

Модификатор	Класс	Раздел	Член класса	Раздел
Не указан	Доступен только в рамках своего пакета	5.1	Доступны только в рамках своего пакета	5.1
`public`	Доступен везде, где указан его пакет	4.6	Доступны везде, где указан их класс	4.6
`private`	Недоступен		Доступны только в рамках их собственного класса	5.1
`protected`	Недоступен		Доступны в рамках их пакета и подклассов	9.2
`abstract`	Экземпляр не может быть создан	9.4	*Метод применяется* не там, где он определен, а в подклассе	9.4

Модификатор	Класс	Раздел	Члены класса	Раздел
final	Не имеет подклассов		Значение не может быть изменено	8.2
			Метод не может быть переопределен	9.2
static	Класс верхнего уровня, не внутренний	8.2	Доступ к членам класса осуществляется путем указания имени их класса	3.1
synchronized	Недоступен	13.3	Другие методы не могут оперировать этими же данными в это же время	13.3

Модификаторы `private` и `public` уже использовались нами в классах. Мы еще вернемся к ним в следующей главе, во время разговора о модификаторах защиты, модификаторах, заданных по умолчанию, и модификаторе `protected`, но прежде рассмотрим наследование. Материал о модификаторе `abstract` изложен в главе 9, а о модификаторе `synchronized` — в главе 13. На данном этапе мы обсудим внутренние классы, которые описаны также в главе 11.

Статические члены

Когда класс реализуется, создаваемый объект имеет переменные этого класса, которые становятся доступными другим его объектам после указания имени. Нам уже знаком данный процесс. Однако существует возможность единожды задать члены для всех объектов класса. Такие члены объявляются как статические (`static`), а доступ к ним осуществляется путем указания имени их класса с точкой (т.е. имя класса.член_класса). Их называют также **членами класса**. Описание модификатора `static` вы найдете в разделе 3.1.

Рассмотрим два примера. Класс `java.lang.Math` (см. раздел 3.1) имеет несколько полезных методов для выполнения математических вычислений, при которых объекты не создаются: исходные значения указываются посредством параметров, а ответы возвращаются как результаты выполнения методов, например:

```
y = Math.sin(x);
```

Для указания переменных, значения которых ассоциируются с классом, надлежит использовать модификатор `static`. Например, в классе `Curio` (вспомните программу о продаже древностей) мы можем определить общий объем партии товара и количество проданных единиц за некоторый период времени. Структура нового класса `Curio` будет следующей:

```
class Curio {
  // Внутренний класс, предоставляющий сведения о древности
  // и выполняемых по отношению к ней операциях.
  // Расширен параметром stock
  // и двумя методами для добавления и продажи товара (stock).
  String name;
  int price;
  String description;
  int Stock;

  static int totalStock = 0;
  static int totalSold = 0;
```

```
  // Конструктор копирует начальные значения.
  Curio (String n, int p, String d, int t) {
    name = n;
    price = p;
    description = d;
    display.prompt (name,t);
  }

  // Три новых действия, выполняемых по отношению к древностям.
  void addStock (int n) {
    stock += n;
    totalStock += n;
  }

  void sell (int n) throws StockException {
    if (stock >= n) {
      stock -= n;
      totalStock -= n;
      totalSold += n;
    }
    else
      throw new StockException
        ("Недостаточно товара. В наличии имеется "+stock+" единиц. ");
  }
 … другие методы как и раньше
}
```

В программе, которая использует класс `Curio`, мы можем теперь сделать так, чтобы класс `CurioShop` ссылался на суммарные значения (`totalStock`, `totalSold`):

```
display.println("Общий объем товара: "+Curio.totalStock+
        " продано " + Curio.totalSold);
```

Со статическим методом связан ряд сложностей. Если метод вызывается из статического метода, то он тоже должен быть статическим. Это касается и всех переменных, объявленных в классе: если они используются статическим методом, то должны быть статическими. Поэтому важно не применять статические методы для структурирования программы. Проблема состоит в том, что метод `main` должен быть статическим. Следовательно, все, что он вызывает или использует, также должно быть статическим. По этой причине мы в книге сделали так, чтобы главный метод вызывал только конструктор и в анонимной форме, не обращаясь к переменным, объявлял новый объект.

Константы

Константы в Java определяются с помощью модификаторов `static final`:

```
static final retirementAge = 60;
```

Модификатор объявления `static` делает элемент членом класса. Следовательно, объявления констант могут происходить только на уровне класса, а не внутри метода, даже такого, как `main`. Второй модификатор — `final` — указывает, что содержимое поля не может быть изменено в процессе выполнения программы (другими словами, это — константа).

Когда объявляются переменные, мы можем или конкретизировать первоначальное значение, или использовать **инициализацию по умолчанию**. При использовании

числовых типов по умолчанию назначается ноль. Программа будет более надежной, если явно задавать для каждой переменной стартовое значение. Чем такая инициализация отличается от объявления константы? Тем, что, будучи объявленной и инициализированной, константа уже не может быть изменена. В переменных же значения могут изменяться посредством оператора присваивания.

Массивы и объекты также могут быть константами. Например, следующие операции являются действительными:

```
static final int daysInMonth [] = {31.28.31.30.31.30.31.31.30.31.30.31};
static final Time midnight = new Time (12, 0);
```

Внутренние классы

В Java 1.1 классы получили статус "высшей категории", благодаря чему они могут быть объявлены где угодно и могут использоваться в качестве параметров и как локальные члены других классов, методов и блоков. В табл. 8.5 представлены четыре вида внутренних классов.

Таблица 8.5. Виды внутренних классов

Тип внутреннего класса	Определяется как	Используется для	Комментарии
Вложенный верхнего уровня	Статический член другого класса	Группирования классов	Имя включает имя класса, внутри которого он находится
Член	Рядовой член другого класса	Вспомогательного класса или членов, используемых классами совместно	— Предоставляет доступ к внешнему экземпляру другого класса — Особый синтаксис для специальных переменных *this* и *super* оператора new — Не может иметь статических членов
Локальный	Член метода блока	Адаптации класса для передачи в качестве обратной связи (например, при обработке событий)	Те же комментарии, что и для члена
Анонимный	Безымянный член метода или блока	Создания "одноразовых" объектов, определяемых и реализуемых там, где это требуется	Те же комментарии, что и для члена — Нет имени или конструктора — Реализуется (создается экземпляр объекта) только один раз — Определяется в выражении

Вложенный класс похож на остальные классы с той разницей, что располагается внутри другого класса, а это упрощает создание такового и работу с ним. Во многих случаях неудобно помещать каждый класс в отдельный файл. Благодаря вложенности классов мы можем содержать их вместе. Если класс объявляется статическим, значит, он будет иметь тот же статус высшего уровня, что и внешний класс. Вложенные классы могут быть импортированы из своего класса, что похоже на организацию пакетов и классов в Java.

Классы-члены являются типичными вспомогательными классами для внешнего класса. Они не видны снаружи, но могут использовать все члены внешнего класса — закрытые или нет. Имена внутреннего и внешнего классов не могут совпадать, однако допускается назначение одинаковых имен членам различных внешних классов. Описанный принцип используется в примере 8.2: класс `Coffee` находится внутри класса `LinkTest`. В нескольких других рассмотренных программах такие классы, как `Curio` и `Rates`, могли быть включены в главные классы `CurioStore` и `Converter`.

Ввиду способности использовать как свои собственные члены, так и члены внешнего класса, внутренние классы могут различать две версии `this`. Безымянное `this` относится к классу-члену, а для того чтобы обратиться к внешнему классу, мы указываем имя внешнего класса в качестве приставки к `this`. Аналогичные расширения выполняются для оператора `new` и специальной переменной `super`. Заслуживает внимания следующее ограничение, присущее классам-членам: они не могут иметь статических членов. Любые статические члены (включая константы), необходимые классу-члену, должны быть сгруппированы в классе высшего уровня.

Анонимные классы выполняют те же функции, что и локальные классы, но с одним ограничением, согласно которому они не имеют имени и, следовательно, могут быть реализованы только единожды — там, где определены. Они эффективны в сочетании с обработчиками событий Java, которые описаны в главе 11. Вот типичный пример использования анонимного класса:

```
f.addWindowListener(new WindowAdapter () {
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
});
```

Интерфейс `WindowAdapter` (см. раздел 9.2) должен иметь уже определенный метод `WindowClosing`, чтобы его можно было вызвать в подходящий момент для окна, которое здесь называется `f`. Поэтому мы создаем новый анонимный класс как параметр метода `addWindowListener`. В качестве параметра передается целый класс, а в соответствующий момент, как определено выше, может быть вызван метод `windowClosing`. Обширную информацию о применении анонимных классов вы найдете в главе 10.

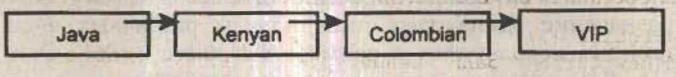
Локальные классы в значительной степени подобны анонимным за исключением того, что они имеют имена и могут использоваться многократно.

8.3. Списки и итераторы

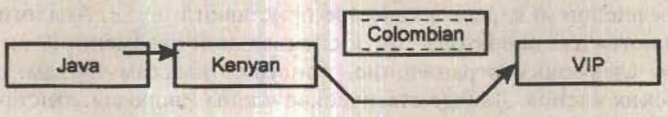
Один из недостатков массивов заключается в том, что их размер и структура жестко фиксируются. Невозможно удалить элемент без последующего смещения индексов оставшихся элементов к началу. Хеш-таблицы (см. главу 6) избавлены от этих ограничений, но зато они характеризуются несколько необычным способом хранения данных. Классической альтернативой массиву является **связный список.** В этом разделе мы остановимся на его описании и рассмотрим, как он может быть использован в примере 8.1.

В массиве элементы хранятся последовательно, т.е. элемент `i+1` следует за элементом `i`. В связном списке элементы рассматриваются как индивидуальные и при этом соединены ссылками. Такие ссылки можно разрывать и создавать вновь, благодаря чему мы получаем структуру, которая в отличие от массива может увеличиваться или уменьшаться по мере выполнения программы.

Предположим, несколько строковых объектов объединены в связный список, например:



Чтобы удалить объект "Colombia", достаточно перенаправить ссылки, как показано на следующем рисунке. В результате область памяти, которую занимал ненужный объект, будет очищена.



Аналогичным образом можно добавить новый объект в любую позицию списка.

Списковые классы Java

Опираясь на пример 8.2, можно было бы определить класс, который обрабатывает списки подобным образом. Однако при переходе от Java 1.1 к Java 1.2 большое количество классов было добавлено в пакет `java.util`, который обеспечивает работу со списками различных форм. Далее мы поговорим о том, как использовать эти классы, и с их помощью модифицируем программу `CoffeeShop`.

Для обозначения всех видов списков и других структур употребляется термин **коллекция**. Подробный материал о коллекциях будет предложен вашему вниманию в главе 15, а на данном этапе мы продолжим разговор о списках. В Java под общим заголовком `AbstractList` предоставляются следующие четыре класса: `LinkedList`, `ArrayList`, `Vector` и `Stack`. Класс `Vector`, с которым мы сталкивались в главе 6, содержит индексированную последовательность объектов, для которой фиксированный размер не является обязательным требованием. Класс `Stack` происходит от класса `Vector` и имеет ограничение, согласно которому добавление и удаление разрешается производить только с одного конца. Класс `ArrayList` во многом похож на класс `Vector`. В наибольшей степени нашим требованиям соответствует класс `LinkedList`, поскольку эффективным способом выполнения операции удаления является, как мы уже знаем, перенаправление ссылок.

Ниже представлена спецификация класса `LinkedList`:

Класс `LinkedList` (сокращенный)

`void clear ()`
`void add (Object item)`
`boolean contains (Object item)`
`boolean remove (Object item)`
`int size ()`
`// Плюс еще 20 классов, относящихся главным образом к индексированным спискам.`

Метод `clear` удаляет любые элементы, которые были добавлены к объекту `LinkedList`. Метод `add` добавляет объекты в конец списка. Метод `contains` ищет объект, переданный методу в качестве параметра, и возвращает значение `true` или `false`. Он базируется на методе сравнения, определенном для объектов. Метод `remove` удаляет первый встреченный объект со значением, равным параметру `item`.

Добавление элементов в конец списка во многих случаях является подходящим решением. Их можно вставлять и в середину списка. Посмотрим, как это делается.

Мы можем составить список древностей следующим образом:

```
LinkedList curioList ();
curioList.add(new Curio("Традиционные кружки", 6,
       "обрамленные в стиле Ndebele",20));
curioList.add(new Curio("Футболки", 30, "размеры от M до XL", 50));
curioList.add(new Curio("Маски", 80, "вырезанные из дерева", 8));
```

Если мы определим равенство различных объектов `Curio` с помощью переменной `name`:

```
public boolean equals (Curio c) {
  return name.equals(c.name));
}
```

то сможем использовать метод `contains` для проверки объекта `Curio` и удаления его из списка.

```
System.out.println("Футболки"+"имеющиеся в магазине:"+
       curioList.contains(new Curio("Футболки",0,"", 0));
curioList.remove(new Curio("Футболки",0,"", 0));
```

Итераторы списка

Как же осуществляется перемещение по списку? Предположим, список должен быть выведен целиком. В предыдущем описании мы преднамеренно не указали методы, которые используют индексы (как должно быть в массиве), чтобы у вас возник вопрос о том, как мы по очереди получаем каждый элемент. Ответ: с помощью итератора. **Итератор** определяет позицию элемента в списке и может быть использован в операторах `for`, `while` или `do` для организации цикла, в котором по очереди просматриваются все пункты списка. Итераторы являются интерфейсами и предоставляют дополнительные методы классу объектов, посредством которых они могут осуществлять итерацию. Примером такого класса является `LinkedLists`. Они очень похожи на перечисления, которые были представлены в разделе 6.5.

Для создания итератора используется следующая запись:

Создание итератора
`ListIterator listiter = list.listIterator();`
При наличии списка `listiter` делает доступными методы, приведенные ниже.

После этого объявления нам будут доступны следующие методы, определенные в классе `ListIterator`.

Класс `ListIterator`
boolean hasNext ()
boolean hasPrevious ()
Object next ()
Object previous ()
void add (Object item)
void remove ()
void set (Object item)

Методы `hasNext` и `hasPrevious` проверяют, достигнут ли конец списка, в двух направлениях: либо с начала, либо с конца.
Методы `next` и `previous` перемещают итератор вперед или назад.
Методы `add`, `remove` и `set` предоставляют безопасные средства для изменения списка во время перемещения по нему. Метод `add` внедряет объект, переданный ему в качестве параметра, непосредственно перед объектом, который должен быть возвращен при следующем вызове метода `next`. Метод `remove` удаляет объект, который был возвращен во время последнего вызова метода `next`, а метод `set` изменяет значение следующего объекта.

Итерация списка с использованием цикла `for` описана ниже.

Оператор цикла `for` с итератором списка

```
for (ListIterator listiter = list.listiterator(); listiter.hasNext(); ) {
  item = listiter.next();
  body
}
```

В качестве примера рассмотрим список простых объектов, которые имеют только числовое поле. Для заполнения списка случайными числами до тех пор, пока не будет обнаружено число 3, мы должны сделать следующее:

```
Random r = new Random ();
LinkedList list = new Linkedlist ();
 int number = 0;
while (number !=3) {
  number = r.nextInt();
// Добавление в конец списка.
  list.add(new Integer (number));
}
```

Для вывода списка на печать необходимо использовать итератор в цикле `for`:

```
for (ListIterator listiter = list.listiterator();
listiter.hasNext(0);)
  System.out.println(listiter.next());
```

Усложним задачу. Предположим, нужно добавить единицу к каждому числу в списке. Возможное решение — применение метода `set`, как в следующем цикле:

```
for (ListIterator listiter = list.listiterator();
 listiter.hasNext(0);) {
  number = listiter.next().intValue();
  number++;
  // Заменяет объект, возвращенный методом next.
  listiter.set(new Integer(number));
}
```

Связные списки и итераторы позволят нам расширить возможности программы `CoffeeShop`.

8.4. Практикум 3. Кофейный магазин Нельсона

Исходные условия

От кофейного магазина Нельсона поступил заказ сделать учет запасов кофе различных сортов и партий динамическим. Сорта и партии не должны указываться заранее, следует обеспечить возможность удалять записи о них, если соответствующий товар полностью продан. Решить эту задачу нам поможет класс `Linkedlist`.

Алгоритмы

Алгоритмическая часть программы связана главным образом с организацией продаж. Данные о них будут вводиться из файла заказов, в котором перечисляются сорта кофе и указывается количество каждого из них, в килограммах. Программа должна найти соответствующий сорт кофе в списке и вызвать метод, "осуществляющий" продажу, который определен в классе `Coffee`. Следует отметить, что возникновение ошибки возможно в двух ситуациях:

- если не найден сорт кофе;
- если в фондах нет нужного количества кофе.

В каждом из описанных случаев программа выводит соответствующее сообщение.

Итак, мы приступаем к описанию продаж. Следует отметить, что соответствующий алгоритм (рис. 8.7) значительно сложнее, чем может показаться сначала, так как необходимо продавать партии последовательно и переходить от одной к другой, если для выполнения заказа нужно найти еще несколько килограммов кофе. Остальные фрагменты программы либо похожи, либо являются более простыми.

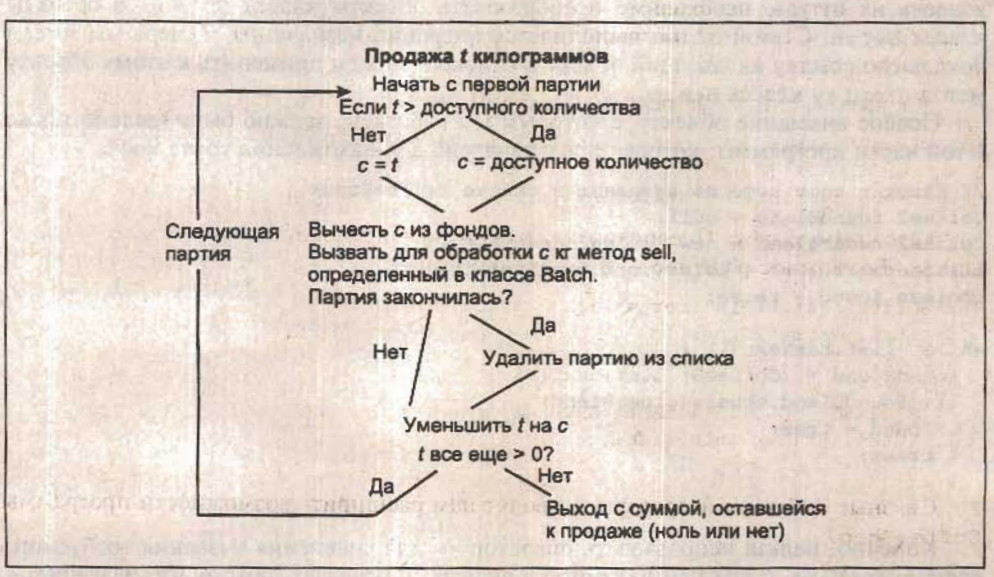


Рис. 8.7. Алгоритм продажи кофе

Преобразование программы

При использовании класса Batch не имеет значения, с чем ведется работа — с массивами партий или со списками партий. Класс Coffee надлежит немного изменить. Новому классу мы присвоим имя Coffee2. В классе Coffee массив заменим связным списком:

```
private LinkedList batchStock = new LinkedList ();
```

В результате будет создан объект, который может быть изменен методами, определенными в классе LinkedList, а также методами итераторов списка. Поскольку счетчик noOfBatches больше не нужен, он устраняется. Добавление новой партии осуществляется путем вызова метода add:

```
batchStock.add(new Batch (nextStock,stockDate));
```

В цикле, который расположен в конце программы и предназначен для вывода информации о всех партиях, будет использоваться итератор списка:

```
for (ListIterator list = batchStock.listIterator(); list.hasNext(0; ) {
  Batch batch = (Batch) list.next();
  batch.display();
}
```

Причина, по которой необходим дополнительный локальный объект batch, заключается в следующем: объект batchStock представляет собой экземпляр класса LinkedList и имеет доступ ко всем методам класса LinkedList. Все они определены для экземпляров суперкласса Object. Нет ничего, что бы связывало список LinkedList класса Coffee2 с классом Batch. Мы поместили в список партии. Чтобы извлечь их оттуда, необходимо преобразовать объекты класса Object в объекты класса Batch. С этой целью выполняется операция назначения. Теперь мы имеем локальную ссылку на текущий объект в списке и можем применить к этому объекту метод display класса Batch.

Особое внимание объекту, с которым мы работаем, должно быть уделено также в той части программы, которая предназначена для нахождения сорта кофе:

```
// Находит сорт кофе по названию в списке coffeeStock.
Coffee2 foundBlend = null;
Coffee2 orderBlend = new Coffee2(orderName);
ListIterator list = coffeStock.listIterator();
boolean found = false;

while (list.hasNext()) {
  foundBlend = (Coffee2) list.next();
  if (foundBlend.equals(orderBlend)) {
    found = true;
    break;
  }
}
```

Конечно, нельзя использовать оператор == для сравнения названия требуемого кофе orderName с каждым из сортов в фондах. Во-первых, потому что название — это строка, а сорта кофе представляют собой объекты класса Coffee2. Следовательно, мы вынуждены создать временный объект orderBlend класса Coffee2 (третья строка). А во-вторых, потому что, мы, как и прежде, должны быть уверены, что элемент, считываемый из списка, определен как объект Coffee2. Вот почему

вызывается метод equals, определенный для класса Coffee2. Он просматривает только названия сортов, а не цены или какие-либо другие данные (автор применяет слово *blend*, которое переводится как *смесь сортов*).

Вас может заинтересовать, почему мы не используем метод contains, который существует в классе LinkedList. Причина заключается в том, что метод contains возвращает в качестве ответа только "да" или "нет" и не предоставляет ссылку на объект в списке. Его позиция может быть установлена с помощью некоторых методов индексации класса LinkedList, но они не рассматривались в данной книге. Вы можете найти эти методы в документах по Java API в Internet.

Список объектов Coffee2 создается таким же образом, как и список партий, с использованием метода add класса LinkedList.

Главные классы системы представлены на рис. 8.8. Пунктир указывает на то, что класс имеет переменные, показанные на диаграмме. Следовательно, класс CoffeeShop2 имеет объектные переменные LinkedList, Coffee2 и Stream. Стрелки служат признаком того, что выполняется импортирование из двух пакетов, включенных в программу. Интерфейс ListIterator ассоциирован с классом LinkedList.

Имеются и другие классы, которые здесь не показаны. Например, класс Date, который импортирован из пакета java.util и применяется тремя главными классами, а также класс DateFormat, импортированный из пакета java.text.

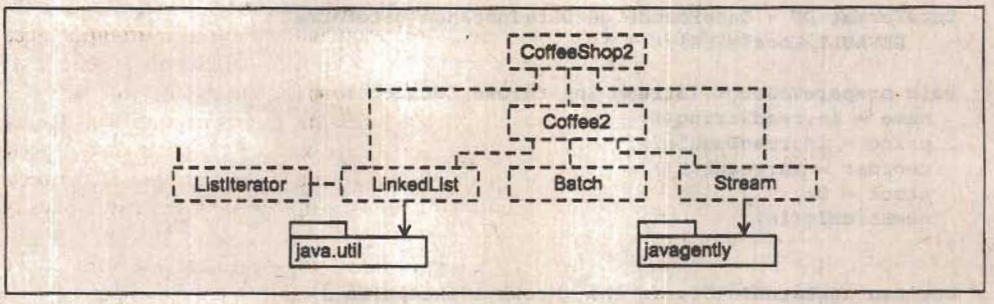


Рис. 8.8. Классы программы CoffeeShop. Знак "I" над ListIterator является обозначением интерфейса (см. раздел 9.3)

Программа

Как же должны быть организованы три указанных класса? CoffeeShop2 — это класс высшего уровня, главный класс в программе. Следует ли объявлять внутренними классы Coffee2 и Batch? Нет. Класс Batch является наиболее вероятным кандидатом на то, чтобы быть вложенным, но вспомним, что он применяется в предыдущей версии программы, в связи с чем остается отдельным, хотя это и не обязательно. (Если бы он являлся вложенным, то был бы доступен для других классов.) Чтобы упростить внесение изменений и компиляцию, оставим класс Coffee2 без изменений. Итак, программа находится в трех файлах, для каждого из которых нужно выполнить компилирование:

```
import java.io.*;
import java.util.*;
import javagently.Stream;
import java.text.*;
class Coffee2 {
```

```
/* Улучшенный класс Coffee. Автор - Дж. М. Бишоп, октябрь 1996 г.
 * - - - - - - - - - - - - - -  Java 1.1 - октябрь 1997 г.
 *                              Обновление - август 2000 г.
 *                              Java 2 - май 2000 г.
 * Сохраняет данные, касающиеся инвентаризации партий кофе
 * конкретного сорта или смеси сортов, с помощью класса
 * LinkedList пакета java.util.
 */
private String name;
private double price;  //за кг
private double stock; //кг (kgs)
private double reorder; //кг (kgs);
private LinkedList batchStock = new LinkedList ();

public Coffee2() {
  /* Пустой конструктор, поскольку все значения
   * вводятся с помощью метода repareToStock.
   */
}
public Coffee2(String s) {
  name = new String(s);
}

DateFormat DF = DateFormat.getDateInstance(DateFormat.
    DEFAULT,Locale.UK);

void prepareToStock (Stream in) throws IOException {
  name = in.readString();
  price = in.readDouble();
  reorder = in.readInt();
  stock = 0;
  newBatchIn(in);
}

boolean newBatchIn(Stream in) throws IOException {
  double nextStock = in.readDouble();
  if (nextStock == 0)
    return false;
  else {
    stock += nextStock;
    Date stockDate = new Date ();
    String dateString = in.readString();
    try {
      stockDate = DF.parse(dateString);
    } catch (ParseException e) {
      System.out.println("Ошибка в дате " + dateString);
    }
    batchStock.add(new Batch(nextStock, stockDate));
    return true;
  }
}

double sell(double ordered) {
  double kgInBatch;
  Batch currentBatch;

  for (ListIterator list =
```

```
      batchStock.listIterator(); list.hasNext(); ) {
      currentBatch = (Batch) list.next();
      if (ordered > currentBatch.available())
        kgInBatch = currentBatch.available();fs24 R      else
        kgInBatch = ordered;
      stock -= kgInBatch;
      boolean soldOut = currentBatch.sell(kgInBatch);
      if (soldOut) {
        System.out.println("Партия исчерпана ");
        list.remove();
      }
      ordered -= kgInBatch;
      if (ordered <= 0)
        break;
    }
    System.out.println("Отчет по фондам: ");
    display();
    return ordered;
  }

  boolean equals(Coffee2 c) {
    return name.equals(c.name);
  }

  void display() {
    System.out.println(name+" @ G"+
      Stream.format(price,1,2)+" за кг");
    for (ListIterator list =
        batchStock.listIterator(); list.hasNext(); ) {
      Batch batch = (Batch) list.next();
      batch.display();
    }
    System.out.println(Stream.format(stock, 1, 1)+" кг в фондах."+
          "Минимальный объем поставки равен "+ reorder +" кг\n");
  }
}

import java.io.*;
import java.util.*;
import javagently.Stream;
import java.text.*;

 class CoffeeShop2 {
  /* Программа CoffeeShop, версия 2. Автор - Дж. М. Бишоп, октябрь 1996 г.
   * - - - - - - - - - - - -  - -  Java 1.1 - октябрь 1997 г.
   *                              Java 1.2 - май 2000 г.
   * Предусмотрена возможность продажи партий кофе
   * конкретного сорта, начиная с наиболее старой.
   *
   * Демонстрирует использование классов LinkedList и
   * ListIterator пакета java.util.
   */
  CoffeeShop2 () throws IOException {

    LinkedList coffeeStock = new LinkedList ();
    stockUp(coffeeStock);
    inventoryReport(coffeeStock);
```

```
    makeSales(coffeeStock);
  }

  void stockUp(LinkedList coffeeStock) throws IOException {
    Stream in = new Stream("coffee.dat",Stream.READ);
    System.out.println("Фонды магазина");
    while (true) {
      try {
        Coffee2 coffee = new Coffee2();
        coffee.prepareToStock(in);
        coffeeStock.add(coffee);
        boolean morebatches = true;
        while (morebatches)
          morebatches = coffee.newBatchIn(in);

      } catch (EOFException e) {
          System.out.println("Фонды магазина на сегодня");
          System.out.println();
          break;
      }
    }
  }

  void inventoryReport(LinkedList coffeeStock) {
    System.out.println("Кофейный магазин Нельсона");
    System.out.println("Инвентаризационный контроль");
    System.out.println("Имеется в наличии на "+
      DateFormat.getDateInstance().format(new Date(DateFormat.FULL)));
    for (ListIterator list =
        coffeeStock.listIterator(); list.hasNext();) {
      Coffee2 current = (Coffee2) list.next();
      current.display();
    }
  }

  void makeSales(LinkedList coffeeStock) throws IOException {

    Stream orderFile = new Stream ("orders.dat",Stream.READ);
    System.out.println("Продажа кофе");

    while (true) {
      try {
        String orderName = orderFile.readString();
        double orderAmount = orderFile.readDouble();
        System.out.println("Заказ на "+orderAmount +" кг кофе сорта "+
                           orderName);
        System.out.println("============================");

        // Проверка наличия сорта кофе.
        Coffee2 foundBlend = null;
        Coffee2 orderBlend = new Coffee2(orderName);

        ListIterator list = coffeeStock.listIterator();
        boolean found = false;

        while (list.hasNext()) {
          foundBlend = (Coffee2) list.next();
```

```
            if (foundBlend.equals(orderBlend)) {
              found = true;
              break;
            }
          }

          if (!found)
            // Нет такого названия.
            System.out.println("Извините, у нас нет такого сорта.\n");
          else {
            // Начало продажи.
            double remainder = foundBlend.sell(orderAmount);
            if (remainder > 0) {
              System.out.println("Может не хватить на "
                 +remainder+" кг\n");
            }
          }

        } catch (EOFException e) {
            System.out.println("\nМагазин закрыт");
            break;
        }
      }
    }

    public static void main (String [] args) throws IOException {
      new CoffeeShop2 ();
    }
}
```

Тестирование

Первоначальный файл данных был таким:

```
Colombian
59.95 10
45.5 23-Jun-00
5 30-Jul-00
0
Kenyan
60.75 10
15 19-May-00
0
Java
63.50 30
61.88 23-May-00
20 12-Jul-00
3 1-Aug-00
0
```

Добавим следующий файл заказов:

```
VIP 2
Kenyan 15
Kenyan 2
Java 70
Colombian 46
Colombian 3
```

Сначала мы смоделируем ситуацию, когда название сорта, указанное в первом заказе, не найдено. Затем зададим, что весь кофе сорта Kenyan продан при выполнении второго заказа, вследствие чего третий заказ не может быть реализован. Для выполнения заказа на кофе сорта Java, как и заказа на кофе Colombian, потребуется взять кофе из двух партий. Последний заказ будет выполнен по схеме было-продали-осталось. Результат работы программы получится следующим:

```
Фонды магазина
Фонды магазина на сегодня

Кофейный магазин Нельсона
Инвентаризационный контроль
Имеется в наличии на 01.01.1970
Colombian @ G59,95 за кг
      23-Jun-00 : 45,5 кг
      30-Jul-00 :  5,0 кг
50,5 кг в фондах.Минимальный объем поставки равен 5.0 кг

Kenyan @ G60,75 за кг
      19-May-00 : 15,0 кг
15,0 кг в фондах. Минимальный объем поставки равен 10.0 кг

Java @ G63,50 за кг
      23-May-00 : 61,8 кг
       12-Jul-00 : 20,0 кг
       01-Aug-00 :  3,0 кг
84,8 кг в фондах. Минимальный объем поставки равен 3.0 кг

Продажа кофе
Заказ на 2.0 кг кофе сорта VIP
==============================
Извините, у нас нет кофе такого сорта.

Заказ на 15.0 кг кофе сорта Kenyan
==============================
Продано 15,0 кг из партии 19-May-00
Партия исчерпана
Отчет по фондам:
Kenyan @ G60,75 за кг
0,0 кг в фондах. Минимальный объем поставки равен 10.0 кг

Заказ на 2.0 кг кофе сорта Kenyan
==============================
Отчет по фондам:
Kenyan @ G60,75 за кг
0,0 кг в фондах. Минимальный объем поставки равен 10.0 кг

Может не хватить на 2.0 кг

Заказ на 70.0 кг кофе сорта Java
==============================
Продано 61,8 кг из партии 23-May-00
Партия исчерпана
Продано 8,2 кг из партии 12-Jul-00
```

```
 Отчет по фондам:
Java @ G63,50 за кг
      12-Jul-00 : 11,8 кг
      01-Aug-00 :  3,0 кг
14,8 кг в фондах. Минимальный объем поставки равен 3.0 кг

Заказ на 46.0 кг кофе сорта Colombian
==============================
Продано 45,5 кг из партии 23-Jun-00
Партия исчерпана
Продано 0,5 кг из партии 30-Jul-00
 Отчет по фондам:
Colombian @ G59,95 за кг
      30-Jul-00 :  4,5 кг
4,5 кг в фондах. Минимальный объем поставки равен 5.0 кг

Заказ на 3.0 кг кофе сорта Colombian
============================
Продано 3,0 кг из партии 30-Jul-00
Отчет по фондам:
Colombian @ G59,95 за кг
       30-Jul-00 :  1,5 кг
1,5кг в фондах. Минимальный объем поставки равен 5.0 кг


Магазин закрыт
```

Мы детально объяснили, как в Java используется класс `LinkedList`. В главе 15 дан обзор нескольких похожих классов, предоставляемых языком Java. Одна из основных целей книги — научить читателей основополагающим принципам программирования. Одним из таковых считается применение интерфейсов API из библиотеки. Возможно, вы не сможете с первого раза освоить приемы программирования класса, создающего списки, но наверняка поймете, как он работает, и сможете применить эти знания в будущем.

Резюме

Данная глава является важнейшей в плане приобретения навыков программирования. Она включает детальные описания объектов, информацию о принципах их использования в сложной программе, а также материал о структурах данных, альтернативных массивам, т.е. о списках. В главе также изложены детальные сведения о модификаторах защиты и внутренних классах, продемонстрированы отношения между пакетами, классами и методами.

Объявление объекта приводит к созданию ссылки на объект, в то время как значения примитивных переменных сохраняются и передаются непосредственно. Согласно этой модели, сравнение и копирование объектов должно осуществляться с помощью методов `equals` и `clone`. Метод `clone` основывается на суперклассе `Object`, что требует преобразования классов.

Средством хранения последовательностей данных, альтернативным массивам, являются списки. Их преимущество состоит в том, что они могут динамически расширяться или сокращаться по мере добавления или удаления элементов. Для составления списков в Java предназначены классы `LinkedList` и `ListIterator`.

В главе раскрыты понятия, связанные со следующими ключевыми словами:

```
this
final
static
```

Представленные ниже ключевые слова упоминались, но подробно будут рассмотрены позже:

```
protected
abstract
synchronized
```

Глава содержит описания ряда интерфейсов API Java:

```
Object          contains
clone           remove
LinkedList      size
add
```

Следующий перечень включает темы, которым в главе уделено особое внимание:

- объявление клона (клонированного объекта);
- класс `LinkedList` (сокращенный);
- создание итератора;
- класс `ListIterator`;
- оператор цикла `for` с итератором списка;
- модификаторы;
- внутренние классы.



Вопросы

8.1. В первой программе Coffee Shop (пример 8.1) компилятор не позволит осуществить доступ к полю `C[0].name`, потому что:

а) объект `C[0]` не был реализован (создан)

б) имя объявляется как закрытое

в) 0 не является действительным индексом для массива

г) значение для имени еще не было введено

8.2. Если задано клонирование для класса `Time`, а переменная `today` — это объект класса `Time`, какая из следующих операций будет действительной?

а) `Time temp = (Time) clone(today);`

б) `Time temp = Time.clone(today);`

в) `Time temp = (Time) today.clone();`

г) `Time temp = today.clone();`

8.3. Класс `Linked List` содержит метод `contains`, который используется в выражении `L.contains(x)`. Работа этого метода основана на том, что:

а) имеется хеш-таблица объектов

б) для объектов списка определен метод `equals`

в) в списке есть объект `x`

г) в списке нет объекта x

8.4. В чем состоит различие между методами add классов LinkedList и ListIterator?

а) различия нет

б) в позиции, где объект добавляется к списку

в) в том, следует ли обеспечивать приведение объекта

г) в том, можно ли будет впоследствии удалить объект

8.5. При просмотре списка L итератором списка Lit следующий элемент можно получить с помощью оператора:

а) `L.getNext();`

б) `L.next();`

в) `Lit.hasNext();`

г) `Lit.next();`

8.6. Если исходить из доступных для массива операций, он больше похож на:

а) объект

б) строку

в) примитивное значение

г) нечто отличное от всего перечисленного выше

8.7. Объявив метод `Coffe.equals()`, мы должны будем также:

а) объявить его открытым

б) объявить его закрытым

в) объявить клон для Coffee

г) ничего делать не нужно

8.8. Различие между вложенным классом N и классом-членом M состоит в том, что:

а) нет различия

б) M должен иметь конструктор

в) M видит члены своего внешнего класса

г) N не может иметь статических членов

8.9. Оба итератора ListIterator в классах Coffee2 и CoffeeShop2 при просмотре списков возвращают переменные типа:

а) Batch

б) Object

в) Coffee2

г) Date

8.10. В примере 8.2 переменная списка имеет значение:

а) coffees

б) Java

в) null

г) неопределенное

Задачи

8.1. **Интерактивные заказы.** Внедрите в программу из практикума 3 возможность использования интерфейса класса `Display` для ввода заказа.

8.2. **Покупки по кредитным карточкам.** В файле заказов некоторые заказы регистрируются как поступившие от покупателей, пользующихся кредитными карточками Gold card. Обладатели таких карточек каждый день имеют приоритет при обслуживании: их заказы выполняются немедленно без занесения в список. Модифицируйте программу `Coffee2` таким образом, чтобы она не заносила в список заказ, поступивший от владельца кредитной карточки Gold card.

8.3. **Упорядоченные партии.** Класс `Linkedlist`, используемый в программе "Кофейный магазин Нельсона", всегда добавляет новые элементы в конец списка. Данные в файле `orders.dat` упорядочены таким образом, что партии кофе, поступившие раньше, указываются в списке первыми и, следовательно, поступают в продажу первыми. Если партии поступают в различное время дня, невозможно гарантировать, что они будут указываться в списке в надлежащем порядке. Добавьте в класс `Batch` новый метод, который определял бы корректную позицию для новой партии, основываясь на указанном для нее крайнем сроке реализации. Затем напишите еще один метод (скажем, `insert`), который, учитывая дату, будет добавлять запись о новой партии, определяя для нее корректную позицию.

8.4. **Упорядочение сортов кофе.** Попробуйте, действуя по схеме, которая предложена в предыдущей задаче, представить сорта кофе в алфавитном порядке. Напишите метод для класса `Coffee2`, который будет находить корректную позицию для записи о новом сорте кофе, а затем воспользуйтесь тем же методом `insert` для его внедрения.

8.5. **База данных клуба.** Выберите вид деятельности, с которым вы знакомы, и разработайте класс и структуру объекта для хранения информации о людях, оборудовании или событиях, имеющих отношение к этой сфере. Добавьте методы для доступа к информации.

8.6. **Клуб желающих похудеть.** Взяв за основу задачу 3.3, усовершенствуйте класс так, чтобы для каждого персонажа было указано значение веса, которого этот человек хотел бы достигнуть. Затем составьте список лиц с проблемным весом, где бы присутствовали желаемые значения, и запустите программу. Когда кто-либо достигнет намеченного веса, поздравьте его и удалите из списка. Необходимо предусмотреть возможность интерактивного добавления записей о лицах в список.

8.7. **Марафон в Саванне.** Каждый год марафон в Саванне привлекает тысячи бегунов, но награды получают только 20 спортсменов, финишировавших первыми. Измените класс, разработанный в задаче 3.2, таким образом, чтобы в нем вместо строковых данных использовались номера бегунов. Это позволит генерировать данные в программе (включая значения времени) случайным образом. Сформируйте связный список, содержащий записи приблизительно о 1000 бегунах. Затем задайте просмотр списка в цикле для поиска 20 лучших результатов, а также вывод этих данных.

8.8. **Информация о погоде.** В примере 6.6 для сохранения данных объявляется матрица заданного размера, поскольку указан и первый, и последний год наблюдения. Предположим, что мы не хотим вводить эти данные заранее.

Создайте диаграмму, демонстрирующую, как данные могут сохраняться в виде массива связных списков. Переделайте программу, используя связные списки.

8.9. **Издание книги.** Основываясь на задачах 6.5, 3.19 и 2.10, разработайте систему, которая генерирует рукопись книги, используя для этого связный список. Затем, оставив рукопись неизменной, создайте окончательную версию книги, клонируя каждую главу в новый связный список. Программа должна быть интерактивной, т.е. допускать внесение изменений в рукопись в любой момент времени и обеспечивать автоматическое обновление окончательной версии книги.

8.10. **Магазин "Лавка древностей".** Вернемся к программе `CurioShop`. Используя в качестве шаблона класс `CoffeeShop`, создайте такую программу, чтобы данные можно было вводить каждый день, а поступление заказов происходило как из файла, так и интерактивным способом. Обработайте заказы надлежащим образом. Отметьте уровни поставки и обеспечьте регулярный вывод отчетов.

ГЛАВА 9



Абстракция и наследование

Объектно-ориентированное программирование позволяет эффективно использовать классы различными способами. Мы интенсивно применяли классы при разработке **композиции**, когда в классе создается объект, базирующийся на другом классе. Примером может служить массив партий кофе, содержащийся в классе `Coffee`. Два других способа, которые мы будем изучать в этой главе, называются **абстракцией** и **наследованием**.

9.1. Способы использования классов

Абстракция

Абстракция позволяет программисту сконцентрироваться на целях разработки класса — требуемых методах, пользовательском интерфейсе. Затем, уже на завершающем этапе, он сможет конкретизировать эти методы и вводить нужные параметры. Мы можем рассматривать это как усовершенствованный способ передачи параметров на уровне классов. В Java существует два средства обеспечения абстракции — **абстрактные классы** и **интерфейсы**.

В предыдущих программах при создании классов использовались характеристики реальных объектов (это были товары, сорта и партии кофе). Для выполнения такой операции, как сортировка массивов древностей, массивов баллов и массивов партий кофе, до этого времени мы должны были создавать три различных метода сортировки или преобразовывать все элементы в экземпляры суперкласса `Object`. Мы рассматривали, как использовать для этой цели объекты Object и хеш-таблицы, но это не идеальное решение, поскольку теряется базисное определение первоначального класса.

Согласно концепции абстрактных классов и интерфейсов в них должны существовать абстрактные методы, что позволит определять и использовать методы с общими именами. Метод может быть внедрен различными способами. Различия между этими двумя технологиями будут проясняться по мере изложения материала.

Наследование

Освещая тему **наследования**, мы сосредоточим внимание на определении класса, в основе которого будут данные и характеристики, знакомые практически всем, и оставим открытым вопрос определения его дополнительных версий. Эти версии будут наследовать свойства и характеристики первоначального класса, благодаря чему не придется повторять эти параметры в спецификациях версий. Мы формируем таким способом иерархии классов и можем на нужном уровне сфокусировать внимание на изменениях и дополнениях. Такая технология называется **прирастающим развитием** и является важным аспектом наследования (см. "Практикум 4").

Еще один аспект наследования — **специализация**. Поскольку все классы в иерархии принадлежат к одному и тому же семейству, они имеют одинаковый тип в механизме определения типов Java, вследствие чего один класс может использоваться вместо другого. Рассмотрим схему, представленную на рис. 9.1. Как и в главе 1, мы будем исходить из того, что класс `Nature` является основой для объектов, соответствующих животным, деревьям и птицам. Животные делятся на травоядных (слоны и носороги) и хищников. Необходимо, чтобы поле (т.е. переменная), содержащее название животного или растения, находилось в самом верхнем классе — `Nature`, поскольку все животные, деревья и птицы должны иметь название. Туда же должен быть помещен метод, предназначенный для записи названия. Так, если нас интересует слон, и мы хотим записать название этого животного, метод, который должен быть для этого вызван, будет одним из методов класса `Nature`. Специфической чертой слона является длина его бивней. Следовательно, в классе `Elephant` нужно объявить переменную и метод для вывода соответствующего значения. На какой-то ступени иерархии должен вестись учет того, сколько травы съедает в день травоядное животное. Поскольку это касается всех травоядных, такая информация должна храниться в классе `Herbivores` (травоядные).

На рис. 9.2 показано, как воздействует на классы объявление объектов `Elephant` и `Rhino` на самом нижнем уровне иерархии. Каждый из объектов `jumbo` и `mafuta` имеет три переменные, причем последняя переменная в каждом случае различна. Остальные две наследуются из классов, находящихся выше в иерархии.

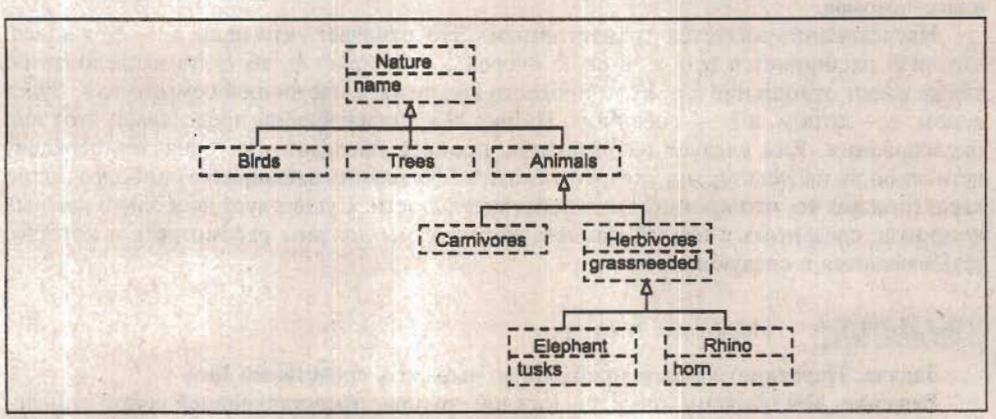


Рис. 9.1. Пример построения иерархии; на вершине находится класс `Nature`

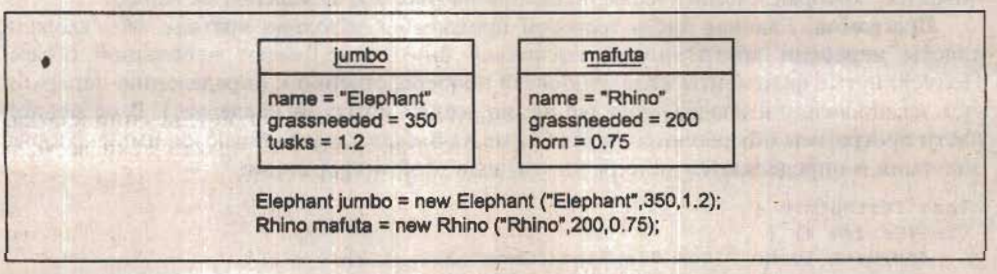


Рис. 9.2. Объявление объектов в иерархии `Nature`

Интерфейсы

Интерфейсы несколько отличаются от обычных классов. В интерфейсе только объявлены константы и методы, которые может внедрить любой другой класс. Все классы, которые реализуют данный интерфейс, доступны везде, где указывается имя интерфейса. Это не означает, что между объектами различных классов могут выполняться операции присваивания или они могут каким-то образом смешиваться. Можно сказать, что интерфейс, связанный с объектом, гарантирует определенное поведение этого объекта. Мы познакомились с тремя интерфейсами: `Enumeration` (раздел 6.5), `Cloneable` (раздел 8.2) и `ListIterator` (раздел 8.3). Если при создании метода, который должен работать с объектами, важным требованием является применение ими интерфейса `Cloneable`, следует в методе указать, что эти объекты имеют тип `Cloneable`. Примеры интерфейсов приведены в разделе 9.3.

9.2. Наследование

Рассмотрим практический пример наследования. Принято считать, что классы каждого нового уровня **расширяют** расположенный выше класс. Речь идет о том, что новый класс может иметь больше переменных и методов, чем первоначальный. Для обозначения классов, являющихся основой для создания новых классов, введены термины **суперкласс** или **класс-предок**, а для создаваемых классов — **подкласс**, или **класс-потомок**.

Наследование является транзитивным. Это означает, что если `a` — это класс, который расширяется `b`, а `c`, в свою очередь, расширяет `b`, то `c` по наследованию также имеет отношение к `a`. Если провести аналогию с настоящей семьей, то `a` будет дедом, `b` — отцом, а `c` — ребенком. На рис. 9.1. схематически представлен этот вид наследования. Как следует из названия, процесс наследования означает передачу чего-либо по наследству, но употребляемый в Java термин **расширение** наиболее полно характеризует то, что происходит в действительности. Существует несколько важных вопросов, связанных с наследованием, которые мы должны рассмотреть и которые затрагиваются в следующем примере.

Пример 9.1. Иерархия `Nature`

Задача. Иерархию `Nature` необходимо выразить средствами Java.

Решение. Мы объявим класс для каждой группы, представляющей собой определенную ступень в иерархии животного мира, от класса `Nature` к классам `Elephant` и `Rhino`. Затем мы напишем небольшую тестовую программу для создания двух объектов, которые схематически показаны на рис. 9.2, и зададим их вывод.

Программа. Главная часть тестовой программы довольно краткая. Мы сделали классы иерархии внутренними, поскольку они также имеют небольшой объем. Подчеркнутые фрагменты кода относятся непосредственно к определению иерархии (т.е. из объявления класса сразу же видно, какой класс он наследует). В остальной части программы объявляются переменные в объектах, присваиваются им некоторые значения и определяются методы для вывода этой информации.

```
class TestNature {
  TestNature () {
    Elephant jumbo = new Elephant("Слон",350, 1.2);
    Rhino mafuta = new Rhino("Носорог",200, 0.75);
    System.out.println(jumbo);
```

```
    System.out.println(mafuta);
  }

  public static void main (String [] args) {
    new TestNature ();
  }

  class Nature {
    protected String name;

    Nature (String n) {
      name = n;
    }

    public String toString () {
      return name;
    }
  }

  class Animals extends Nature {
    Animals (String n) {
      super (n);
    }
  }

  class Herbivores extends Animals {
    protected int grassneeded;

    Herbivores (String n, int g) {
      super (n);
      grassneeded = g;
    }
  }

  class Elephant extends Herbivores {
    private double tusks; // длина бивней
    Elephant (String name, int weight, double length) {
      super (name, weight);
      tusks = length;
    }
    public String toString () {
      return name+" "+grassneeded+" "+tusks;
    }
  }

  class Rhino extends Herbivores {
    private double horn; // длина рога
    Rhino (String n, int w, double l) {
      super (n,w);
      horn = l;
    }
    public String toString () {
      return name+" "+grassneeded+" "+horn;
    }
    // дополнительные данные о носороге
  }
}
```

Тестирование. Результат выполнения программы будет следующим:

```
Слон 350 1.2
Носорог 200 0.75
```

Защита при доступе

Мы уже работали с различными модификаторами защиты: `private`, `public` и `default` (см. разделы 4.6, 5.1 и 8.2). В рамках иерархии наследования существует необходимость в еще одном уровне доступа, похожем на задаваемый модификатором `private`. Любой член класса, который объявлен как `protected`, доступен в рамках своего класса и любого из его подклассов. В этой связи переменные `name` и `grassneeded` были объявлены как `protected`, а переменные `horn` и `tusks` — как `private`, поскольку в данный момент эти классы не имеют подклассов.

Суперконструирование

Полезной технологией в иерархии классов считается **суперконструирование**. Подкласс может вызвать конструктор класса-предка для инициализации тех переменных, за которые несет ответственность класс-предок. Это позволяет сосредоточиться на новых переменных и методах.

В примере 9.1 классам `Elephant` и `Rhino` передаются значения для всех трех параметров, но они полагаются на свои классы-предки в отношении того, какие значения должны быть присвоены переменным, которые в них не объявляются. Процесс выполняется снизу вверх. Переменной `w` управляет класс `Herbivores`, передающий переменную `n` классу `Animals`. Последний передает свой параметр классу `Nature`.

Обратите внимание на то, что вызов суперконструктора должен быть первым оператором конструктора.

Работа в рамках иерархии

Иерархия классов создается для того, чтобы мы имели возможность работать на различных уровнях. В рассматриваемом примере на самом верхнем уровне фиксируются названия живых существ. По мере перехода к более низким уровням увеличивается объем информации, которую мы можем обрабатывать требуемым образом. В рамках иерархии существует возможность свободно присваивать один объект другому. Другими словами:

- любой объект подкласса может быть присвоен объекту его суперкласса;
- любой объект суперкласса может быть присвоен объекту подкласса с соответствующим **приведением типа**.

После присвоения объект не меняется, изменяется только восприятие его типа компилятором. В следующем примере дан ряд операторов присвоения и прокомментирована реакция компилятора.

```
Animals a;

a = jumbo;          // правильно
mafuta = a;         // неправильно: ошибка    компилятора
mafuta = jumbo;     // неправильно: ошибка компилятора
mafuta = (Rhino) a; // правильно
```

Из примера видно, что мы не можем использовать в операторе присваивания объектные переменные классов, находящихся на разных уровнях. Между объектами

jumbo и mafuta существует значительная разница, поэтому нельзя присваивать их друг другу. Но мы можем работать с ними одновременно в рамках класса Herbivores или другого вышестоящего класса, что отражено в следующем примере:

```
Herbivores greedier;

if (jumbo.grassneeded > mafuta.grassneeded)
  greedier = jumbo;
else
  greedier = mafuta;
```

Следует отметить, что в Java имеется ограничение, согласно которому класс может расширять только один класс: наследование нескольких классов не поддерживается. Однако класс может включать несколько интерфейсов, что даст подобный эффект (см. следующий раздел).

Скрывающие переменные

Подкласс не только добавляет новые переменные к унаследованным данным, но и способен заменить имеющиеся данные собственной версией. Описанная операция выполняется путем объявления переменной с тем же именем, которое было дано классом-предком. Как правило, при этом преследуется цель изменить тип на какой-нибудь более подходящий. Когда объект создается, все переменные остаются, но та, которая подлежит замене, как бы маскируется. Данный процесс называется **сокрытием**. Например, если зоологи, изучающие слонов, захотят более точно описать их рацион, они могут решить использовать действительные числа для записи необходимого объема корма. Следовательно, к классу Elephant потребуется добавить:

```
double grassneeded;
```

Теперь при работе с данными о слонах следует использовать именно эту версию переменной. Другие классы будут получать первоначальную версию от класса Herbivores, где grassneeded является целым числом (т.е. имеет тип integer).

Скрытую переменную можно получить из класса Elephant, воспользовавшись префиксом super. Поэтому обращение из класса Elephant

```
super.grassneeded;
```

относится к первоначальной переменной из класса Herbivores.

Перегрузка методов

Там, где это имеет смысл, мы можем предоставить различные версии метода, оставив для каждой из них прежнее имя. Важно, чтобы списки формальных параметров различались. Только при соблюдении этого условия не возникнет путаницы при вызове методов. Характерным примером служат два метода format, объявленные в классе Display:

```
String format (int number, int align)
String format (double number, int align, int frac)
```

Переопределение методов

В подклассе можно определить другую версию метода, уже предоставленного суперклассом. В этом заключается суть **переопределения**. Мы будем использовать это средство во время определения в суперклассе главного метода или метода по

умолчанию. По мере продвижения вниз по иерархии мы сможем предоставить более специализированные версии.

Классическим примером является метод `toString`. Он находится в классе `Nature` и предназначен для вывода названия. Поскольку о конкретном животном имеется больше информации, чем в суперклассе, мы можем определить более детализированный метод `toString`, например такой, как в классе `Elephant`, который выводит на печать значения всех переменных, видимых на данном уровне.

Заметьте, что переопределение — это не то же самое, что **перегрузка**. Перегрузка предусматривает предоставление нескольких методов с одинаковыми именами, но разными списками параметров, благодаря чему Java рассматривает их как различные экземпляры. В процессе переопределения списки параметров те же, но методы относятся к различным классам, образующим иерархию. В следующем практикуме мы рассмотрим пример, посвященный наследованию в малой системе. Немало тематических примеров вы найдете и в следующей главе, поскольку наследование широко применяется в собственных пакетах Java.

Динамическое связывание

Переопределенные методы являются более мощным инструментом программирования, чем скрытые переменные, о чем свидетельствуют следующие их характеристики. Когда в иерархии есть несколько экземпляров метода, всегда используется тот, который находится в ближайшем подклассе. Это утверждение остается верным, даже если объект подкласса присвоен объекту суперкласса и из него вызывается метод подкласса. Рассмотрим ряд операторов, используя в качестве примера объект `jumbo`:

```
Animals a ("An animal");

System.out.println(a);
        // Вызывает версию метода toString,
        // определенную в классе Animals.
System.out.println(jumbo);
        // Вызывает версию метода toString,
        // определенную в классе Elephant.
a = jumbo;
System.out.println(a);
        // По-прежнему вызывает версию метода toString,
        // определенную в классе Elephant.
```

Для нахождения корректного метода во время переопределения используется **динамическое связывание**. Каждый объект имеет таблицу своих методов. Java в процессе выполнения программы ищет подходящую версию любых переопределенных методов. По умолчанию считается, что для методов потребуется динамическое связывание, что чревато резервированием значительных объемов памяти. Поэтому, если точно известно, что метод не будет переопределяться, следует объявить его с модификатором `final`, и это предотвратит снижение скорости выполнения программы. Хороший пример динамического связывания будет предложен вашему вниманию в практикуме далее в этой главе.

9.3. Абстракция через интерфейсы

Мы предполагали, что абстракция позволит избежать повторов при определении классов и обеспечит возможность многократного применения написанных нами классов. В Java достигнуть абстракции можно благодаря **интерфейсам** и **абстрактным классам**. Сначала мы рассмотрим интерфейсы, а в разделе 9.4 перейдем к изучению абстрактных классов.

Интерфейс — это особый вид класса, который определяет спецификацию комплекта методов. Любой класс, который реализует данный интерфейс, может использовать эти методы. Мы можем рассматривать интерфейс как определение комплекта стандартов. Класс, который объявляется как применяющий данный интерфейс, как бы получает "печать", подтверждающую его соответствие этим стандартам. Следовательно, объекты класса могут получить доступ к любым методам, которым нужен объект с такими стандартами.

Объявление интерфейса

```
interface Interfaceid {
  спецификации метода
}
```

Реализация интерфейса

```
class Classid implements Interfaceid {
  Переменные и тела определенных методов,
  принадлежащие данному интерфейсу.
}
```

На схемах интерфейс обозначается так же, как класс, но с дополнительным символом I в левом верхнем углу; кроме того, имя его выделяется курсивом. Наряду с методами интерфейс может иметь константы, которые перечисляются в том же разделе класса, что и переменные.

Главная программа может объявить переменные или массивы, где их типом будет класс-интерфейс. Любой класс, который реализует интерфейс, является приемлемым для создания экземпляров объектов. Массив может даже содержать объекты, относящиеся к различным классам, которые реализуют один и тот же интерфейс. Далее мы проанализируем пример, демонстрирующий применение данной технологии. В следующей главе также описан целый ряд случаев использования интерфейсов.

Пример 9.2. Сортировка

Задача. Метод сортировки, разработка которого рассматривалась в главе 6, чрезвычайно полезен. Поэтому целесообразно модифицировать данный метод таким образом, чтобы его работа не зависела от вида сортируемых данных.

Решение. Мы рассмотрим, что делает класс объектов *доступным для сортировки*, и инкапсулируем эту операцию (операции) в интерфейс, который называется `Sortable`. Затем изменим метод `selectionSort`, чтобы он работал с объектами, имеющими тип `Sortable`. После этого любой класс, объекты которого следует отсортировать, должен просто реализовать интерфейс `Sortable`, после чего он сможет вызывать методы класса `Sort`.

На рис. 9.3 схематически показаны отношения между различными классами и интерфейсом (мы исходили из того, что сортировке подлежат данные о странах и студентах). К чему обращается метод x? В чем различия между процессами сортировки, выполняемыми для разных классов? Ответ таков: выполняется не присваивание, а сравнение объектов, поскольку сравнение предполагает обмен элементами — так работают все методы сортировки. Для каждого класса сравнение будет выполняться по-своему. Есть два положения в сравнении неравных элементов — меньше чем и больше чем. В следующей программе мы будем использовать *меньше чем*.

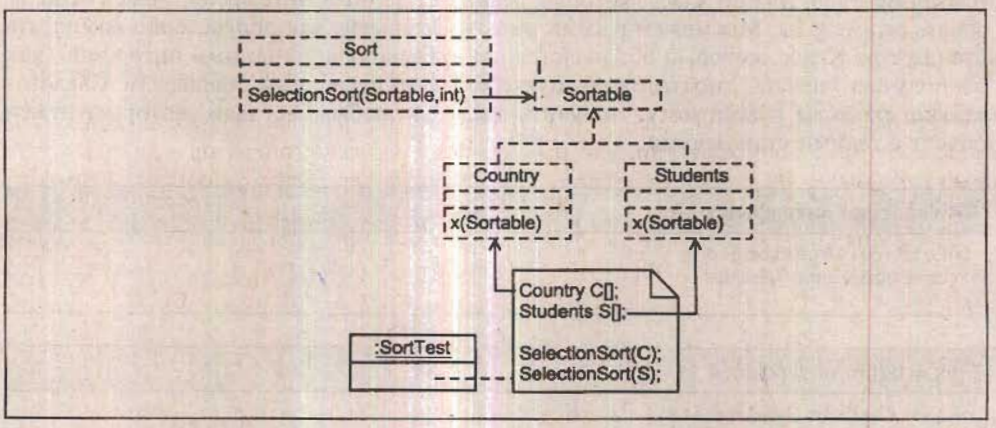


Рис. 9.3. Классы при абстрактной сортировке

Алгоритм. Процедура сортировки применяется довольно широко, поэтому мы поместим ее в пакет `myutilities`. Сначала представим интерфейс `Sortable` следующим образом:

```
package myutilities;

public interface Sortable {
  public boolean lessThan (Sortable a);
}
```

Класс `Sort`, описанный в разделе 6.4, должен быть слегка изменен и представлен в следующем виде:

```
package myutilities;

public class Sort {
   /* Класс Sort.   Автор - Дж. М. Бишоп, февраль 1997 г.
    *               Исправлен в октябре 1997 г.
    * Предоставляет один метод сортировки
    * для массивов объектов любой длины,
    * где класс объектов реализует интерфейс Sortable.
    */

  public static void selectionSort(Sortable a [], int n) {
    Sortable temp;
    int chosen;
```

```
    for (int leftmost = 0; leftmost < n-1; leftmost++) {
      chosen = leftmost;
      for (int j = leftmost+1; j < n; j++) {
        if (a[j].lessThan(a[chosen])) {
          chosen = j;
        }
      }
      temp = a[chosen];
      a[chosen] = a[leftmost];
      a[leftmost] = temp;
    }
  }
}
```

Теперь и параметр массива, и промежуточная переменная `temp` объявлены как `Sortable`. Также в сравнении, где раньше использовался оператор <, теперь для объекта вызывается метод `lessThan`, а в качестве параметра указывается другой объект.

Далее мы внедрим в класс объектов, подлежащих сортировке, реализацию метода `lessThan`. Например, простой класс `Country` будет таким:

```
import myutilities.*;

class Country implements Sortable {
   /* Класс Country.    Автор - Дж. М. Бишоп, декабрь 1996 г.
    * - - - - - - - - - - Java 1.1
    * Содержит название страны и пригоден для сортировки.
    */
  private String name;

  Country(String s) {
    name = s;
  }

  public boolean lessThan(Sortable a) {
    Country b = (Country) a;
    return (name.compareTo(b.name) < 0);
  }

  public String toString() {
    return name;
  }
}
```

В реализации метода `lessThan` сначала необходимо выполнить операцию приведения типа по отношению к объекту `Sortable`, переданному в качестве параметра, к классу `Country`. Затем метод `compareTo` (часть класса `String`) приступает к работе с переменной `name` текущего объекта и переменной `name` параметра. Метод `compareTo` возвращает отрицательное значение, если результатом сравнения является "меньше" (если первый операнд меньше второго), положительное значение — если "больше", 0 — если "равно". Обратите внимание на то, что метод `lessThan`, как и метод `toString`, должен быть объявлен как открытый (`public`), поскольку он переопределяет метод в классе другого пакета.

Программа. Теперь классы `Country`, `Sortable` и `Sort` можно использовать в тестовой программе.

```
import java.io.*;
import javagently.*;
import myutilities.*;

class SortTest {
   /* Программа SortTest.    Автор - Дж. М. Бишоп, декабрь 1996 г.
    * - - - - - - - - - -     Java 1.1 - октябрь 1997 г.
    *                         Исправлена в августе 2000 г.
    * Предназначена для сортировки названий стран или целых чисел.
    * Демонстрирует интерфейсы и ссылается на
    * независимый сортировщик.
    */

  public static void main(String [] args) throws IOException {
    new SortTest ();
  }

  SortTest () throws IOException {
    Country t [] = new Country[10];
    String s;

    Stream in = new Stream ("Countries.data", Stream.READ);
    for (int i = 0; i < t.length; i++) {
      s = in.readString();
      t[i] = new Country(s);
    }

    System.out.println("Первоначальная таблица");
    System.out.println("= = = = = = = = = = =\n");
    for (int i = 0; i < t.length; i++) {
      System.out.println(t[i]);
    }

    Sort.selectionSort(t, t.length);

    System.out.println();
    System.out.println("Отсортированная таблица");
    System.out.println("= = = = = = = = = = =\n");
    for (int i = 0; i < t.length; i++) {
      System.out.println(t[i]);
    }
  }
}
```

Заметьте, что в программе фактически не упоминается интерфейс `Sortable`. Другими словами, эта программа может работать со старой версией метода сортировки. Изменения коснулись только отношений между объектом и методом сортировки. В результате мы достигли гибкости, позволяющей производить сортировку для других классов.

Тестирование. Чтобы убедиться в том, что сортировка выполняется правильно, мы выведем обрабатываемые данные и результат работы программы.

```
Japan
Russia
Argentina
Finland
Ukraine
Spain
Algeria
France
Iceland
Israel

Первоначальная таблица
= = = = = = = = = = = = =

Japan
Russia
Argentina
Finland
Ukraine
Spain
Algeria
France
Iceland
Israel

Отсортированная таблица
= = = = = = = = = = = = =

Algeria
Argentina
Finland
France
Iceland
Israel
Japan
Russia
Spain
Ukraine
```

Расширение. Важно проверить, выполняются ли в программе требования, позволяющие говорить о ее независимости. Наиболее быстрый способ добиться желаемого состоит в изменении класса `SortTest` для сортировки иных данных, например целых чисел. Поскольку интерфейс `Sortable` имеет дело с объектами, мы должны поместить целое число в класс, который, в свою очередь, должен находиться во внутреннем классе.

```
import java.io.*;
import javagently.*;
import myutilities.*;

class SortTestInt {
   /* Программа SortTestInt. Автор - Дж. М. Бишоп, декабрь 1996 г.
    * - - - - - - - - - - - -  Java 1.1 - октябрь 1997 г.
    *                          Исправлена в августе 2000 г.
    * Предназначена для сортировки целых чисел в таблице.
    * Демонстрирует использование ссылки на независимый
    * сортировщик, а также помещение целых чисел в
    * классы-оболочки.
    */
```

```
  public static void main(String [] args) throws IOException {
    new SortTestInt ();
  }

  SortTestInt () throws IOException {
    OurInteger[] t = new OurInteger[10];
    Stream in = new Stream(System.in);

    for (int i = 0; i < 10; i++) {
      System.out.println("Введите число");
      t[i] = new OurInteger(in.readInt());
    }

    System.out.println("Первоначальная последовательность чисел");
    System.out.println("= = = = = = = = = = = = = = = = = = = = ");
    for (int i = 0; i < t.length; i++) {
      System.out.print(t[i].val+" ");
    }
    System.out.println();

    Sort.selectionSort(t, t.length);

    System.out.println();
    System.out.println("Отсортированная последовательность чисел");
    System.out.println("= = = = = = = = = = = = = = = = = = = = ");
    for (int i = 0; i < t.length; i++) {
      System.out.print(t[i] +" ");
    }
    System.out.println();
  }

class OurInteger implements Sortable {

  /* Этот класс-член инкапсулирует целое число
   * как объект и применяет интерфейс Sortable
   * для сортировки целых чисел.
   */

  int val;

  public OurInteger(int i) {
    val = i;
  }

  public boolean lessThan(Sortable a) {
    OurInteger i = (OurInteger) a;
    return val < i.val;
  }

  public String toString () {
    return String.valueOf(val);
  }
 }

}
```

Прогон программы (с использованием данных, введенных с клавиатуры) даст следующее:

```
Первоначальная последовательность чисел
= = = = = = = = = = = = = = = = = = =
5 8 1 2 0 4 3 7 9 6

Отсортированная последовательность чисел
= = = = = = = = = = = = = = = = = = =
0 1 2 3 4 5 6 7 8 9
```

В заключение отметим, что интерфейсы могут содержать только спецификации методов, но не завершенные методы или переменные. При необходимости они могут включать также константы.

9.4. Абстрактные методы и классы

Последний компонент "трио", обеспечивающего эффективную работу с классами, — абстрактные методы и абстрактные классы. В сочетании с интерфейсами и наследованием они позволяют четко компоновать программы большого объема.

Абстрактные методы

Абстрактные методы выполняют функцию "заглушек" для методов, которые могут быть указаны на одном уровне, а выполняться будут разнообразными способами на нижних уровнях. Возьмем, к примеру, метод из иерархии `Nature`, посвященный сну. Вероятно, нет смысла иметь общий метод, описывающий модели сна всех животных. Но при работе с объектами на уровне класса `Animals` полезной окажется возможность вызвать различные методы для разных классов, расположенных ниже. Объявим в классе `Animals` метод `sleep`:

```
abstract void sleep ();
```

В результате класс `Animals` станет абстрактным.

Абстрактные классы

Абстрактным является любой класс, в котором есть хотя бы один абстрактный метод. Абстрактный класс не может быть использован для объявления объектов: наличие абстрактного метода означает, что класс не завершен. Сначала должны быть созданы подклассы и заполнены абстрактные методы. Так, приведенный выше пример с классом `Animals` должен иметь следующий вид:

```
abstract class Animals extends Nature {
  // в этом месте следуют необходимые объявления

  abstract void sleep ();

  // еще объявления
}
```

Теперь каждый из классов нижнего уровня, таких как `Elephant` и `Rhino`, сможет завершить определение метода `sleep`. Если в классе `Herbivores` не будет ничего указано относительно метода `sleep`, то этот класс следует объявить абстрактным.

Как теперь организовать работу с иерархией? Правильно, все объявления отныне будут выполняться на определенном нижнем уровне, например:

```
Elephant E [] = new Elephant [10];
Rhino R [] = new Rhino [5];
```

Но теперь мы не можем объявить ни класс `Herbivores`, ни класс `Animals`, поскольку они являются абстрактными. Вот чего мы достигли за счет абстракции: определили, что должно быть сделано, и переложили на другие классы ответственность за выполнение намеченного.

Пакеты Java содержат несколько примеров абстрактных классов. Некоторые их них уже рассматривались в предыдущих главах (`Date`, `TimeZone`, `NumberFormat` и т.д.). Эти классы предоставляют методы класса, которые могут быть использованы для создания экземпляров класса, что и выполнялось в предыдущих программах.

9.5. Практикум 4. Ветеринарные бирки

Исходные условия

Несколько лет назад в Ассоциации ветеринаров Саванны была разработана система учета домашних животных: каждому животному выдавалась бирка, по которой можно было быстро установить хозяина, если животное потеряется. На бирке указывались кличка животного и номер телефона владельца. Впоследствии выяснилось, что после изменения контактных данных хозяева не всегда обновляют бирки животного, и обнаружить местонахождение владельцев потерянных животных становилось все труднее.

Было принято решение централизовать регистрацию всех бирок, создать один отдел, где каждый мог бы узнать, зарегистрировано ли животное как потерявшееся, и внедрить усовершенствованные бирки `Xtag`, содержащие еще и телефон отдела регистрации. Очевидно, что старая система бирок также должна поддерживаться. На внедрение новой системы регистрации (`register`) отведено несколько месяцев, после чего должна быть обеспечена обработка информации, нанесенной как на старые, так и на новые бирки.

Решение 1 и конструкция класса

Для решения поставленной задачи мы воспользуемся функцией наследования. У нас есть программа для работы со старыми бирками. Теперь необходимо определить новый класс бирок — `XTags`, который будет расширять старый класс бирок (`extends Tags`) и содержать новый элемент данных. Оба класса могут быть сохранены в массиве `register` как объекты класса `Tags` — суперкласса данной иерархии.

В программе для создания объектов используются классы `Tags` и `XTags`. Метод `main` служит для вызова трех методов, которые выполняют необходимую обработку, а метод `makeTags` предназначен для создания объектов `Tags` и размещения их в массиве. Массив объявляется и определяется следующим образом:

```
static Tags register [] = new Tags [100];
```

Метод `showTags` выводит на экран информацию, содержащуюся в массиве `register`, а метод `checkTags` является средством нахождения номера телефона владельца по кличке животного (`name`).

Изначально класс Tags выглядит следующим образом:

```
class Tags {
   /* Класс Tags.      Автор - Дж. М. Бишоп, январь 1997 г.
    * Предназначен для хранения данных о домашнем животном.
    */

  String name, phone;

  public Tags (String n, String p) {
    name = n;
    phone = p;
  }

  public String toString() {
    return name+" тел: "+phone;
  }

}
```

Класс содержит конструктор и один стандартный метод toString, который позволяет выполнить конкатенацию строковых значений соответствующих переменных с другими строками и используется в операторе println, а также других операторах, где требуются строки. Вот как выглядит часть метода makeTags, которая предназначена для добавления бирки (tag) в массив register:

```
Tags tag = new Tags (petsName, ownersPhone);
register[index] = tag;
```

В методе showTags строки, предварительно сформированные методом toString, выводятся следующим оператором:

```
display.println(i+"\t"+ register[i]);
```

Данные для программы (клички животных и телефонные номера) хранятся в файле. Полную исходную версию программы вы найдете на Web-узле, а мы перейдем к разработке расширенной версии для демонстрации наследования.

Решение 2

Теперь надлежит добавить в программу класс XTags, который выглядит следующим образом:

```
public class XTags extends Tags {

   /* Класс Xtags.     Автор - Дж. М. Бишоп, январь 1997 г.
    * Предназначен для создания новых ветеринарных бирок, содержащих
    * номер телефона отдела регистрации. Применяется
    * наследование.
    */
  String vet;

  public XTags(String n, String p, String v) {
    super(n, p);
    vet = v;
  }
```

```
  public String toString() {
    return name+" тел: "+phone+" тел. отдела: "+vet;
  }

}
```

Ряд моментов заслуживает особого внимания. Класс `XTags` имеет доступ к переменным `name` и `phone`, определенным в классе `Tags`. Он переопределяет метод `toString` класса `Tags` и объявляет собственную новую переменную `vet`.

Программа. Прежде чем мы приступим к рассмотрению полного текста программы по ветеринарии, давайте проанализируем важный фрагмент кода, относящийся к наследованию.

```
1.  char kind = Text.readChar(fin);
2.  petsName = Text.readString(fin);
3.  ownersPhone = Text.readString(fin);
4.  Tags tag;
5.  switch (kind) {
6.    case 'p':
7.    case 'P':
8.    tag = new Tags(petsName, ownersPhone);
9.    break;
10.  default:
11.    case 'x':
12.    case 'X':
13.      String vetsPhone = Text.readString(fin);
14.      tag = new Xtags(petsName, ownersPhone, vetsPhone);
15.  }
16.  register[index] = tag;
```

Первая строка кода получает информацию о том, какая бирка должна быть выдана для животного (т.е. какой объект должен быть создан) — `Tags` или `XTags`, что указывается с помощью специальных символов: `'P'` — простая бирка `Tags`, `'X'` — усовершенствованная бирка `XTags`. Во второй и третьей строках считывается такая информация, как кличка животного и номер телефона хозяина. Далее для создания двух типов бирок используется оператор `switch`. Поскольку мы хотим ускорить внедрение новых бирок, то по умолчанию задаем создание объекта `XTags`. Используя несколько вариантов `case`, мы обеспечим возможность ввода специальных символов `'P'` и `'X'` в любом регистре.

Оператор для ввода символа `'P'` (`case 'P':`) взят из старой версии программы и создает новый объект `Tags`. Если же бирка новая, мы вводим большее количество информации и создаем объект `XTags`. *Однако в каждом из случаев значения присваиваются одним и тем же переменным объекта.* Поскольку класс `XTags` происходит от класса `Tags`, т.е. является представителем того же семейства, он будет приемлем везде, где применяется класс `Tags`. Таким образом, наследование — это эффективное средство, позволяющее обеспечить многократное использование программы и расширение ее функциональных возможностей.

Ниже приведен полный текст обновленной программы. Главная программа создает экземпляр вложенного класса `DataHandler`, в котором и выполняется большая часть операций.

```
import java.io.*;
import javagently.*;

class VeterinaryTags {

   /* Программа "Ветеринарные бирки".  Автор - Дж. М. Бишоп, январь 1997 г.
    *                                  Java 1.1 - октябрь 1997 г.
    *                                  Класс Display -  июль 1999 г.
    * Содержит информацию о домашних животных
    * и осуществляет поиск нужных данных по имени найденного
    * животного. Использует два типа бирок.
    * Демонстрирует наследование.
    */

  public static void main(String args []) throws IOException {
    TagDataHandler vetAssoc = new TagDataHandler ();
    vetAssoc.initialize();
    vetAssoc.makeTags();
    vetAssoc.showTags();
    while (true)
      // Завершает работу после нажатия кнопки "Закрыть"
      // в рабочем окне программы.
      vetAssoc.checkTags();
  }

  static class TagDataHandler {

    int index;
    Tags[] register = new Tags[100];
    Display display = new Display ("Система ветеринарных бирок");
    BufferedReader fin = Text.open(System.in);

    void initialize () {
      // По умолчанию при отсутствии готового файла с
      // данными для потока fin используется клавиатура.

      display.println("Система бирок, принятая в Саванне");
      display.prompt ("Имя файла с данными", "tags.data");
      while (true) {
        display.ready("Введите имя файла и нажмите кнопку \"Готово\"");
        String filename = display.getString("Имя файла с данными");
        try {
          fin = Text.open(filename);
          break;
        } catch (FileNotFoundException e) {
          display.println("Файл не обнаружен, попытайтесь еще раз");
        }
      }
    }
    void makeTags() throws IOException {
      String petsName, ownersPhone;
      while (true) {
        try {
          char kind = Text.readChar(fin);
```

```
            petsName = Text.readString(fin);
            ownersPhone = Text.readString(fin);
            Tags tag;
            switch (kind) {
              case 'p':
              case 'P':
                tag = new Tags(petsName, ownersPhone);
                break;
            default:
              case 'x':
              case 'X':
                String vetsPhone = Text.readString(fin);
                tag = new XTags(petsName, ownersPhone, vetsPhone);
            }
            register[index] = tag;
            index++;
          }
          catch (EOFException e) {
            break;
          }
        }
        display.println("Введены все данные о "+index+"-и животных\n");
      }

      void showTags() {
        display.println("Отдел регистрации животных\n");
        display.println("Номер животного, кличка и номер телефона.");
        for (int i = 0; i < index; i++)
          display.println(i+"  "+register[i]);
        display.prompt("Найдено животное по кличке","Бастер");
      }

      void checkTags() {
        display.ready("\nВведите имя животного, указанное на бирке,
                      \n и нажмите кнопку \"Готово\"");
        String info = display.getString("Найдено животное по кличке");
        boolean found = false;
        for (int i = 0; i < index; i++)
          if (register[i].name.equals(info)) {
            display.println("Животное по кличке "+
              info+" \nзарегистрировано под номером "+i);
            display.println("Полная информация:\n"+register[i]);
            found = true;
            break;
          }
        if (!found)
          display.println("Животное по кличке "+
            info+" не зарегистрировано.");
      }
    }
}
```

Тестирование. На рис. 9.4 показан результат работы программы.

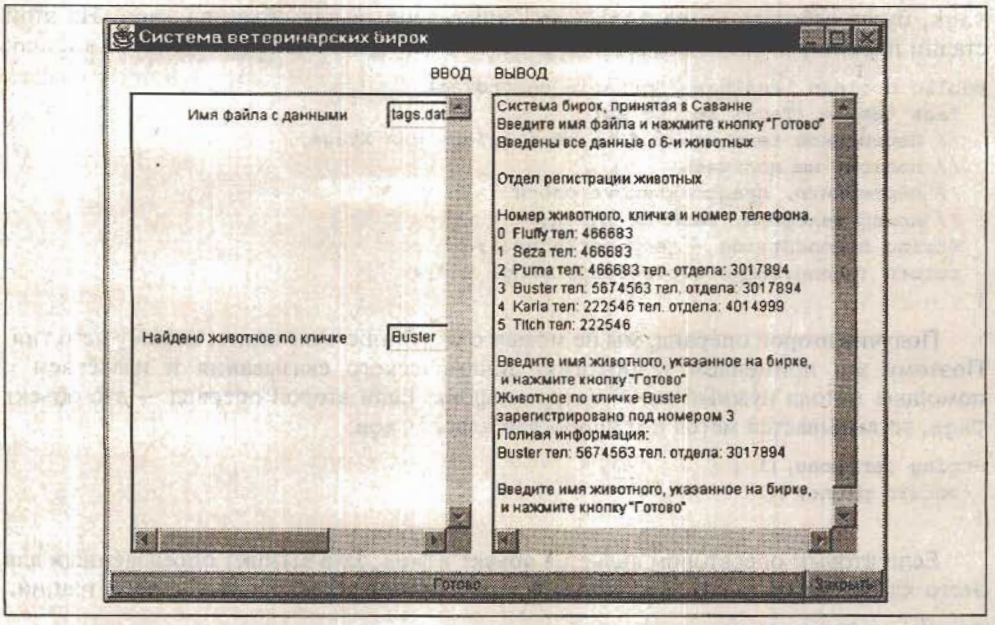


Рис. 9.4. Результат выполнения программы "Ветеринарные бирки"

Сортировка данных в массиве register

В этом разделе мы поговорим о том, как сделать массив бирок `register` доступным для сортировки. Это означает, что нужно сделать сортируемыми объекты класса `Tags`. В первую очередь предстоит решить, на основе какой переменной проводить сортировку. Мы можем отсортировать клички животных. В этом случае метод `lessThan` будет выглядеть следующим образом:

```
public boolean lessThan (Sortable secondTag) {
  Tags temp = (Tags) secondTag;
  return (name.compareTo(temp.name) < 0);
}
```

Поскольку мы должны привести объект, переданный методу как параметр (который, к тому же, принадлежит к общему типу — интерфейсу `Sortable`), к конкретному типу `Tags`, прежде чем сравнивать переменные, принадлежащие объектам `Tags`, нам нужен локальный объект `temp`. Только после операции приведения типа любой объект `XTags` для использования метода `lessThan` сможет обращаться к высшим классам в иерархии наследования.

Однако предположим, что в Ассоциации захотят выполнить сортировку по телефонным номерам. При использовании старой системы бирок в результате будут собраны вместе животные одного хозяина. Но для бирок `XTags` надлежит выполнять сортировку по телефону отдела регистрации. Кроме того, как определить метод `lessThan`, если одним операндом является объект `Tags`, а другим — `XTags`?

Мы начнем рассмотрение процесса сортировки со следующего: метод `selectionSort` вызывает метод `lessThan`, определенный в классе первого операнда. Поэтому, если первым операндом, который должен сравниваться, является объект

Tags, будет работать метод lessThan, определенный для объектов Tags. На этой стадии параметр может быть любого типа. Вот как будет выглядеть метод lessThan:

```
public boolean lessThan (Sortable secondTag) {
  Tags temp = (Tags) secondTag;
  // Переменной temp может быть объект Tags или Xtags,
  // поэтому мы получаем
  // переменную, представляющую собой
  // номер телефона, вызывая метод:
  String secondString = temp.getPhone();
  return (phone.compareTo(secondString) < 0);
}
```

Получив второй операнд, мы не можем с уверенностью сказать, какой у него тип. Поэтому мы применяем технологию динамического связывания и извлекаем с помощью метода нужный нам номер телефона. Если второй операнд — это объект Tags, то вызывается метод getPhone для класса Tags:

```
String getPhone () {
  return phone;
}
```

Если вторым оператором является объект XTags, Java вызовет определенный для этого класса метод getPhone, который вернет номер телефона отдела регистрации.

```
String getPhone () {
  return vet;
}
```

Метод lessThan в классе XTags аналогичен описанному для класса Tags за исключением того, что последняя строка относится к переменной vet. Вот эта строка:

```
return (vet.compareTo(secondString) < 0);
```

Как уже говорилось, второй операнд предназначен для считывания номера телефона хозяина животного или отдела регистрации, что зависит от типа сравниваемого объекта. В завершение ознакомимся с полным кодом класса* Tags:

```
import myutilities.*;

class Tags implements Sortable{

   /* Класс Tags.     Автор - Дж. М. Бишоп, январь 1997 г.
    * - - - - - -      Обновлен в июле 1999 г.
    * Предназначен для хранения данных о домашнем животном.
    */

  String name, phone;

  public Tags (String n, String p) {
    name = n;
    phone = p;
  }

  String getPhone () {
    return phone;
  }
```

* Нужно также добавить в начало кода класса XTags строку import myutilities.*; — *Прим. ред.*

```
  public boolean lessThan (Sortable secondTag) {
    Tags temp = (Tags) secondTag;
    // Переменной temp может быть объект Tags или XTags,
    // поэтому мы получаем соответствующую переменную,
    // представляющую собой номер телефона, вызывая метод
    String secondString = temp.getPhone();
    return (phone.compareTo(secondString) < 0);
  }

  public String toString() {
    return "Владелец: "+phone+" животное: "+name;
  }

}
```

Главную программу можно немного настроить. Программа не реализует интерфейс `Sortable`; все это выполняется класами `Tags` и `XTags`. В главной программе нужно лишь вызвать класс `Sort` и его метод *:

```
Sort.selectionSort (register,index);
```

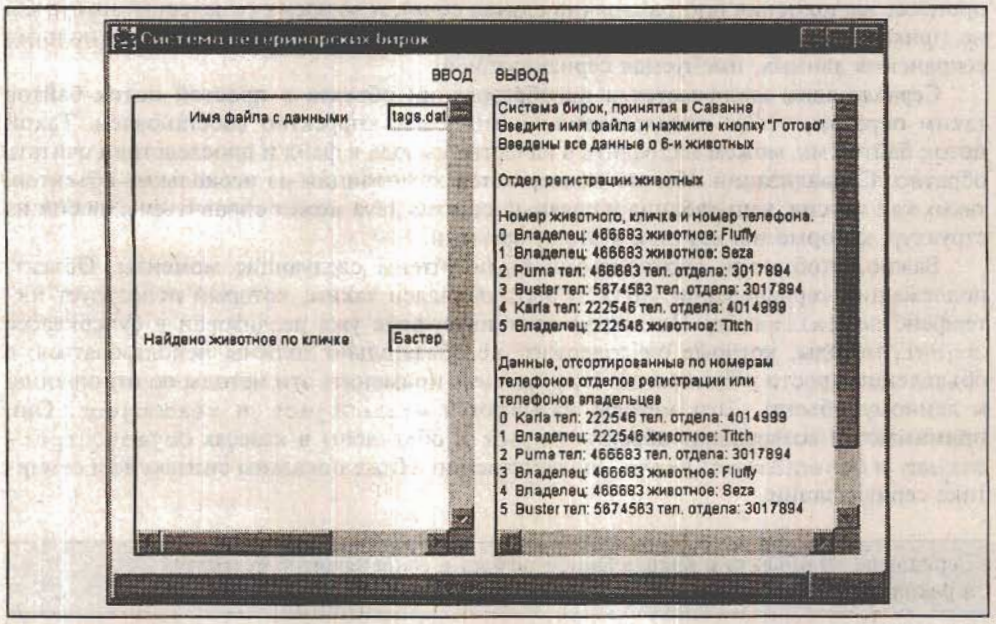


Рис. 9.5. Данные, отсортированные по телефонным номерам отделов или владельцев животных

---

* Для этого вам нужно в начало программы поместить оператор `import myutilities.*;`, а перед закрывающей скобкой метода `showTags` вставить следующие строки кода:

```
display.println("     ");
display.println("Данные, отсортированные по номерам: \nтелефонов отделов регистрации, или ");
display.println("телефонов владельцев ");
Sort.selectionSort(register,index);
for (int i = 0; i < index; i++)
    display.println(i+"  "+register[i]); — Прим. ред.
```

На рис. 9.5 представлен результат работы программы после сортировки данных по телефонным номерам. Чтобы показать, как происходит сортировка, мы изменили вид каждой выводимой строки. В окне программы мы видим, что номера телефонов на бирках животных, которые принадлежат одному хозяину и имеют клички Флаффи, Сез и Пум, сортируются отдельно, потому что Пум имеет бирку `XTags`. Данный объект сортируется по телефонному номеру отдела регистрации. Поскольку переменная `vetsPhone` рассматриваемого объекта имеет наименьшее значение, данные с этой бирки выводятся первыми.

9.6. Сериализация

Внимательный читатель очевидно заметил, что и в программе "Кофейный магазин Нельсона", и в программе "Ветеринарные бирки" мы прибегали к сохранению данных в памяти. Данные вводились из текстового файла, после чего их можно было обновлять (как в примере о продажах кофе) или обрабатывать во время выполнения запроса (как при регистрации бирок). Однако не было средств, позволяющих сохранять данные между запусками программ. Поэтому обе программы до некоторой степени нереальны. Очевидно, что надлежит сохранять данные и в процессе выполнения программы (на случай сбоя), и во время ее завершения (чтобы не приходилось начинать с "нуля"). В Java применяется специальная технология сохранения данных, именуемая сериализацией.

Сериализация заключается в преобразовании объекта в простой поток байтов таким образом, чтобы впоследствии он мог быть корректно восстановлен. Такой поток байтов мы можем направить в качестве вывода в файл и впоследствии считать обратно. Сериализации обычно подвергается композиция из нескольких объектов, таких как массив, хеш-таблица и связный список. Java может справиться с любой из структур, которые мы изучили к этому времени.

Важно, чтобы при сериализации были учтены следующие моменты. Объект, подлежащий сериализации, должен быть объявлен таким, который использует интерфейс `Serializable`. Поскольку этот интерфейс уже реализован в суперклассе `Object`, методы, которые он содержит, не обязательно должны использоваться: в объявлении просто указывается, что *разрешено* применять эти методы по отношению к данному объекту. Эти методы называются `writeObject` и `readObject`. Они принимают и возвращают объекты `Object` и объявлены в классах `ObjectOutputStream` и `ObjectInputStream` соответственно. Ниже показаны синтаксис и семантика сериализации:

Сериализация объекта с направлением потока в сохраняющую структуру и реконструкция объекта

```
// Сохраняет вывод.
ObjectOutputStream outstream =
  new ObjectOutputStream  (outputStreamObject);
 outstream.writeObject(theObject);
outstream.close();

// Вводит сохраненные данные.
ObjectInputStream instream =
  new ObjectInputStream (inputStreamObject);
theObject = (Serializable) instream.readObject();
instream.close();
```

Конструктору объекта `outstream` передается в качестве параметра поток, соответствующий файлу, массиву или сетевому соединению (см. главу 14). Соответствующий конструктор для объекта ввода `instream` должен предоставить в качестве параметра тот же фактический поток.
Метод `writeObject` принимает направляемый ему параметр, преобразует его в поток байтов и передает объекту `outstream`.
Метод `readObject` извлекает байты из соответствующего объекта `instream` и восстанавливает структуру первоначального объекта (массива, списка и т.д.).
До того как выполнение программы будет продолжено, потоки `outstream` и `instream` должны быть закрыты.
Класс `theObject` должен объявляться как реализующий интерфейс `Serializable`.

Существует несколько исключительных ситуаций, которые могут возникнуть при выполнении описанного процесса. Они связаны с ошибками ввода-вывода (некорректное имя файла и т.д.) или ошибкам класса (для объекта не указано, что он использует интерфейс `Serializable`). Следовательно, метод, в который включены приведенные выше конструкции, должен перехватывать исключительные ситуации `IOException` и `ClassNotFoundException`.

В следующем примере, а также в некоторых задачах в конце главы производится сериализация данных в файлы. В главе 14 рассказано, как выполнять сериализацию через сеть.

Пример 9.3. Применение интерфейса Serializable

Задача. Цель программы — продемонстрировать принципы применения интерфейса `Serializable`.

Решение. Необходимо разработать программу для сериализации любого объекта и проверить, как она работает с массивом объектов и со списком объектов (скажем, со списком сортов кофе, составленном в примере 8.2). При разработке метода следует указывать, что объекты применяют интерфейс `Serializable`. Метод должен выглядеть так:

```
Serializable outAndInAgain (Serializable structure) {
  Serializable copy = null;
    try {
    // Сериализация объекта в файл mydata.
      ObjectOutputStream out =
        new ObjectOutputStream (new FileOutputStream("mydata"));
      out.writeObject(structure);
      out.close();

    // Ввод объекта обратно в программу.
      ObjectInputStream in =
        new ObjectInputStream (new FileInputStream("mydata"));
      copy = (Serializable) in.readObject();
      in.close();
    } catch (IOException e)
      {System.out.println("Проблемы ввода-вывода "+e.getMessage());}
      catch (ClassNotFoundException e)
      {System.out.println("Проблемы с классом "+e.getMessage());}
  return copy;
}
```

Этот метод предназначен только для тестирования, поскольку он записывает поток байтов в файл и немедленно считывает его оттуда, что нельзя считать нормальной последовательностью событий. Обратите внимание на то, что объекты, которые передаются конструкторам в качестве параметров для объектов `ObjectOutputStream` и `ObjectInputStream`, являются файлами, базирующимися на классах, которые мы не использовали. В разделе 7.3, рассматривая класс `Stream`, мы использовали классы `FileReader` и `FileWriter`. Разница заключается в том, что классы, применявшиеся ранее в методе, имеют дело с необработанными потоками байтов, а последние два, используемые `Stream`, работают на уровне символов.

Программа. Программа состоит из двух частей. В первой части выполняется сериализация небольшого массива, элементы которого являются объектами класса `Students` и имеют переменные, содержащие имена студентов и информацию об их денежных сбережениях (`marks`). Во второй части используется проверочный класс `Coffee` из примера 8.2 и демонстрируется, как может быть выполнена сериализация списка с помощью одного оператора.

```
import java.io.*;
import javagently.*;

class TestFiles implements Serializable {

   /* Сериализация в файл.    Дж. М. Бишоп, апрель 2000 г.
    * = = = = = = = = = = = =
    *
    * Демонстрирует, как сохранить и извлечь
    * структуру данных из файла.
    * Программа и класс объектов должны быть объявлены как
    * реализующие интерфейс Serializable.
    */

  TestFiles () throws IOException {
    doItWithAnArray();
    doItWithAList();
  }

  Serializable outAndInAgain (Serializable structure) {
    Serializable copy = null;

    try {
    // Сериализация объекта.
      ObjectOutputStream out =
        new ObjectOutputStream (new FileOutputStream("mydata"));
      out.writeObject(structure);
      out.close();

    // Вводит объект обратно.
      ObjectInputStream in =
        new ObjectInputStream (new FileInputStream("mydata"));
      copy = (Serializable) in.readObject();
      in.close();
    } catch (IOException e)
      {System.out.println("Проблемы ввода-вывода "+e.getMessage());}
      catch (ClassNotFoundException e)
```

```
        {System.out.println("Проблемы с классом "+e.getMessage());}
        return copy;
    }

    void doItWithAnArray (){
      Students [] table = new Students [10];
      Students [] copy = new Students [10];
      // Ввод данных для проверки.
      table[0] = new Students("Peter",84);
      table[1] = new Students("John",51);
      table[2] = new Students("Kathy",65);
      int n = 2;

      // Выводит на печать первоначальные данные.
      System.out.println("\nПервоначальная таблица");
        for (int i = 0; i <=n; i++)
          System.out.println(table[i]);

      // Делаем небольшие изменения.
      table[1].marks = 58;
      table[2].name = "Cathy";
      table[3] = new Students("Sipho",70);
      n = 3;

      copy = (Students []) outAndInAgain (table);

      // Выводит на печать скопированную таблицу.
      System.out.println("\nСкопированная таблица");
      for (int i = 0; i <=n; i++) {
          System.out.println(copy[i]);
      }
    }

    void doItWithAList () throws IOException {
      Coffee coffees = null;
      Coffee copy = null;
      String name;
      Stream in = new Stream ("names.dat", Stream.READ);

      // Считывает данные по сортам кофе из текстового файла.
      for (int i = 0; i <=4; i++) {
        name = in.readString();
        coffees = new Coffee (name, coffees);
      }

      // Выводит список.
      Coffee list = coffees;
      System.out.println("\nСорта кофе в списке");
      for (int i = 0; i <=4; i++) {
        System.out.println(list.name);
        list = list.link;
      }

      copy = (Coffee) outAndInAgain (coffees);
```

```
    // Выводит список, возвращенный в программу.
    list = copy;
    System.out.println("\nСписок, возвращенный в программу ");
    for (int i = 0; i <=4; i++) {
      System.out.println(list.name);
      list = list.link;
    }
  }
  public static void main(String args []) throws IOException {
    new TestFiles ();
  }
class Students implements Serializable {

  String name;
  int marks;

  Students (String n, int m) {
    name = n;
    marks = m;
  }

  public String toString () {
    return name + " " + marks;
  }
}

class Coffee implements Serializable {
  /* Небольшой проверочный класс со ссылкой. */

  Coffee (String name, Coffee c) {
    this.name = name;
    link = c;
  }

  String name;
  Coffee link;
}

}
```

Тестирование. Полученные результаты позволяют убедиться в том, что процесс работает.

```
Первоначальная таблица
Peter 84
John 51
Kathy 65

Скопированная таблица
Peter 84
John 58
Cathy 65
Sipho 70
```

```
Сорта кофе в списке
Итальянский
Кенийский
VIP
Колумбийский
Ява

Список, возвращенный в программу
Итальянский
Кенийский
VIP
Колумбийский
```

Резюме

Круг возможностей при объектно-ориентированном программировании значительно расширяется благодаря абстракции и наследованию. Абстракция позволяет сосредоточиться на сути метода или класса (на том, что он дает пользователю). Впоследствии может быть выполнена детализация (определение) метода. В качестве средств для достижения абстракции Java предоставляет интерфейсы, абстрактные классы и методы.

Наследование служит как минимум двум целям. Мы можем расширить сферу применения программы, создав новые версии классов, обладающих дополнительными возможностями и в то же время наследующих старые классы. Считается, что оба класса принадлежат к одному семейству, вследствие чего класс-потомок может применяться вместо класса-предка. Создав иерархию классов, каждый из которых можно применять как совместно с другими, так и отдельно, мы можем классифицировать информацию и выделить наиболее существенное для каждого уровня.

Объекты не должны существовать только в памяти: их легко передать для сохранения в постоянную среду, выполнив сериализацию.

В главе раскрывались понятия, связанные со следующими ключевыми словами:

```
extends         abstract
interface       protected
implements
```

Рассматривалось применение таких API:

```
Sortable            FileInputStream
Serializable        ObjectOutputStream
FileOutputStream    ObjectInputStream
```

Следующий перечень включает темы, на которые в главе обращено особое внимание:

- объявление интерфейса;
- реализация интерфейса;
- сериализация объекта и его восстановление.

Вопросы



9.1. В иерархию, в которой главным является класс `Nature`, необходимо добавить класс, одна из переменных которого содержит данные о сроке вынашивания потомства. В каком классе (классах) это нужно сделать?

а) `Nature`

б) `Animals`

в) `Elephant` и `Rhino`

г) `Herbivores`

9.2. Класс `TestNature` не может "видеть" переменную `name` внутри класса `Nature`, потому что:

а) переменная `name` объявлена с модификатором `protected`

б) в классе `TestNature` не создан экземпляр класса `Nature`

в) класс `Nature` по отношению к классам `Elephant` и `Rhino` является суперклассом, а не подклассом

г) переменная `name` не инициализирована

9.3. В иерархии `Nature` метод `toString` является примером:

а) переопределения

б) перегрузки

в) затенения

г) динамического связывания

9.4. Элементы списка сортов кофе из примера 8.2 можно отсортировать:

а) реализовав интерфейс `Sortable`

б) реализовав интерфейс `Sortable` и определив метод `lessThan`

в) сортировка невозможна, поскольку метод `selectionSort` работает только с массивами

г) сортировка невозможна, потому что нельзя определить метод `lessThan` для сравнения двух элементов списка

9.5. Если `a` — это объект класса `Animals`, то он имеет следующие переменные:

а) `name`

б) `name` и `grassNeeded`

в) ни одной

г) `name`, `grassNeeded`, а также или `tusks`, или `horn`

9.6. Предположим, есть классы `O` и `P`, причем `O` расширяет `P`. В этом случае правило работы в рамках иерархии будет следующим:

а) объект `O` может быть присвоен объекту `P`

б) объект `P` может быть присвоен объекту `O` с приведением типа

в) оба предыдущих высказывания верны

г) правилом не может быть ни одно из них

9.7. Следующий код призван реализовать метод `lessThan` для сравнения переменных `horn` объектов `Rhino`:

```
public boolean lessThan (Sortable a) {
  return horn < a.horn;
}
```

Он не будет скомпилирован, потому что:

а) переменная `horn` объявлена как `private`

б) объект `a` не обязательно должен иметь переменную `horn`

в) оператор < не определен для переменных, имеющих тип `double`

г) метод `lessThan` уже определен для переменных `tusks`

9.8. Для того чтобы вновь включить в программу объект, который представляет собой хеш-таблицу `hash` и был записан в файл (будем исходить из того, что для объекта-потока `oos` сделано корректное соединение), надлежит использовать следующий оператор:

а) `hash = (HashTable) oos.readObject();`

б) `hash = (Serializable) oos.readObject();`

в) `hash = oos.readObject();`

г) `hash = new readObject();`

9.9. В результате выполнения процесса, который объявлен как реализующий интерфейс `Serializable`, система выдала сообщение `ClassNotFoundException`. Причина заключается в том, что:

а) класс не скомпилирован

б) не создан экземпляр класса

в) класс не объявлен как реализующий интерфейс `Serializable`

г) класса не существует

9.10. Чтобы получить экземпляр абстрактного класса, мы:

а) определяем любые абстрактные методы, после чего объявляем класс

б) создаем экземпляр класса

в) создаем экземпляр его суперкласса

г) вызываем метод класса для создания экземпляра

Задачи

9.1. **Демонстрация свойств объекта.** Модифицируйте программу 9.1 таким образом, чтобы в ней использовались свойства объектов, рассмотренные в разделах 8.2 и 9.2. Напишите подробные комментарии к программе.

9.2. **Оценка успеваемости.** Разработайте программу для оценки успеваемости, которая содержала бы класс `Students`, состоящий из переменных, значения которых — порядковые номера студентов и полученные ими баллы. Каждому студенту выставляется пять оценок. Необходимо предусмотреть возможность вычисления суммы баллов. Программа должна выполнять следующие операции:

— запрашивать значения переменных с целью вычисления общего балла для каждого студента;

— сортировать объекты по порядковым номерам студентов;
— выполнять сортировку на основе баллов;
— выводить всю информацию.

9.3. **Расширенная программа оценки успеваемости.** Университет Саванны планирует ввести в 2001 году программу оценки успеваемости, при выполнении которой учитываются новые положения. Отметки, выставляемые студентам, теперь будут делиться на *открытые*, относящиеся к заданиям и проектам, и *контролируемые*, относящиеся к тестам и экзаменам. Старая система оценок должна остаться действительной. Решите, как расширить класс `Students`, чтобы учесть новые положения, и проанализируйте, что следует изменить в классе `Marker` и нужны ли эти коррективы. Напишите новую программу.

9.4. **Черчение графика.** Разработайте программу, при выполнении которой базовый метод вызывает для черчения графика класс `Graph`. Диапазон координат `x` и функция построения графика должны предоставляться экземплярами объектов другого класса. Следовательно, предельные координаты должны передаваться как параметры, а функция будет абстрактным методом. Создайте несколько экземпляров таких объектов для функций sin, $2x-1$ и т.д. Вводите предельные значения через интерфейс класса `Display`.

9.5. **Магазин Нельсона продает мороженое.** Нельсон решил рассмотреть возможность продажи мороженого в течение летнего сезона в своем кофейном магазине. Очевидно, что он должен отслеживать фонды и соблюдать сроки реализации продукции, поэтому программа для управления продажами мороженого во многом будет подобна той, которую мы писали для управления продажами кофе. Проанализируйте, как адаптировать программу `CoffeeShop` и ее классы к продажам мороженого. Нужно ли использовать наследование, интерфейсы, абстрактные классы, и в каком сочетании? Насколько существенными должны быть изменения, вносимые в класс `Coffee`?

9.6. **Динамическая работа кофейного магазина.** Используя сериализацию, усовершенствуйте программу `CoffeeShop` так, чтобы данные о фондах сохранялись после выполнения программы. Включите в нее средства для интерактивного добавления новых партий кофе.

9.7. **Отметки на всех курсах.** Измените программу `Marker` из примеров 9.2 и 9.3 так, чтоб оценки содержались в списке, а не в массиве. Это позволит увеличивать количество оценок с каждым новым курсом. Предусмотрите в программе выполнение операции сериализации, чтоб данные можно было записать в файл, а затем вновь ввести в программу и использовать в вычислениях.

ГЛАВА 10



Графика и пользовательские интерфейсы

10.1. Введение в awt и Swing

Язык Java предоставляет средства для поддержки технологии WYSIWYG* и GUI**. Средства, связанные с GUI, находятся в пакетах `java.awt` (пакет абстрактных классов для реализации оконного интерфейса) и `javax.swing`. Далее мы будем называть их `awt` и `Swing`. Отметим также, что данные пакеты обеспечивают возможность создания платформно-независимого пользовательского интерфейса.

Указанные пакеты существуют независимо от Java, т.е. инструменты компоновки и генерирования кода графического интерфейса не являются средствами этого языка. Такие инструменты существуют, но для работы с ними требуется мощный компьютер и больше ресурсов, чем обычно.

Полный обзор средств, содержащихся в пакетах `awt` и `Swing`, выходит за рамки настоящей книги. Мы ограничимся тем, что исследуем структуру пакетов, а затем на примерах рассмотрим наиболее распространенные функции.

Структура пакета awt

Классы пакета `awt` можно разделить на следующие группы:

- классы графики;
- компоненты (включая окна и меню);
- менеджеры размещения;
- обработчики событий;
- классы преобразования изображений.

Класс графики предоставляет возможность создавать фигуры, линии, изображения, а также выбирать шрифты, цвета и т.д. **Компонентами** называются такие элементы, как кнопки, текстовые поля, разнообразные меню и полосы прокрутки. Мы можем поместить их в **контейнеры**, а затем выбрать один из **менеджеров размещения** для позиционирования на экране надлежащим образом. Подпакет `java.awt.event` предназначен для обработки внешних событий (нажатий кнопок, перемещений мыши и т.д.) с помощью **обработчиков событий, слушателей** и **адаптеров**. Подпакет `java.awt.image` используется пакетом `awt` для представления изображений в различных форматах. Общая схема главного пакета `awt` воспроизведена на рис. 10.1 .

---

* WYSIWYG — это аббревиатура от "what you see is what you get", что в переводе означает "что вы видите, то и получите" (другими словами, "что на экране — то и на принтере").

** GUI (Graphical User Interface) — графический интерфейс пользователя.

Главными абстрактными классами являются Component, Container, MenuComponent и Graphics. Наряду с ними существуют классы FontMetrics, Image, PrintJob и Toolkit. Класс LayoutManager — это интерфейс, а остальные представляют собой классы, унаследованные от тех, которые изображены выше. Таким образом, класс Frame — это наследник класса Window, являющегося потомком класса Container, предок которого называется Component.

При таком разнообразии средств разработки пользовательского интерфейса трудно принять решение, с чего начать. Мы воспользуемся простой программой из главы 2, которая выводит на экран изображение флага, чтобы представить различные части пакета awt.

Swing

Swing — это еще один комплект инструментов GUI. В то время как пакет awt предоставляет общие абстрактные инструменты GUI, которые реализуются с использованием средств конкретной операционной системы, пакет Swing содержит инструменты, которые не зависят от операционной системы. Например, при использовании пакета awt окно и кнопка имеют вид и функции, свойственные подобным элементам на той машине, где выполняется программа. Объясняется это тем, что пакет awt ссылается на средства GUI операционной системы. Размер программ, использующих инструментарий пакета awt, уменьшается после компилирования, поскольку большая часть кода существует в операционной системе в готовом виде.

С другой стороны, компоненты пакета Swing полностью реализуются* в Java, поэтому вид и функции созданных с его помощью элементов (окон, кнопок) сохраняются независимо от того, на каком компьютере выполняется программа. Следовательно, хотя Swing-программы больше по размеру, они обладают существенным преимуществом, которое заключается в том, что конструкция GUI фиксируется разработчиком, и это предотвращает внезапный выход какого-либо элемента за рамку окна (например, потому что кнопки одной операционной системы больше, чем кнопки другой). Тем не менее существует возможность изменения вида и функций элементов GUI во время выполнения программы для обеспечения соответствия настройкам конкретной операционной системы.

Далее в книге мы будем рассматривать работу с пакетом awt, поскольку он прост в применении и невелик по объему, а принципы разработки GUI и обработки событий те же, что и при использовании классов пакета Swing. Мы все же выполним пример, демонстрирующий применение пакета Swing. Хорошее учебное пособие по использованию классов пакета Swing вы найдете на Web-узле компании Sun www.javasoft.com.

---

* Программы не используют средства GUI операционной системы, поэтому после компиляции байт-код содержит весь необходимый инструментарий для построения GUI. — *Прим. ред.*

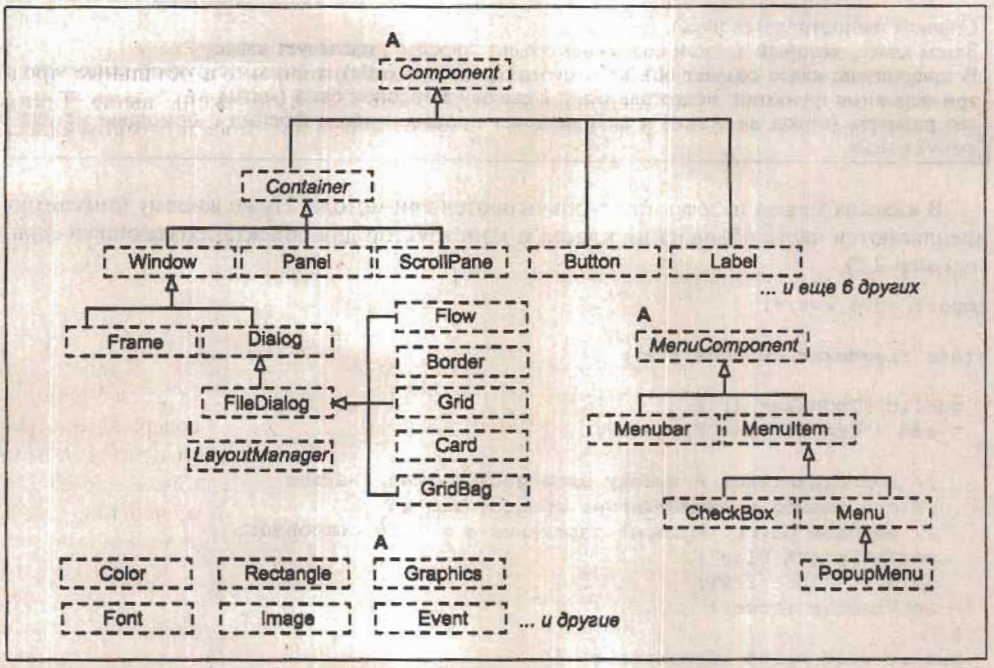


Рис. 10.1. Структура пакета `awt` (A — абстрактный класс, I — интерфейс)

10.2. Размещение графики в окне

Графический интерфейс реализуется посредством окон. В Java имеется класс `Window`, но на практике обычно используется один из его подклассов — `Frame`.

Frame — базисное окно

Приложение может иметь несколько окон, для каждого из которых объявляется класс, который наследует классу `Frame` и включает вызовы методов, необходимых для вывода информации в окно и выполнения других операций. Если используется одно окно (как в примере 2.2, когда мы выводили на экран флаг), главный класс программы и будет тем классом, который объявляет его. В следующем примере показано, как создается простое окно.

Создание окна (определение класса, наследующего класс `Frame`)

```
import java.awt.*;

class Classid extends Frame {

  Classid {
    setTitle ("title"); // дополнительный метод
    setSize (width, height);
    setVisible (true);
  }
… где-нибудь (например, в методе main) должен быть оператор:
  new Classid ();
```

Сначала импортируется пакет `java.awt`.
Затем класс, который должен содержать объект "фрейм", наследует классу `Frame`.
В завершение класс создает объект (экземпляр самого себя), а его конструктор выполняет три основные функции: подготавливает к выводу заголовок окна (метод `setTitle`), задает его размеры (метод `setSize`) и активизирует процесс вывода фрейма с помощью метода `setVisible`.

В классах `Frame` и `Component` объявляются три метода. Ниже вашему вниманию предлагаются часть объявления класса и конструктор для объекта, создающего флаг (пример 2.2).

```
import java.awt.*;

class FlagMaker extends Frame {

  public FlagMaker () {
    add ("Center",new Flags());

    // Подготавливает к выводу заголовок фрейма, задает
    // его размер и активизирует процесс вывода
    // методом paint, который определен в классе Component.
    setTitle ("A Flag");
    setSize (300, 200);
    setVisible (true);
  }
  public void paint (Graphics g) {...}
  public static void main (String args []) {
    new FlagMaker ();
  }
}
```

Добавление графики в окно

На данном этапе выполнения программы пакет `awt` использует метод `paint` для добавления в окно графики. Метод `paint` определяется:

- в классе, который расширяет класс `Frame` (см. пример 10.1);
- в другом объекте класса, который расширяет класс `Canvas` и добавляется к нашему "фрейму" (см. пример 10.2).

Вот описание метода `paint`, переопределяющего объявленный в классе `Component` метод:

Переопределение метода `paint`

```
public void paint (Graphics g) {
  Вызывает методы, определенные в Graphics,
  которые будут иметь префикс g.
}

public void repaint (Graphics g) {
  Вызывает методы, определенные в Graphics,
  которые будут иметь префикс g.
}
```

Метод `paint` вызывается, когда компонент становится видимым, а также когда окно, в котором он находится, скрыто и должно быть выведено заново.
Метод `repaint` может быть вызван программой для той же цели.

В процессе выполнения метод `paint` использует объект `Graphics`, адаптированный для платформы, на которой запущена программа. Это одно из преимуществ языка Java: используются графические элементы конкретной платформы, т.е. пакет `awt` выводит на экран компоненты, знакомые пользователю.

Класс `Graphics` довольно обширен, поэтому мы смогли дать характеристику только некоторым из его методов:

Спецификация класса `Graphics`

```
clearRect  (int x, int y, int width, int height)
copyArea   (int x, int y, int height, int width, int dx, int dy)
drawChars  (char [ ] data, int offset, int length, int x, int y)
drawLine   (int x1, int y1, int x2, int y2)
drawOval   (int x, int y, int width, int height)
drawRect   (int x, int y, int width, int height)
drawString (string str, int x, int y)
fillOval   (int x, int y, int width, int height)
fillRect   (int x, int y, int width, int height)
setColor   (color c);
setFont    (Font f);
// плюс 38 других
```

Пять методов черчения (`draw*`) широко распространены и позволяют выводить на экран геометрические фигуры и текст. Значениями параметров `x` и `y` являются выраженные в пикселях расстояния от верхнего левого угла экрана до заданной точки. Ширина (`width`) и высота (`height`) также измеряются в пикселях.
Метод `clearRect` "очищает" прямоугольную область экрана, а метод `copyArea` копирует содержимое некоторой области экрана в указанное место, относительные координаты которого задаются посредством параметров `dx` и `dy`.
`setColor` и `setFont` — это примеры простых методов, которые предназначены для настройки внешнего вида изображения.
Класс `Color` содержит константы для тринадцати стандартных цветов (см. ниже) наряду со средствами создания новых цветовых гамм.

К числу тринадцати стандартных цветов относятся следующие: `black`, `blue`, `cyan`, `darkGray`, `gray`, `green`, `lightGray`, `magenta`, `orange`, `pink`, `red`, `white` и `yellow`. Например, для черчения красного прямоугольника служит такой метод:

```
public void paint (Graphics g) {
  g.setColor (Color.red);
  g.fillRect (10, 10, 200, 100);
}
```

Рисование фигуры начинается с точки, находящейся на расстоянии 10 пикселей от левого верхнего угла экрана по вертикали и горизонтали. Ширина создаваемого прямоугольника составляет 200 пикселей, а высота — 100 пикселей (рис. 10.2).

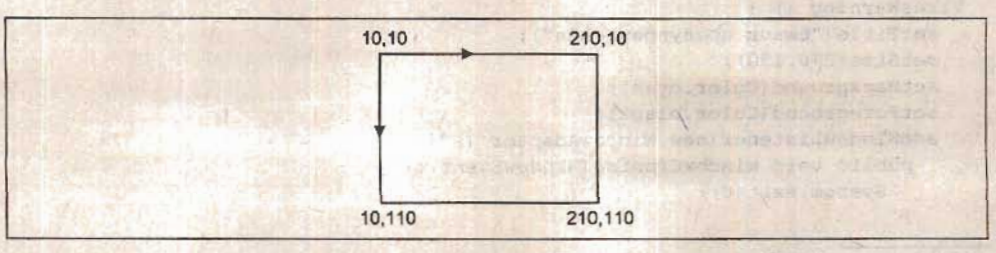


Рис. 10.2. Черчение прямоугольника с использованием методов пакета Graphics

Вывод текста в окне awt с помощью метода drawString

В процессе работы с пакетом `Graphics` используется метод `drawString`, который эквивалентен методу `println`. Например, оператор

```
g.drawString ("Hello Pierre:", 30,15);
```

выведет на экран строку "Hello Pierre", в которой левая нижняя точка буквы "H" находится на расстоянии 30 пикселей по горизонтали и 15 пикселей по вертикали от левого верхнего угла экрана.

Различие между методами `println` и `drawString` состоит в том, что последний позволяет выбрать шрифт, а также его начертание и размер. Впоследствии позиционирование начала строки может оказаться сложной задачей. Для создания универсальной программы, выводящей данные в удобочитаемом виде, следует опираться на относительные размеры символов. Абстрактный класс `FontMetrics` из пакета `awt` содержит методы, определяющие подходящие размеры символов, однако иногда в простых программах можно задавать и абсолютные значения в пикселях. Теперь мы готовы к рассмотрению первого в этой главе примера.

Пример 10.1. Вывод предупреждения о возможном заражении вирусом

Задача. Необходимо вывести на экран предупреждение о возможном заражении вирусом.

Решение. Мы воспользуемся окном, созданным с помощью пакета `awt`, методом `drawString` и описанными выше средствами для работы с прямоугольником. Чтобы сделать прямоугольник более броским, нам потребуется найти цветовое решение. Для работы с цветом предназначены методы `setForeground` и `setBackground` класса `Component`, а также цветовые константы, определенные в классе `Color`*. Мы остановим выбор на бирюзовом цвете, который соответствует константе `Color.cyan`.

Программа. Программа очень проста.

```
import java.awt.*;
import java.awt.event.*;

class VirusWarning extends Frame {
   /* Графическая программа.      Автор - Дж. М. Бишоп, октябрь 1996 г.
    *                    Java 1.1 - Т. Эбот, октябрь 1997 г.
    *                              Обновлена до 1.2 в мае 2000 г., Дж. М. Бишоп.
    * Выводит на экран предупреждающее сообщение, при
    * оформлении которого используются черный и бирюзовый цвета.
    * Демонстрирует, как назначать размеры окна и отобразить в нем
    * простую фигуру и текст, активизирует кнопку закрытия окна.
    */

  VirusWarning () {
    setTitle("Вывод предупреждения");
    setSize(270,150);
    setBackground(Color.cyan);
    setForeground(Color.black);
    addWindowListener(new WindowAdapter () {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
```

* Слово "colour" без "u" ("color") — это американский вариант написания.

```
    });
    setVisible(true);
  }

  public void paint (Graphics g) {
    g.setColor(Color.pink);
    g.fillRect(45, 30, 210, 90);
    g.setColor(Color.black);
    g.drawString("ПРЕДУПРЕЖДЕНИЕ",          90,  60);
    g.drawString("Обнаружен вирус",         65,  75);
    g.drawString("Перезагрузите компьютер", 70,  90);
    g.drawString("и уничтожьте вирус",      80, 105);
  }

  public static void main (String [] args) {
    new VirusWarning ();
  }
}
```

Тестирование. На рис. 10.3 показан результат работы программы (к сожалению, в оттенках серого цвета). Для остановки программы мы щелкаем мышью на кнопке закрытия окна, находящейся в правой части строки заголовка*. В заключение вызывается метод `System.exit(0)`. О том какие средства обеспечивают закрытие, рассказывается далее.



Рис. 10.3. Результат работы программы VirusWarning

Закрытие окна

В этом разделе речь пойдет о том, как завершается выполнение программы, работающей в окне. Один из способов — закрыть окно, щелкнув на специальной кнопке в правом углу. Щелчок мышью — это **событие**, которое может быть воспринято **слушателем** из подпакета `Event` пакета `awt`. В главном методе мы задаем ссылку на такой слушатель окна путем вызова метода `addWindowListener` и создания экземпляра новой версии адаптера окна. В реализации адаптера, которая рассматривается ниже, переопределяется метод класса `WindowAdapter`, называемый `windowClosing`, и выполняется соответствующее действие, в данном случае — вызов метода `System.exit(0)`.

Ниже описана последовательность определений и действий, связанных с закрытием окна.

---

* На рисунке показан результат работы программы на платформе Windows, который аналогичен выводу в Unix; на платформе Macintosh кнопка закрытия окна находится слева.

Закрытие окна

```
addWindowListener(new WindowAdapter () {
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
});
```

Класс `WindowAdapter` — это простой абстрактный класс, который реализует интерфейс `WindowListener` и предоставляет абстрактные методы для различных содержащихся в нем методов. Мы можем решить, какой из них переопределить. В данном случае из семи имеющихся методов нас интересует только `windowClosing`. Создание экземпляра объекта `WindowAdapter` и переопределение метода `windowClosing` осуществляются при передаче параметра методу `addWindowListener`.

В этом описании все слова являются или ключевыми (выделены полужирным шрифтом), или идентификаторами, которые уже определены в пакете `awt` (набраны обычным шрифтом), за исключением параметра (события "`e`"), который не используется и поэтому всегда может оставаться неизменным. Код описания можно рассматривать как "мантру" и помещать в главный метод любых GUI-программ. В коде используется анонимный класс, представленный в разделе 8.2. Применению анонимного класса существует альтернатива: методу `addWindowListener` в качестве параметра можно предоставить выражение `new x()`, а затем определить `x` как локальный класс с заголовком, в котором указано, что данный класс расширяет класс `WindowAdapter`, и телом, являющимся реализацией метода `windowClosing`. Очевидно, что в любом случае код анонимного класса выглядит компактнее.

Пример 10.2. Гистограмма уровней осадков в Саванне

Задача. Требуется начертить гистограмму уровней осадков, выпавших за год на территории Саванны, основываясь на данных из примера 6.6.

Решение. Мы возьмем данные, которыми оперировали в примере 6.6, определим уровень осадков для каждого месяца и зададим вывод на экран столбцов гистограммы.

Алгоритм. Создаваемый класс будет наследовать класс `Frame`. Для того чтобы вывести окно, в главную программу мы включим те же инструкции, которые использовались в предыдущей.

Конструктор будет иным. Мы воспользуемся им для ввода всех данных, необходимых для решения текущей задачи (ранее с этой целью применялся метод `readIn`). Однако содержимое не меняется. Наиболее важная часть программы связана с отображением диаграммы. Необходимо получить на выходе такую же гистограмму, как на рис. 10.4.

Рассмотрим составные части диаграммы поочередно.

1. **Оси.** Точка отсчета имеет следующие координаты: x = 50, y = 300. Данные значения выражены в пикселях. Направление отсчета то же, что и на рис. 10.2.
2. **Столбцы.** Для вывода столбцов применяется метод `fillRect`. Мы должны предоставить параметры, указанные в таблице "Спецификация класса `Graphics`", которая опубликована в начале данного раздела. Другими словами, нам нужны: точка в основании столбца, ширина и высота. Шириной будет некоторая константа (не число, объявленное как константа, а повторяющееся значение для вывода столбцов одинаковой ширины), например 20. Высотой — фактическое значение уровня осадков за месяц, умноженное на 10 (значение переменной "a").

Таким образом обеспечивается и легкость восприятия диаграммы (10-ти мм осадков соответствуют 100 пикселей). Начальной точкой прямоугольника послужит координата "y-a", где "y" соответствует координате базовой линии (т.е. 300, как определено выше). Сложность при нахождении начальной точки координаты "x" связана с тем, что она специфична для каждого месяца. Формула, в которой учитываются ширина столбцов и интервал между ними, выглядит следующим образом:

```
g.fillRect(month*(width+gap)+gap+x, y-a, width, a);
```

3. **Метки.** Градуировка осей осуществляется посредством двух циклов, принципы работы которых легко понять. Однако их самостоятельная разработка потребует некоторых усилий. Для градуировки оси "x" используется массив строк, в которых хранятся названия месяцев.
4. **Заголовок.** При выводе на экран заголовка мы увидим, как изменять шрифты средствами Java. Имеется класс `Font`, экземпляр которого может быть создан с тремя параметрами: название шрифта, начертание и размер. Доступны такие шрифты: `Serif`, `SansSerif`, `Monospaced`, `Dialog` и `DialogInput`. Применяться будут стили `PLAIN`, `BOLD` и `ITALIC`. Размер шрифта выражается в пунктах. Нормальным считается размер в 12 пунктов. Если же текст должен быть крупным, можно установить размер в 24 пункта. Не рекомендуем применять шрифт, размер которого меньше 8 пунктов. Поскольку измененный шрифт используется последним из используемых в программе методом `drawString`, восстанавливать его не потребуется.



Рис. 10.4. Результат выполнения программы "Гистограмма уровней осадков"

Программа. Настоящая программа демонстрирует, что графика может быть очень мощным инструментом, если одни и те же элементы используются многократно. Но когда каждая деталь требует индивидуальной обработки и при этом используется только один раз, программирование становится довольно трудоемким процессом. Подтверждением сказанному служат следующие примеры.

```
import java.io.*;
import javagently.*;
import java.awt.*;
import java.awt.event.*;
import myutilities.*;  // для импортирования класса Stats

class WeatherChart extends Frame {
   /* Программа "Гистограмма уровней осадков".  Автор - Дж. М. Бишоп, декабрь 1997 г.
    *                                           Java 1.1
    *                                           Обновлена в мае 2000 г.
    * Выводит на экран гистограмму уровней осадков для каждого месяца на основе
    * данных, собранных за несколько лет. Данные должны быть
    * представлены в следующем формате: значение года, за которым следуют 12 цифр,
    * соответствующих количеству осадков в каждом месяце этого года.
    * Демонстрирует, как создаются несложные графические объекты.
    * Использует класс Stats из пакета myutilities.
    */

  int           base = 1950;
  int           startYear, endYear, nYears = 0;
  double[] []   rainTable = new double[12] [70];
  String        months [] = {"Янв", "Фев", "Мар", "Апр", "Май", "Июн",
                             "Июл", "Авг", "Сен", "Окт", "Ноя", "Дек"};

  WeatherChart () throws IOException {
    getData ();
    setTitle("Гистограмма уровней осадков");
    setSize(400,350);
    setVisible(true);
    addWindowListener(new WindowAdapter () {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });
  }

  public void paint (Graphics g) {
    int x = 50;
    int y = 300;
    int width = 20;
    int gap = 5;
  // Отображение осей.
    g.drawLine (x,y,x+12*(width+gap),y);
    g.drawLine (x,y,x,30);
  // Вывод меток на осях.
    for (int m = 0; m < 12; m++)
      g.drawString(months[m],m*(width+gap)+gap+x,y+20);
    for (int i = 0; i < y; i+=100)
      g.drawString(String.valueOf(i), 20,y-i);
  // Заголовок.
    Font heading = new Font("SansSerif",Font.BOLD,14);
    g.setFont(heading);
    g.drawString("Уровни осадков в Саванне",120,40);
    g.setColor(Color.cyan);
```

```
// Столбцы
  for (int month = 0; month < 12; month++) {
    int a = (int) Stats.mean
            (rainTable[month], nYears)*10;
    g.fillRect(month*(width+gap)+gap+x, y-a,width, a);
  }
}

void getData () throws IOException {
  Stream fin = new Stream ("rain.dat", Stream.READ);

  int actualYear = 0;        /* например, 1997 */
  int yearIndex = 0;         /* например, 0 */
  try {
    while (true) {
      actualYear = fin.readInt();
      if (yearIndex == 0)
        startYear = actualYear;
      for (int m = 0; m < 12; m++)
        rainTable[m][yearIndex] = fin.readDouble();
      yearIndex++;
    }
  } catch (EOFException e) {
  // Считывает значение последнего года, данные за который будут
  // введены в программу.
  endYear = actualYear;
  nYears = endYear-startYear+1;
  }
}

public static void main (String [] args) throws IOException {
  new WeatherChart ();
}

}
```

Использование холста

При создании сложной графики необходимы средства группировки отдельных графических элементов в один объект. Этой цели служит холст (объект класса *Canvas*). Холст может быть помещен во фрейм и раскрашен посредством соответствующего метода этого фрейма, а может использоваться как неизменное изображение.

Пример 10.3. Несколько флагов

Задача. С помощью программы "Создание флагов" нужно вывести на экран разнообразные флаги.

Решение. Многие флаги создаются по той же модели, что и флаг Германии, т.е. за счет вывода трех горизонтальных полос разного цвета. Например, флаг Нидерландов состоит из красной, белой и голубой полос, а флаг Эфиопии — зеленой, желтой и красной. Можно сделать так, что цвета будут указываться как параметры в процессе создания изображения флага. Метод, выполняющий раскрашивание, в результате будет опираться на эти параметры, а не на абсолютные значения.

Программа. Создается класс `Flags`, расширяющий класс `Canvas` и содержащий исправленный конструктор и метод раскраски. Для каждого флага формируется отдельный фрейм, чтобы любой из этих объектов имел свое окно. С целью выполнения описанных операций разрабатывается метод `makeFlag`, который создает экземпляр нового фрейма, а затем производит процедуры создания заголовка, назначения размера и т.д. Но в этот раз мы явно добавляем во фрейм холст как компонент.

Рассмотрим еще один аспект многооконной программы. Разместить фрейм в позиции с координатами `x` и `y` позволяет метод:

```
setLocation(x, y);
```

Для этого надлежит пронумеровать флаги и расположить окна ступенчато, в направлении к правому нижнему углу экрана (рис. 10.5).

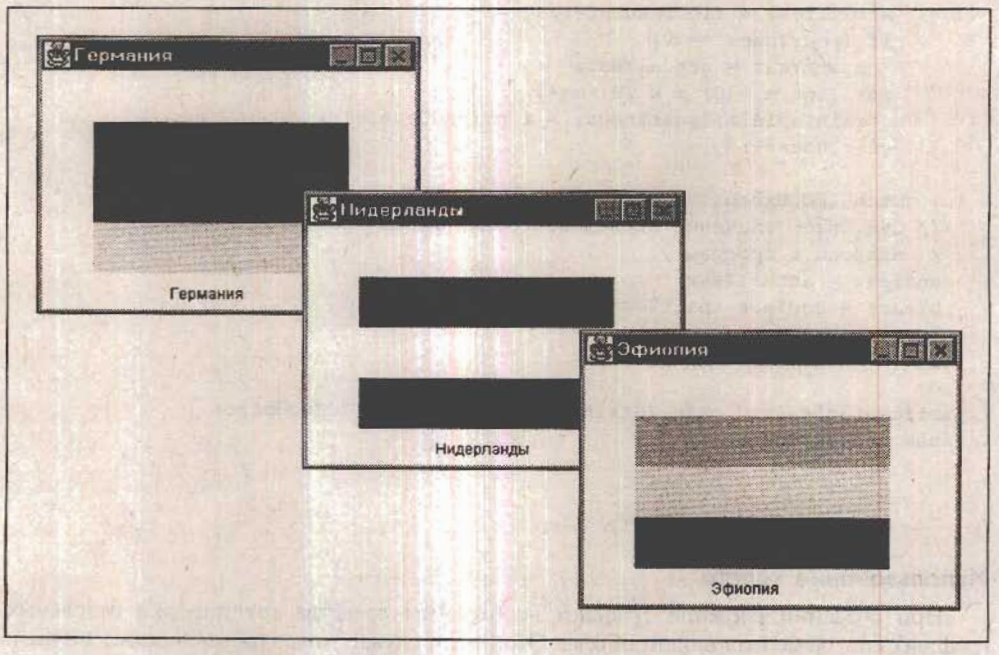


Рис. 10.5. Вывод программы FlagMaker2

Итак, программа*:

```
import java.awt.*;
import java.awt.event.*;

public class FlagMaker2 extends Frame {

   /** Программа для черчения флагов.
    *                                  Автор - Дж. М. Бишоп, апрель 2000 г.
    *   = = = = = = = = = = = = = =
```

* Поскольку в блоке закрытия окна используется оператор `e.getWindow().dispose();`, для выхода из программы нужно после закрытия всех трех окон нажать клавиши [Ctrl-C]. — *Прим. ред.*

```
    * Пригодна для вывода нескольких флагов,
    * а также различных конструкций.
    * Демонстрирует наследование.
    **/

  public FlagMaker2() {
    FlagCanvas flag;
    flag = new FlagCanvas
        ("Германия",Color.black,Color.red,Color.yellow,0);
    makeFlag(flag);
    flag = new FlagCanvas
        ("Нидерланды",Color.red,Color.white,Color.blue,1);
    makeFlag(flag);
    flag = new FlagCanvas
        ("Эфиопия",Color.green,Color.yellow,Color.red,2);
    makeFlag(flag);
  }

  public static void main (String [] args) {
    new FlagMaker2();
  }
  public static void makeFlag (FlagCanvas canvas) {
    Frame f = new Frame();
    f.add(canvas);
    f.setTitle(canvas.country);
    f.setSize(300,220);
    f.setLocation(200*canvas.flagNo,20*canvas.flagNo);
    f.setVisible(true);
    f.addWindowListener(new WindowAdapter () {
      public void windowClosing(WindowEvent e) {
        e.getWindow().dispose();
      }
    });
  }
}

// = = = = = = = = = = = = = = = = = = = = = = =
class FlagCanvas extends Canvas {

  String country;
  Color bar1, bar2, bar3;
  int flagNo;

  FlagCanvas (String c, Color b1, Color b2, Color b3, int f) {
    country = c;
    bar1 = b1;
    bar2 = b2;
    bar3 = b3;
    flagNo = f;
  }

  public void paint (Graphics g) {
    // Формирует флаг из цветных прямоугольников.
    g.setColor (bar1);
```

```
        g.fillRect (40, 40, 200, 40);
        g.setColor (bar2);
        g.fillRect (40, 80, 200, 40);
        g.setColor (bar3);
        g.fillRect (40, 120, 200, 40);
        // Отображение метки (названия страны).
        g.setColor (Color.black);
        g.drawString(country,100,180);
    }
}
```

Методы и классы пакета awt: резюме

В завершение раздела поговорим о том, какие классы пакета `awt` и какие методы этих классов наиболее часто употребляются. В наших программах применялись главным образом классы `Frame`, `Window`, `Container`, `Component` и `Graphics`. Наибольшее распространение получили классы `Component` (главный класс пакета `awt`) и `Graphics`. Метод `setTitle` можно получить из класса `Frame`, а класс `Container` используется только в тех случаях, когда планируется применение группирующих компонентов, как показано в следующем разделе. Отношения между классами в пакете `awt` представлены на рис. 10.6.

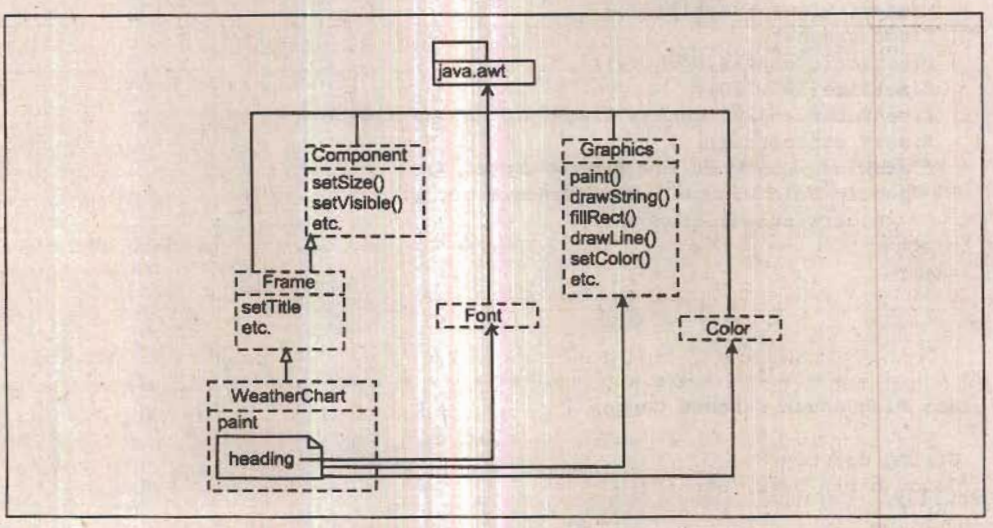


Рис. 10.6. Схема к программе WeatherChart, на которой показаны отношения между классами пакета `awt`

10.3. Компоновка пользовательского интерфейса

Обработка графики — это всего лишь один из аспектов GUI. Другой — создание легко реагирующей среды, с помощью которой пользователь сможет взаимодействовать с программой. Для этой цели Java предоставляет множество компонентов (*рис. 10.1*). Эти компоненты не реализуются методом `paint`, а создаются как объекты с определенными значениями конструктора, после чего добавляются во фрейм в соответствии с действительной схемой размещения.

Наиболее простые компоненты — это класс Label, предоставляющий возможность выводить фрагменты текста, и класс Button, который является базисным средством для взаимодействия с программой. Пакет awt содержит много таких компонентов, некоторые из них мы рассмотрим в следующих разделах.

Класс Label обеспечивает вывод простого текста, передаваемого в качестве параметра его конструктору:

Создание объекта Label

```
add(new Label ("some text"));
```

Метка (объект Label) создается с заданным текстом и добавляется в текущий контейнер (обычно текущий фрейм).

Например, вывести на экран метку **Школа:** позволяет следующий оператор:

```
add(new Label ("Школа: "));
```

Объект Button создается аналогичным образом, но при этом имеется дополнительное средство, позволяющее реагировать на событие (т.е. мы щелкнем на кнопке, которая отображена на экране, а программа совершит некоторое действие). Вот как протекает процесс создания кнопки:

Создание объекта Button

```
// объявляется компонент
Button buttonid;

// создается экземпляр объекта
buttonid = new Button ("some text");
 add(buttonid);
```

Объявляется объект buttonid, а потом создается экземпляр объекта с заданным текстовым значением.
Затем этот компонент добавляется в текущий контейнер (обычно текущий фрейм).

Поскольку метод add определяется в классе Container, мы выполняем следующее действие — добавляем кнопку в контейнер, назначенный по умолчанию, т.е. во фрейм, с которым работаем. Скоро вы узнаете, как объявлять другие контейнеры.

Различие между объявлениями объектов Label и Button состоит в том, что объекты Label пассивные: для них нельзя задать выполнение какого-нибудь действия. Вот почему редко возникает необходимость делать их объектными переменными (использовать какой-либо идентификатор в качестве имени). С другой стороны, можно утверждать, что в дальнейшем к кнопкам будут обращаться, поэтому они должны быть представлены переменными с постоянными идентификаторами. Это также тот случай, когда идентификатор используется вне конструктора, поэтому объектную переменную мы объявляем перед созданием кнопки.

Например, для создания кнопки submitButton и размещения ее в окне можно использовать следующий код:

```
Button submitButton;

submitButton = new Button ("Запустить");
  add(submitButton)
```

Почему нет параметров, задающих координаты x и y фрейма, в который добавляются компоненты? Причина заключается в том, что с этими компонентами (в отличие от графических элементов) работают **менеджеры размещения**.

Менеджеры размещения

Менеджеры размещения контролируют, как выполняется позиционирование компонентов, добавляемых в окно, а также их упорядочение. Если пользователь изменяет размер окна, менеджер размещения переупорядочивает компоненты в новой области так, чтобы они оставались видимыми и в тоже время сохранили свои позиции относительно друг друга.

В Java насчитывается пять таких менеджеров, но мы рассмотрим только три из них: `FlowLayout`, `BorderLayout` и `GridLayout`. Все они реализуют интерфейс `LayoutManager`, как показано на рис. 10.1. Этот интерфейс входит в пакет `aw` и становится доступным при его импортировании. Рассмотрим, как же внедряется менеджер размещения:

Подключение менеджера размещения
`setLayout (new Manager(parameters));`
Один из пяти менеджеров размещения реализуется как объект; его экземпляр создается и передается методу `setLayout` текущего контейнера (например, `Frame`), который затем фиксирует, как должны быть расположены компоненты.

По умолчанию в Java-программе применяется менеджер размещения `BorderLayout`, но для наших целей больше подходит `FlowLayout`. Он также является менеджером по умолчанию для апплетов. Вот пример назначения менеджера размещения:

```
setLayout (new FlowLayout(FlowLayout.CENTER,horigap,vertigap));
```

Первый параметр указывает, что элементы, добавляемые во фрейм, должны быть помещены в центр (они также могут быть выровнены по левому или правому краю). Параметры `horigap` и `vertigap` — это константы, которые служат для указания минимального расстояния (в пикселях) между элементами во фрейме. Все эти параметры являются дополнительными, а константа `CENTER` назначена по умолчанию. Примеры использования менеджера `FlowLayout` вы найдете в следующих программах.

Реакция на нажатие кнопки

Мы должны не только предусмотреть возможность закрытия окна, но и заставить программу реагировать на нажатие кнопки. В пакете `awt.event` наряду со слушателем `windowListener` определен также `actionListener`. Этот интерфейс-слушатель имеет только один метод для реализации — `actionPerformed`. Поэтому кнопки будут связаны посредством ссылок со слушателем `actionListener` и иметь версии метода `actionPerformed`. Установка ссылки выполняется в конструкторе немедленно после того, как кнопка объявляется и добавляется в контейнер. Три оператора для кнопки запуска (`submitButton`) выглядят следующим образом:

```
submitButton = new Button ("Запуск");
  add(submitButton)
  submitButton.addActionListener (this);
```

Ссылка на `this` указывает, что *текущий* фрейм отвечает за определение метода `actionPerformed`. На схеме для такой ссылки полезно выделить компоненты GUI и слушатели из массы других классов. В качестве обозначений на схеме мы выбрали для компонента — прямоугольник с двойной рамкой, для слушателя — конус, а для обработчика событий — стандартный символ метода (рис. 10.7).

Метод `actionPerformed` имеет один параметр — объект `ActionEvent`, к которому можно обратиться, чтобы выяснить, совпадает ли он с одной из кнопок. Вот в какой последовательности выполняется опрос:

Реакция на нажатие кнопки

```
public void actionPerformed (ActionEvent e) {
  if (e.getSource () == buttonname1) {
    statements
  }.else
  if e.getSource () == buttonname2) {
    statements
  } // и т.д.
}
```

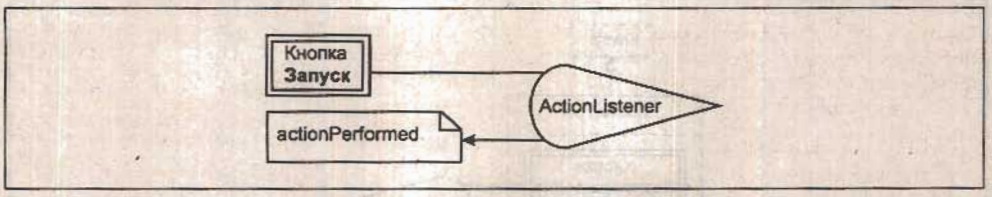


Рис. 10.7. Блок-схема, на которой показана связь между кнопкой, слушателем и обработчиком событий

Предположим, несколько кнопок имеют различные строковые метки. Единственный способ отличить их друг от друга состоит в использовании последовательности операторов `if-else`, учитывающих все варианты. Нажатие кнопки — это только одно из событий, поэтому схема, подобная описанной, может пригодиться нам в дальнейшем. Обработке других событий посвящен раздел 11.2.

Применение отступов

До этого времени при написании кода мы использовали отступы традиционным способом, унаследованным от более старых языков, таких как Pascal и C: отступ создавался каждый раз при вводе нового метода или блока операторов. Операторы внутри одного блока имели одинаковые отступы.

При программировании GUI часто встречаются очень длинные последовательности операторов, вовлеченных в вывод компонентов на экран. Каждый компонент может иметь три или более оператора, которые ассоциированы с ним. Не существует установленного порядка, согласно которому должны выполняться операторы, но обычно мы работаем с каждым компонентом по очереди и следуем схеме создание-ссылка-добавление. Исходя из этого для программ, разработанных средствами Java, была внедрена новая схема использования отступов, в соответствии с которой создание компонента рассматривается как введение нового уровня. Благодаря этому легко увидеть все операторы, относящиеся к данному компоненту. Этот принцип уже использовался в приведенном выше примере (добавление кнопки `submitButton`) и применяется также в следующих двух упражнениях.

Пример 10.4. Предупреждение с двумя ответами

Задача. Усовершенствуйте программу, которая отображает сообщение о том, что обнаружен вирус, задав вывод двух кнопок в окне сообщения. С помощью одной кнопки пользователь сможет подтвердить то, что сказано в сообщении. На этой стадии все окно должно стать красным. Вторая кнопка должна имитировать перезагрузку компьютера.

Алгоритм. Используйте для достижения необходимого результата потоковый менеджер размещения `FlowLayout`, а также метки и кнопки. Выберите цвета для фона и переднего плана с помощью класса `Color`. Действием, связанным с кнопкой **Перезагрузить**, должно быть закрытие окна. При нажатии кнопки **Подтверждение** должен измениться цвет фона. Конструктор устанавливает необходимую связь между компонентами и слушателями. Слушатель, в свою очередь, активизирует метод `actionPerformed`, который реагирует на события.

Программа. Программа демонстрирует последовательность программирования GUI. Сообщение, которое будет выведено на экран, объявляется как массив строк. Каждая строка текста выводится на экран путем добавления ее во фрейм в качестве метки. Схема, соответствующая программе, показана на рис. 10.8.

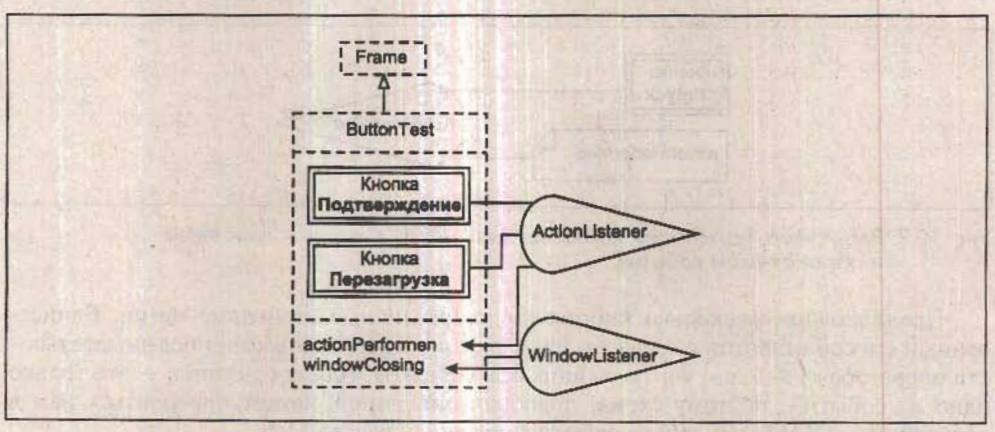


Рис. 10.8. Схема программы `ButtonTest`

```
import java.awt.*;
import java.awt.event.*;

class ButtonTest extends Frame implements ActionListener{

   /* Графическая программа.          Автор - Дж. М. Бишоп, октябрь 1996 г.
    * Выводит  предупреждение.      Java 1.1 - Т. Эбот, октябрь 1997 г.
    *                          Обновлена до 1.2 в мае 2000 г., Дж. М. Бишоп
    * Выводит на экран предупреждающее сообщение,
    * используя черный и бирюзовый цвета. После нажатия
    * кнопки "Подтверждение" окно становится красным.
    * Демонстрирует использование кнопок и слушателей, а также
    * обработку событий.
    */

  Button waitButton;
```

```
 Button rebootButton;

 String[] message = {
   "П Р Е Д У П Р Е Ж Д Е Н И Е",
   "     Обнаружен вирус.      ",
   "Перезагрузите компьютер",
   "и уничтожьте вирус." };

ButtonTest () {
/* Конструктор отвечает за вывод
 * первоначального фона кнопок и окна.
 */
  setBackground(Color.cyan);
  setForeground(Color.black);
  setLayout(new FlowLayout(FlowLayout.CENTER, 15, 10));
  for (int i = 0; i < message.length; i++)
    add(new Label(message[i]));
  waitButton = new Button("Подтверждение");
    waitButton.addActionListener(this);
    add(waitButton);
  rebootButton = new Button("Перезагрузка");
    rebootButton.addActionListener(this);
    add(rebootButton);
  setTitle("Вывод предупреждения");
  setSize(220,150);
   setBackground(Color.cyan);
   setForeground(Color.black);
   addWindowListener(new WindowAdapter () {
     public void windowClosing(WindowEvent e) {
       System.exit(0);
     }
   });
   setVisible(true);
}

public void actionPerformed(ActionEvent e) {
  if (e.getSource() == rebootButton) {
    setVisible(false);
    dispose();
    System.exit (0);
  } else
  if (e.getSource() == waitButton) {
    setForeground(Color.white);
    setBackground(Color.red);
  }
}

public static void main (String[] args) {
  new ButtonTest ();
}
}
```

Тестирование. Окно, выведенное на экран после выполнения первого этапа работы программы, показано на рис. 10.9.

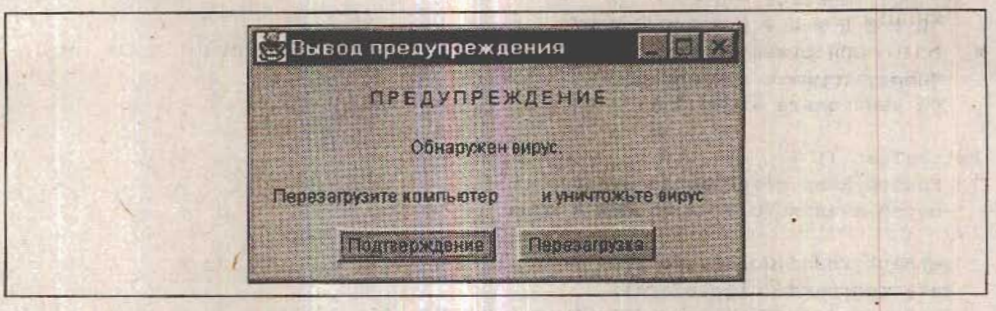


Рис. 10.9. Результат работы программы `ButtonTest`

Другие менеджеры размещения

Наряду с потоковым менеджером `FlowLayout` в Java имеются другие менеджеры размещения, на описании которых мы и остановимся.

- `BorderLayout` позволяет позиционировать элементы (полосы прокрутки, меню, кнопки и т.д.) в областях фиксированного размера, граничащих со сторонами фрейма, которые обозначаются параметрами `North`, `South`, `East` и `West` (что соответствует верхней, нижней, правой и левой границам окна). Остальное пространство обозначается как `Center`.
- `CardLayout` создает вкладки, содержимое которых отображается при выполнении щелчка на заголовке.
- `GridLayout` разделяет фрейм на заданное количество рядов и колонок.
- `BoxLayout` размещает некоторое количество компонентов по вертикали или горизонтали. На способ расположения компонентов изменение размеров фрейма не влияет. Хотя этот менеджер принадлежит к пакету `Swing`, он полностью совместим с пакетом `awt`.
- `GridBagLayout` предоставляет возможность выполнять тонкую настройку, в процессе которой для каждого компонента задаются точные координаты в пикселях. Данный менеджер сложен в использовании, но в то же время является наиболее универсальным.

Поскольку `LayoutManager` — это интерфейс, вы можете определить собственный менеджер размещения.

Характеристики других компонентов

Метки и кнопки — это компоненты. Мы упоминали, что существуют и другие подобные классы. Ниже дан их обзор.

- Класс `TextComponent` вместе с двумя подклассами `TextArea` и `TextFiled` позволяет работать с текстом, состоящим из большого количества строк, выделять текст и редактировать его.
- Класс `ScrollBar` тоже предназначен для работы с текстом и предоставляет возможность просматривать в окне содержимое контейнера.
- Класс `ScrollPane` — это контейнер, который позволяет компоненту, имеющему большую площадь, перемещаться под ним с помощью полос прокрутки так, что в конкретный момент времени виден только участок этого компонента.

- Класс `Canvas` — это дополнительная область окна, которая может использоваться для отображения других объектов, чтобы не возникали помехи при работе с кнопками.
- Благодаря классу `Checkbox` можно выбрать один из двух вариантов для какого-либо действия — активизировать или отменить.
- Класс `Choice` предоставляет раскрывающиеся списки, из которых может быть сделан выбор. Допускается выбор только одого варианта.
- Класс `Menus` предназначен для создания в окне строки с ниспадающими меню. В каждый момент времени может быть выбран только один пункт меню. Некоторые опции можно сделать недоступными для выбора.
- Класс `PopupMenu` позволяет создать в любом месте окна контекстное меню, содержащее ряд подменю. Его опции также можно сделать недоступными для выбора.
- Класс `Print` обеспечивает вывод на принтер всего окна или его части.

Панели для группирования

Java предоставляет средства для группирования компонентов в окне, которые можно будет перемещать совместно без риска, что один из них перекроет другой. Примером является панель инструментов. Панель передается менеджеру размещения как один элемент. Так, кнопки **Запустить** и **Очистить** могут быть сгруппированы следующим образом:

```
Panel p = new Panel ();
  p.add (new Button ("Запустить"));
  p.add (new Button ("Очистить"));
add(p);
```

(Обратите внимание на то, как в коде используются отступы перед операторами.) Затем панель может быть добавлена в конструктор фрейма. Если требуется, чтобы кнопки всегда располагались в нижней части экрана, следует воспользоваться менеджером размещения `BorderLayout` и поместить панель в область `South`. Менеджер размещения `BorderLayout` по умолчанию назначен для фреймов, а `FlowLayout` — для панелей. Поэтому их не нужно будет вызывать до тех пор, пока не потребуется изменить, например, расстояние между компонентами. При добавлении компонента с использованием менеджера `BorderLayout` в качестве первого параметра указывается одна из пяти областей (строковых констант), показанных на рис. 10.10.

Области у границ окна узкие, поэтому в них может быть размещен только один компонент: ряд кнопок, заголовок или полоса прокрутки. Вот как может выглядеть конструктор, эквивалентный приведенному в примере 10.4:

```
ButtonTest(0 {
  // По умолчанию используется менеджер BorderLayout.

  setBackground(Color.cyan);
  setForeground(Color.black);

  Panel m = new Panel ();
    for (int i = 0; i < message.length; i++)
      m.add(new Label(message[i]));
    add ("Center", m);
  Panel p = new Panel ();
    // По умолчанию используется менеджер FlowLayout.
    waitButton = new Button("Подтверждение")
```

```
    waitButton.addActionListener(this);
    p.add(waitButton);
  rebootButton = new Button("Перезагрузка");
    rebootButton.addActionListener(this);
    p.add(rebootButton);
  add ("South", p);

  // Другие операторы, относящиеся ко фрейму.
}
```

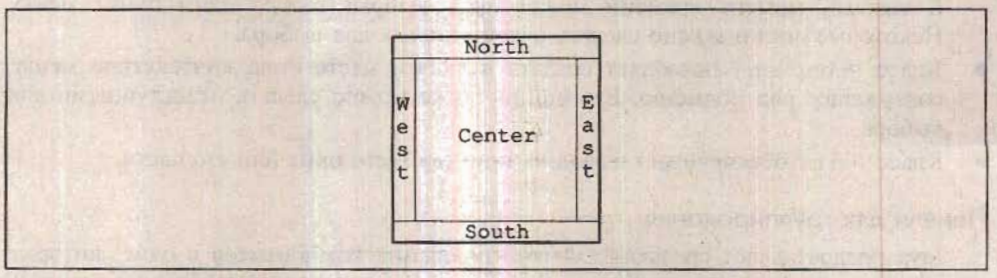


Рис. 10.10. Названия областей для менеджера размещения `BorderLayout`

Пример 10.5. Светофор

Задача. Управление безопасности движения Саванны распорядилось создать симулятор светофора и проследить, как нажатие пешеходами кнопок на переходе воздействует на изменение цветов.

Решение. Необходимо создать и вывести на экран окно, похожее на то, которое изображено на рис. 10.11. Затем должен быть разработан алгоритм изменения цветов светофора в результате нажатия кнопки. В процессе программирования мы будем использовать события. Следует отметить также, что разработка настоящей программы будет продолжена в разделе 11.1.

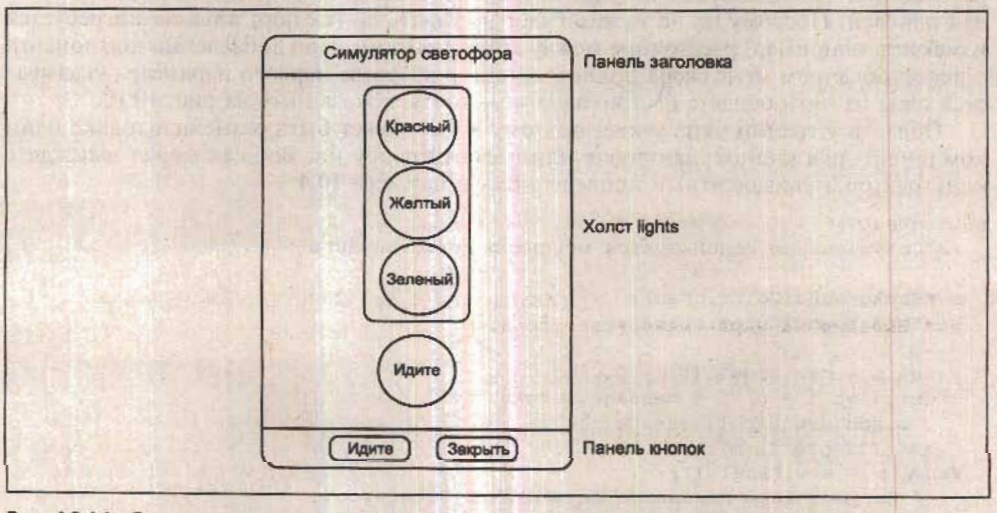


Рис. 10.11. Схема размещения элементов на экране при симуляции светофора

Конструирование класса. Поместите изображение светофора на холст, используя методы класса `Graphics`. Кнопки должны находиться на одной панели, а заголовок — на другой. Схема для программы представлена на рис. 10.12. Чтобы не усложнять схему, мы решили не включать в нее слушатель для события закрытия окна. Обратите внимание на то, что кнопка **Идите** на схеме еще ни с чем не связана.

Класс `Traffic0` расширяет класс `Frame` для вывода на экран графики. Объекты `buttons` и `title` являются экземплярами класса `Panel`. Класс `lights` — это объект, объявленный как `private`, который представляет собой экземпляр небольшого класса-члена `LightCanvas`. Класс `LightCanvas` наследует классу `Canvas` и имеет единственный метод `paint`, который вызывает методы, определенные в классе `Graphics`.

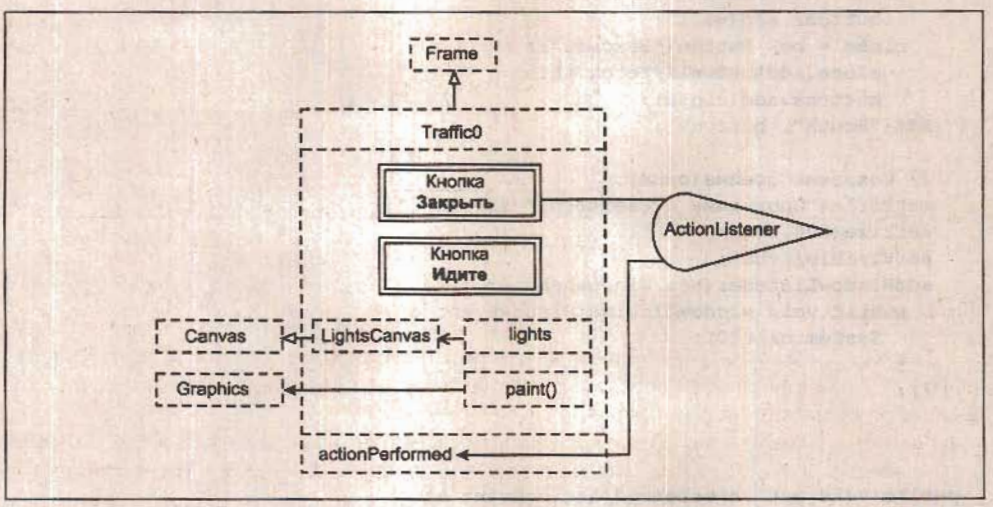


Рис. 10.12. Схема эмуляции светофора

В этой программе кнопка **Закрыть** связана с соответствующим слушателем, а для кнопки **Идите** еще не задано действие. Это будет сделано в следующей главе.

Программа

```
import java.awt.*;
import java.awt.event.*;

class Traffic0 extends Frame implements ActionListener {

   /* Программа "Светофор", вариант 1.
    *                              Автор - Дж. М. Бишоп, октябрь 1996 г.
    *                              Java 1.1 - Т. Эбот, октябрь 1997 г.
    *                              Обновлена в мае 2000 г., Дж. М. Бишоп.
    * Выводит на экран макет светофора.
    *
    * Кнопка "Идите" еще не активизирована.
    *
    * Демонстрирует использование панелей, холстов и
    * менеджера размещения BorderLayout.
    */
  LightsCanvas lights;
```

```
Button close;
Button walk;

Traffic0() {
  // Добавление компонентов.
  Panel title = new Panel();
    title.add (new Label("Симулятор светофора"));
  add("North", title);
  lights = new LightsCanvas();
    add("Center", lights);
  Panel buttons = new Panel();
    walk = new Button("Идите");
      buttons.add(walk);
    close = new Button("Закрыть");
      close.addActionListener(this);
      buttons.add(close);
  add("South", buttons);

  // Создание фрейма(окна).
  setTitle("Программа \"Светофор\" версия 0");
  setSize(300,200);
  setVisible(true);
  addWindowListener(new WindowAdapter () {
    public void windowClosing(WindowEvent e) {
      System.exit(0);
    }
  });

}

public void actionPerformed(ActionEvent e) {
  // В следующей версии программы здесь должны обрабатываться действия,
  // связанные с кнопкой "Закрыть".
  setVisible(false);
  dispose();
  System.exit(0);
}

public static void main (String [] args) {
  new Traffic0 ();
}

class LightsCanvas extends Canvas {
  public void paint(Graphics g) {
    g.drawOval(97, 10, 30, 68);
    g.setColor(Color.red);
    g.fillOval(105, 15, 15, 15); // красный
    g.setColor(Color.yellow);
    g.fillOval(105, 35, 15, 15); // желтый
    g.setColor(Color.green);
    g.fillOval(105, 55, 15, 15); // зеленый
    g.fillOval(105, 85, 15, 15); // "идите"
    g.setColor(Color.black);
    g.drawString("КРАСНЫЙ", 15, 28);
```

```
        g.drawString("ЖЕЛТЫЙ", 15, 48);
        g.drawString("ЗЕЛЕНЫЙ", 15, 68);
        g.drawString("ИДИТЕ", 15,98);
    }
  }
}
```

Тестирование. Поскольку рисунок в книге представлен в черно-белом формате, мы можем выделить контуры каждой цветной области. Для этого следует добавить в программу код, в котором применяется метод `drawOval`, выводящий в качестве окантовки черный круг. Этот метод должен устанавливать те же координаты для центра круга, что и соответствующий метод `fillOval`, только ширина и длина должны быть на один пиксель больше. Например:

```
g.setColor (Color.red);
g.fillOval (105, 15, 15, 15);
setColor (Color.black);
g.drawOval (105, 15, 16, 16);
```



Рис. 10.13. Окно программы эмуляции светофора

Резюме

Класс `Graphics` позволяет выводить геометрические фигуры, строки и изображения, а также задавать шрифты, цвета и т.д. Объектами класса `Component` являются такие элементы, как кнопки, текстовые поля, меню и полосы прокрутки. Они могут быть помещены в контейнеры и выводиться на экран надлежащим образом посредством одного из менеджеров размещения. Подпакет `java.awt.event` служит для обработки внешних событий, таких как нажатие кнопок и закрытие окна, с помощью набора обработчиков, слушателей и адаптеров. Подпакет `java.awt.image` пакета `awt` используется для внедрения изображений в разнообразных форматах.

Существуют различия между программированием отображения графических фигур и элементов GUI. Графические методы Java доступны в классе `Graphics`, их можно использовать с помощью вызовов в переопределенных методах `paint` и `repaint`. Для отображения графических элементов применяются фреймы — объекты `Frame`, которые являются подклассами абстрактного класса `Component`. При разработке GUI можно использовать компоненты из широкого диапазона: объекты классов `Button`, `ScrollBar`, `Textfield` и т.д. Пакет `Swing` является альтернативой пакету `awt` и предоставляет платформно-независимые элементы и функции GUI.

В этой главе не раскрывались понятия, связанные с ключевыми словами, но широко использовались API пакета `java.awt`. Рассматривались такие пакеты, классы и методы:

```
java.awt                Frame
java.awt.event            setTitle
javax.swing               setSize
ActionListener            setVisible
  actionPerformed       Graphics
BorderLayout              drawOval
Button                    drawRect
  addActionListener       drawString
Canvas                    fillOval
Component                 fillRect
  paint                   Color
  repaint                 setColor
  setBackground         Label
  setForeground         LayoutManager
  setLocation           Panel
Container               System.exit
  add                   Window
  setLayout               addWindowListener
FlowLayout              WindowAdapter
FontMetrics               windowClosing
                        WindowEvent
```

Следующий перечень включает темы, на которых в главе сделано особое ударение:

- создание окна (объекта класса, наследующего классу `Frame`);
- переопределение метода `paint`;
- спецификация класса `Graphics`;
- закрытие окна;
- создание объекта `Label`;
- создание объекта `Button`;
- подключение менеджера размещения;
- реакция на нажатие кнопки.

Вопросы



10.1. Чтобы скомпилировать программу графического пользовательского интерфейса, необходимо:

а) наследовать классу `Window`

б) наследовать классу `Frame`

в) использовать метод `setVisible(true)`

г) вызвать метод `repaint`

10.2. Круг, создаваемый посредством метода `drawOval(300, 20, 50, 50)` во фрейме размером 600×400, будет позиционирован:

а) в нижней части фрейма слева

б) в верхней части фрейма слева

в) в верхней части фрейма по центру

г) в центре

10.3. Установить шрифт `SansSerif` с полужирным начертанием позволяет оператор:

а) `setFont (sanserif, bold);`

б) `g.setFont(Font.sanSerif, Font.Bold, size);`

в) `g.setFont(new Font (Font.SansSerif, Font.BOLD, size));`

г) `g.setFont(new Font ("SansSerif", Font.BOLD, size));`

10.4. Для GUI-приложения, созданного в Java, менеджером размещения по умолчанию является:

а) `FlowLayout`

б) `BorderLayout`

в) `Flow centred`

г) менеджер по умолчанию не назначается

10.5. При выполнении GUI-программы нажатие некоторой кнопки ни к чему не привело. Причина, возможно, заключается в том, что:

а) не создан экземпляр кнопки

б) кнопка не связана со слушателем

в) нет обработчика для события, источником возникновения которого является кнопка

г) переменная кнопки присвоена другой кнопке

10.6. `ActionListener` — это:

а) метод, вызываемый программой

б) интерфейс, который программа должна реализовать

в) класс, наследуемый программой

г) метод, который вызывает программу при наступлении события

10.7. Ниже приведен альтернативный вариант двух операторов из примера 10.3:

```
  p.add (new Button ("Перезагрузить"));
add("South", p);
```

Какое высказывание является верным?

а) панель `p` должна быть указана перед методом `add` или передана как параметр

б) первый метод `add` относится к панели, а второй — к фрейму

в) если мы записываем второй метод `add` с отступом, то должны включить в него панель `p`

г) для размещения панели `p` используется менеджер `BorderLayout`

10.8. Какой из операторов предназначен для обращения к менеджеру `BorderLayout`?

а) `setLayout (BorderLayout);`

б) `setLayout (new BorderLayout);`

в) `setLayout (new BorderLayout());`

г) `setLayout (new BorderLayout(), CENTER);`

10.9. Сколько фреймов создается на заключительном этапе веполнения программы `FlagMaker2` из примера 10.3?

а) один

б) три

в) четыре

г) ни одного

10.10. Наиболее значительное различие между пакетами `awt` и `Swing` заключается в том, что:

а) методы `Swing` начинаются с `j`

б) методы `Swing` ориентируются на вид и функции конкретной платформы

в) методы `Swing` более эффективны

г) методы `Swing` реализуются независимо от какой-либо конкретной платформы

Задачи

Примечание. Первые три задачи связаны с разработкой и программированием графического пользовательского интерфейса. В них не обязательно организовывать обработку событий — для проверки GUI достаточно, чтобы обработчик выдал сообщение о том, что произошло определенное событие. Такое сообщение может быть сделано посредством метода `System.out.println` и выведено в консольном окне Java или в командном окне MS-DOS.

10.1. **Электронные часы.** Разработайте эмулятор электронных часов с цифровой индикацией. Сымитируйте работу часов. Для изменения значения времени должен использоваться цикл.

10.2. **Пульт управления.** Реализуйте пользовательский интерфейс, имитирующий пульт управления телевизором.

10.3. **Сотовый телефон.** Разработайте экранную версию сотового телефона.

10.4. **Дверной замок.** Отдел безопасности Саванны занимается изготовлением дверных цифровых замков с десятью кнопками, на которых указаны цифры от 0 до 9, и с двумя кнопками: **Индикатор** и **Открыть**. При нажатии корректной последовательности из четырех цифр и кнопки **Открыть** загорается зеленая лампочка. В противном случае загорается красная лампочка. Создайте GUI-программу, обеспечивающую работу такого замка. Протестируйте ее, введя заранее в программу пять комбинаций цифр, которые пользователь должен выбрать, чтобы загорелся зеленый свет.

10.5. **Система безопасности.** После того как выпуск замков был налажен, потребовалось специальное программное обеспечение. Напишите небольшую программу, в которой обрабатываемыми данными служат номера квартир, имена людей и соответствующие коды. Обеспечьте в программе возможность назначения каждому замку корректной последовательности цифр на те случаи, если будут меняться замки, а следовательно, и коды. Новый код должен вводиться с клавиатуры. При изменении кодов используйте пароль.

10.6. **Усовершенствованные замки.** Усовершенствованные замки должны быть снабжены тремя дополнительными кнопками: **X**, **Y** и **Z**. Создайте фрейм, который наследовал бы фрейму, описанному в GUI-программе из задачи 10.4, и введите дополнительную информацию. Как программа из задачи 10.5 будет работать с обоими видами замков после незначительных изменений?

10.7. **Светофоры.** Усовершенствуйте программу из примера 10.5 таким образом, чтобы получить на экране два идентичных изображения светофора. Рассмотрите различные способы их представления: в виде двух различных фреймов и в виде двух холстов в одном фрейме.

10.8. **Улучшенная программа `FlagMaker`.** Программа `FlagMaker2` из примера 10.3 будет обновлена в главе 11, а сейчас можно произвести некоторые интересные усовершенствования. Замените три фиксированных тестовых объекта в конструкторе GUI собственной разработки, позволяющим вводить четыре параметра (название страны и три цвета) в интерактивном режиме.

ГЛАВА 11



Управление событиями

Очевидно, что реагирование на события представляет собой важную составляющую программирования GUI. В языке Java пакет `awt.event` предоставляет средства для прослушивания событий, записи информации о событиях и соединения с помощью ссылок, определенных пользователем, с обработчиками событий. В этой главе мы рассмотрим:

- методы и приемы классификации событий, а также назначения соответствующих слушателей и обработчиков;
- принципы работы с последовательностями событий.

Чтобы подготовить читателя к восприятию излагаемого материала, мы, рассматривая компоненты пользовательского интерфейса, уделим особое внимание текстовым полям как одной из основных составляющих процесса взаимодействия с пользователем.

11.1. Взаимодействие с текстовыми полями

Для ввода текста в GUI-программу Java предоставляет компонент `TextComponent` с двумя подклассами: `TextField` и `TextArea`. Класс `TextArea` предназначается для хранения текста, состоящего из множества строк, и "работает" в союзе с полосами прокрутки. Здесь мы рассмотрим текстовые поля, состоящие из одной строки. Для объявления такого компонента, как `TextField`, воспользуемся описанием, приведенным ниже.

Создание текстового поля

```
TextField t;

t = new TextField("initial text",n);
  add(t);
```

С помощью этого объявления создается объектная переменная для текстового поля `t`.
Создается экземпляр класса с первоначальной строкой текста и заданной шириной символа, от чего зависит непосредственный размер текстового поля (см. ниже).
Метод `add` внедряет текстовое поле в текущий контейнер.

Переменная `n` определяет необходимое количество вводимых в поле символов, но не задает точно его ширину, поскольку на ширину поля оказывают влияние выбранный шрифт и используемый менеджер размещения. Однако если пользователь вводит больше чем `n` символов, то в поле добавляется полоса прокрутки, поэтому

указание некорректного значения n не приведет к возникновению проблем. В следующем описании перечислены некоторые методы, доступные для объекта TextField:

Методы класса TextField

```
String  getText ();
void    setText ("value");
void    selectText (start, end);
void    selectAll ()
void    setEditable (booleanvalue);
boolean isEditable ();
```

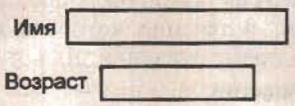
Метод getText получает любой текст, который был введен с клавиатуры в текстовое поле, и возвращает его как значение типизированного метода.
Метод setText заменяет весь текст, имеющийся в поле, данной строкой.
Метод selectText выделяет часть текстового поля, ничего не изменяя;
Метод selectAll выделяет весь текст.
Текстовое поле может быть доступным для изменения или недоступным, и это состояние может быть назначено и проверено методами getText и setText.

Методы getText и setText являются эквивалентами методов чтения и записи для класса TextField. Каждое поле может быть открыто для редактирования (что устанавливается по умолчанию), а может быть закрыто. В первом состоянии оно белого цвета, и пользователь, переместив в него курсор мыши, выполняет щелчок, вводит текст, после чего нажимает клавишу [Enter]. Будучи не доступным для редактирования, поле приобретает серый цвет, и ввод данных не допускается (цвет поля зависит от используемого компьютера). Метод selectText позволяет выделить заданную часть поля, (как правило, этот участок становится голубого цвета).

Текстовые поля обычно имеют ассоциированные метки. Для помещения текстовых полей, предназначающихся для ввода имени и возраста, в одну панель мы будем использовать код

```
Panel p = new Panel ();
  TextField name = new TextField("",40);
    p.add (new Label ("Имя"));
    p.add (name);
  TextField age = new TextField("",5);
    p.add (new Label ("Возраст"));
    p.add (age);
```

что даст на выходе:



Потоковый менеджер размещения FlowLayout аккуратно разместит прямоугольники текстовых полей во фрейме заданного размера. Если фрейм окажется больше, метка **Возраст** может быть помещена в верхний ряд, поэтому для достижения нужного результата может понадобиться дополнительная настройка.

Ввод чисел

Вы, наверное, заметили, что класс `TextField` имеет только один метод — `get`, возвращающий объект `String`. Следовательно, мы вновь сталкиваемся с преобразованием строки в число. Об этом процессе говорилось в разделе 7.2, где для целых чисел использовался следующий ключевой оператор:

Транслирование строки в целое число
`i = Integer.parseInt(s);`
Транслирует строку `"s"` в целое число `"i"`. Если строка имеет некорректную форму, система выдаст сообщение об исключительной ситуации `ParseException`.

Точно такая же технология может быть использована в программировании GUI. Однако простой оператор, наподобие описанного, предполагает, что при преобразовании все будет происходить надлежащим образом, т.е. не встретится такой случай, как, скажем, не числовой символ в строке. Этими проблемами занимается класс `Stream`, и можно предположить, что должен быть написан аналогичный класс (например, `GUIText`), который будет выполнять те же функции, но работать с данными, вводимыми из класса `TextField`, а не класса `BufferedReader`.

Компоненты, позволяющие работать без ввода данных

В конце раздела 10.3 мы упоминали о том, что в `awt` есть несколько компонентов, предоставляющих возможность выбирать нужную кнопку выбора или нужное значение из раскрывающихся списков. Этим экономится время пользователя, а также устраняется возможность некорректного ввода. Так, при необходимости указать день недели пользователю вовсе не обязательно вводить информацию в текстовое поле — достаточно щелкнуть мышью на нужной строке в списке `Choice`. Компоненты `CheckBox`, `List`, `Menu` и `Popup` используются аналогичным образом. Пример использования объекта `Choice` приведен в усовершенствованной версии программы "Светофор" (Пример 11.1).

Пример 11.1. Светофор с кнопками

Задача. Программа "Светофор", которая создавалась как симулятор, предназначена для исследования возможности изменения длительности горения каждого света светофора. Например, должен ли красный свет гореть дольше, чем зеленый.

Решение. Чтобы программа-симулятор была проста в использовании, нам нужна возможность изменять время, в течение которого данный свет горит. Другими словами, мы должны вводить новые значения. Для этого требуется:

- выбор одного из четырех цветов;
- ввод новых значений длительности.

Необходимые компоненты могут быть добавлены к панели кнопок и помещены в нижней части экрана. Это будет раскрывающийся список, в котором представлены четыре цвета, а также текстовое поле для ввода значений длительности (в секундах).

Программа. У нас еще недостаточно знаний, чтобы составить программу, в которой свет будет загораться — это будет рассматрено в главе 13. Однако мы можем реагировать на выбор света и установление длительности его горения, отправляя программе для обработки сообщение, в котором указан выбранный свет и новая длительность. Вот как выглядит конструктор для фрейма, содержащего "светофор":

```
Traffic1() {
  setTitle("Светофор Версия 1");
  panel title = new Panel();
    title.add(new Label("Симулятор светофора"));
    add("North", title);

  lights = new LightsCanvas();
  add("Center", lights);

  Panel buttons = new Panel();
    colours = new Choice ();
    colours.addItem("Красный");
    colours.addItem("Желтый");
    colours.addItem("Зеленый");
    colours.addItem("Идите");
    buttons.add(colours);

  duration = new TextField("", 3);
    buttons.add(new Label("Длительность"));
    duration.setEditable(true);
    buttons.add(duration);

  walkButton = new Button("Идите");
    buttons.add(walkButton);

  closeButton = new Button("Закрыть");
    closeButton.addActionListener(this);
    buttons.add(closeButton);
  add("South", buttons);

  private TextField duration;
  private Choice colours;
  private Button walkButton;
  private Button closeButton;
}
```

Тестирование. Результат работы конструктора показан на рис. 11.5.

Несмотря на то, что в этом конструкторе определены компоненты для выбора цвета и длительности, он не указывает, каким образом должны обрабатываться события, относящиеся к этим компонентам. Как обычно, для кнопки **Закрыть** устанавливается ссылка на обработчик, но теперь мы должны рассмотреть способы распространения обработки событий на другие виды компонентов и учесть более сложные последовательности событий.

11.2. События слушатели и обработчики событий

Классификация событий связана с типами компонентов, генерирующими события, но для каждого типа компонента может существовать несколько типов событий. Скорее, можно сказать, что существует 11 типов **событий (events)**, которые совместно используются многими типами **компонентов (components)**. У каждого типа события есть ассоциированный с ним **слушатель (listener)** (класс `MouseEvent` имеет два слушателя), и для каждого слушателя соответствующий **обработчик (handler)** должен реализовывать один или более его методов.

Например, компонент `Button` ассоциирован с классом `ActionEvent` и интерфейсом `ActionListener`, при этом программистом должен быть реализован метод `actionPerformed`. В табл. 11.1 приводится полный список имеющихся в Java событий, слушателей, методов и компонентов. Все слушатели являются интерфейсами, поэтому те из них, к которым обращается программа, должны быть перечислены в заголовке класса.

Чтобы понять табл. 11.1, рассмотрим пример из предыдущей программы. В ней определен список, и выбор какого-либо пункта из списка должен вызывать соответствующие действия. Класс `Choice` — это компонент, который генерирует событие `ItemEvent` и которому нужен слушатель `ItemListener`, поэтому реализуется метод `itemStateChanged`. Как выглядит такая реализация? Обратимся к обработчику, который уже рассматривался при определении метода `actionPerformed` в примере 10.4:

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == rebootButton) {
    setVisible(false);
    dispose();
    System.exit (0);
  }else if (e.getSource() == waitButton) {
    setForeground(Color.white);
    setBackground(Color.red);
  }
}
```

Следовательно, для возвращения объекта, сгенерировавшего событие, класс `ActionEvent` использует метод `getSource`. Эту ссылку на объект можно успешно опросить и выяснить, какое событие произошло для того, чтобы выполнить соответствующие действия. Метод `getSource` определен в классе `EventObject`, который расположен очень высоко в иерархии — он находится в пакете `java.util`, а не в `java.awt`! В каждом из классов-событий есть соответствующий специфический метод, который может быть использован вместо `getSource`. Проблема состоит в том, что нужно помнить все имена и возвращаемые значения, поскольку они не следуют одинаковой модели, как это делают слушатели и другие методы, указанные в табл. 11.1. Например, для класса `ActionEvent` методом, альтернативным методу `getSource`, является `getActionCommand`, возвращающий строку. Следовательно, вместо

```
if (e.getSource() == rebootButton) {
```

можно применить

```
if (e.getActionCommand() == "Перезагрузить") {
```

Однако для события `ItemEvent`, которое генерируется компонентом `Choice`, используемым методом будет `getItemSelectable`, возвращаеющий значение, базирующееся на интерфейсе `ItemSelectable`. Для упрощения программы мы применяем метод `getSource` как средство идентификации событий.

Таблица 11.1. Классификация событий, слушателей, методов и компонентов

Событие	Слушатель	Метод	Компонент

`ActionEvent`	`ActionListener`	`actionPerformed`	`Button`
			`List`
			`MenuItem`
			`TextField`

`AdjustmentEvent`	`AdjustmentListener`	`adjustmentValueChanged`	`ScrollBar`

`ComponentEvent`	`ComponentListener`	`componentHidden`
		`componentMoved`
		`componentResized`
		`componentShown`
			`Component`

`ContainerEvent`	`ContainerListener`	`componentAdded` `componentRemoved`	`Container`
`FocusEvent`	`FocusListener`	`focusGained` `focusLost`	`Component`

`ItemEvent`	`ItemListener`	`itemStateChanged`	`CheckBox`
			`Choice`
			`List`

`KeyEvent`	`KeyListener`	`keyPressed`
		`keyReleased`
		`keyTyped`
			`Component`

`MouseEvent`	`MouseListener`	`mouseClicked`
		`mouseEntered`
		`mouseExited`
		`mousePressed`
		`m ouseReleased`
			`Component`

	`MouseMotionListener`	`mouseDragged` `mouseMoved`	
`TextEvent`	`TextListener`	`textValueChanged`	`TextComponent`

`WindowEvent`	`WindowListener`	`windowActivated`
		`windowClosed`
		`windowClosing`
			`Window`

11.3. Практикум 5. Разработчик флагов

Задача

В примерах 2.2 и 10.3 мы видели, что простые флаги начертить не сложно, и цвета их полос могут быть введены как параметры. Теперь рассмотрим возможность введения параметров, которые позволяли бы работать с другими файлами, скажем, чертить флаги с вертикальными полосами (например, флаг Франции), а также обеспечить GUI-интерфейс для создания абсолютно нового дизайна флага.

Решение

Решение делится на две части:

- определение типов флагов, черчение которых можно легко выполнить;
- предоставление GUI-интерфейса, позволяющего выбрать компоненты для создания таких флагов.

Мы начнем с трех типов дизайна флагов: с горизонтальными полосами, с вертикальными и с горизонтальными полосами и изображением (`motif`) в центре. На рис. 11.1 приведены два последних типа дизайна. У нас есть четыре кнопки для выбора цвета каждой полосы флага (четвертая предназначена для выбора цвета изображения в центре флага, если таковое предусмотрено в дизайне), само изображение и название флага (или страны). Варианты собственно изображения могут быть следующими: отсутствие изображения, круг, звезда и звезда Давида. Все эти компоненты позволяют создать простой GUI-интерфейс (такой, как на рис. 11.2), используя который можно разработать флаг довольно сложного дизайна (второй флаг на рис. 11.1).

Для назначения цвета мы используем кнопку выбора элемента, а затем указываем в раскрывающемся списке цвет. Это более эффективно, чем использование четырех раскрывающихся списков — по одному для каждой полосы и изображения. Список для выбора цвета будет относиться к любому элементу, выбранному на текущий момент с помощью кнопки выбора.

Процесс разработки программы состоит из трех частей:

- конструирование окна;
- реагирование на события;
- отображение флага.

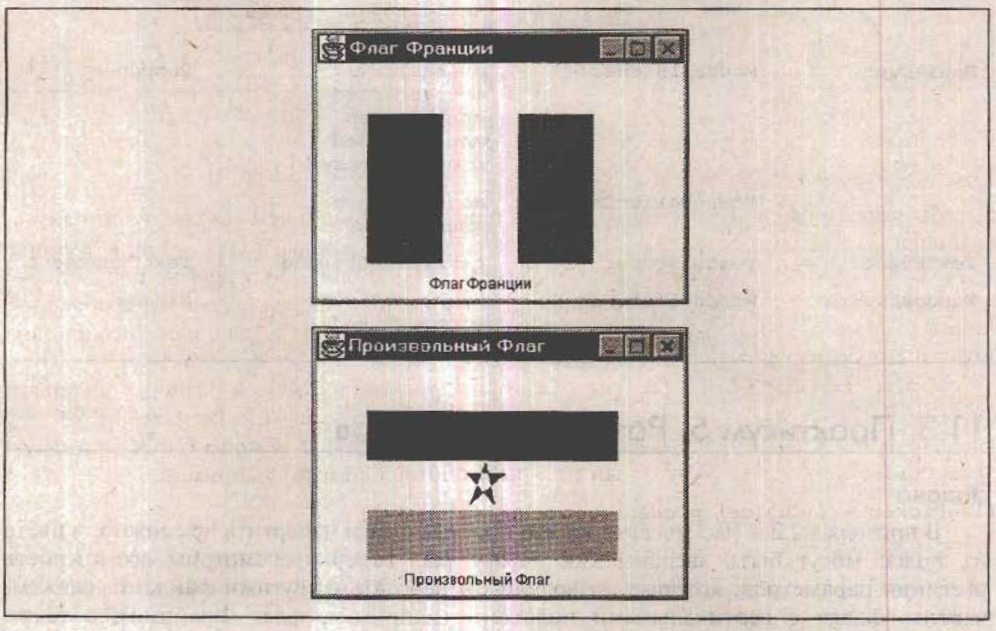


Рис. 11.1. Два типа флагов

Разработка GUI

Это довольно простой GUI, включающий в себя метки и восемь других компонентов. Они идентифицированы необходимыми нам GUI-переменными:

```
CheckboxGroup    flagParts;
Checkbox []      colourCheck = new Checkbox[4];
Choice           colourChoice;
Choice           flagChoice;
Choice           motifChoice;
TextField        nameField;
Button           drawButton;
```

Компонент `CheckboxGroup` и массив `Checkbox` вместе взятые предоставляют четыре кнопки выбора, расположенные в верхей части GUI-окна. Затем следуют три списка, текстовое поле и кнопка **Начертить**. Добавление этих элементов в GUI — довольно длительное по времени занятие, в результате которого получается такой же длинный конструктор, но другого пути пока не существует. Возможности работы с параметрами будут рассмотрены далее.

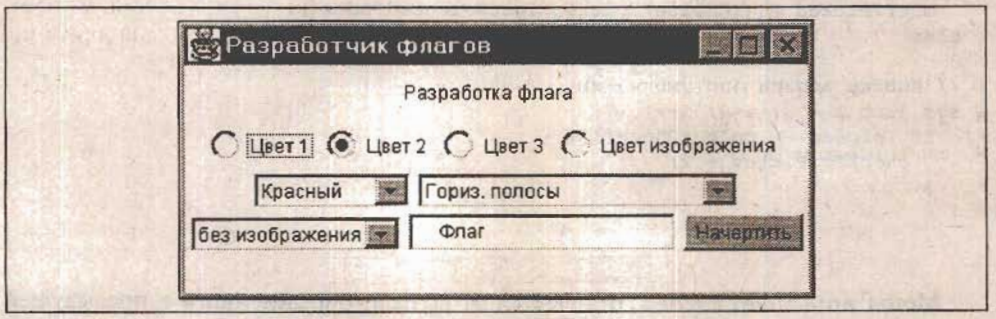


Рис. 11.2. GUI для программы "Разработчик флагов"

Реагирование на события

По мере добавления компонентов во фрейм мы соединяем их со слушателями. Из табл. 11.1 видно, что и для раскрывающихся списков, и для кнопок выбора необходимым является слушатель `ItemListener`, а для двух других событий — слушатель `ActionListener`. Это означает, что мы должны реализовать два обработчика, `itemStateChanged` и `actionPerformed`, и различать для каждого из них компонент, который стал источником события.

Метод для обработки раскрывающихся списков и кнопок выбора довольно сложный. В нем имеется четыре блока — последовательно идущие операторы `if`, каждый из которых обрабатывает различный компонент. В каждом блоке мы получаем объект `Choice` — выбранный раскрывающийся список, например:

```
flagPicked = ((Choice) picked).getSelectedIndex();
```

К объекту `Choice` применяется метод `getSelectedIndex`. Объект `picked` возвращается методом `getSource`, и поскольку он определен как экземпляр суперкласса `Object`, то должен быть приведен к типу `Choice`. При выборе цвета фиксируется результат для конкретной полосы или изображения, причем по умолчанию устанавливается вторая полоса. То есть если мы выберем цвет раньше, чем выберем элемент, для которого этот цвет предназначен, это будет предусмотренной,

нормальной ситуацией. В завершение проверяется, какая из кнопок выбора была выбрана, после чего в переменной `selectedColour` сохраняется соответствующее порядковому номеру целое число.

```
public void ItemStateChanged(ItemEvent e) {
  Object picked = e.getSource();

  // Различие между четырьмя компонентами.
  // Выбор цвета.
  if (picked == colourChoice) {
    int selectedColour = ((Choice) picked).getSelectedIndex();
    colour[checkBoxPicked] = listedColours[selectedColour];
  }

  // Выбор (дизайна) флага.
  else if (picked == flagChoice)
    flagPicked = ((Choice) picked).getSelectedIndex();

  // Выбор изображения.
  else if (picked == motifChoice)
    motifPicked = ((Choice) picked).getSelectedIndex();
  else

  // Кнопка выбора (по умолчанию).
  for (int i = 0; i<4; i++) {
    if (picked == colourCheck[i]) {
      checkBoxPicked = i;
    }
  }

}
```

Метод `actionPerformed` отличается от метода, определенного в предыдущей программе, только тем, что в процессе выполнения реагирует на нажатие кнопки **Начертить**. Теперь перейдем третьему этапу программы.

Отображение флагов

У нас есть класс для черчения флага, состоящего из трех горизонтальных полос. Метод, который создает фрейм, и его детали выглядят следующим образом:

```
public static void makeFlag (FlagCanvas canvas) {
  Frame f = new Frame();
  f.add(canvas);
  f.setTitle(canvas.country);
  f.setSize(300,200);
  f.setLocation(200,200);
  f.setVisible(true);
  f.addWindowListener(new WindowAdapter() {
  public void windowClosing(WindowEvent e) {
  e.getWindow().dispose();
    }
  });
}
```

Черчение флага с вертикальными полосами тоже не является проблемой. Но чтобы не усложнять метод `makeFlag`, определим все три варианта дизайна как объекты, наследующие класс `FlagCanvas`, которые будем передавать этому методу в качестве параметров. Если в коде имеются строки

```
class FlagCanvas extends Canvas
class FlagCanvasVerti extends FlagCanvas
class FlagCanvasMotif extends FlagCanvas
```

то любой объект любого из этих трех классов будет доступен методу `makeFlag`.

Единственным еще не применявшимся нюансом фактического процесса черчения является процесс создания звезд в изображении. Для их вычерчивания мы будем использовать предварительно инициализированные массивы точек, которые можно передать методу `fillPolygon` или `drawPolygon`, определенным в классе `Graphics`.

Программа

Теперь, после рассмотрения деталей, мы приводим полный текст программы.

```
import java.awt.*;
import java.awt.event.*;

public class FlagMakerGUI extends Frame
          implements ActionListener, ItemListener {
   /* Программа "Разработчик флагов"  Автор - Дж.М. Бишоп, май 2000 г.
    * = = = = = = = = = = = = = = = = =
    *
    * Представляет простой GUI для выбора одного из трех
    * стилей флага, плюс три изображения и семь цветов
    * для каждой вертикальной или горизонтальной полосы.
    *
    * Демонстрирует импользование наследования и компонентов GUI, в частности
    * раскрывающихся списков и кнопок выбора.
    */

  static final int none = 0, circle = 1, star = 2, starOfDavid = 3;
  static final int stripes = 0, bars = 1, motif = 2;
  static final Color listedColours [] = {Color.red, Color.blue,
          Color.white, Color.black, Color.green,
          Color.orange, Color.yellow};

  // Переменные GUI.
  // **************

  CheckboxGroup  flagParts;
  Checkbox []    colourCheck = new Checkbox[4];
  Choice         colourChoice;
  Choice         flagChoice;
  Choice         motifChoice;
  TextField      nameField;
  Button         drawButton;

  public FlagMakerGUI() {

  // Размещение элементов GUI.
  // ************************
    setLayout (new BorderLayout());
```

```
    Panel intro = new Panel();
      intro.add(new Label("Разработка флага"));
      add("North", intro);
    Panel middle = new Panel();
      middle.setLayout(new FlowLayout());
      flagParts = new CheckboxGroup();
        for (int i=0; i<3; i++)
          colourCheck[i] = new  Checkbox("Цвет "+(i+1),i==1,
              flagParts);
        colourCheck[3] = new  Checkbox("Цвет изображения", false,
              flagParts);
        for (int i=0; i<4; i++) {
          colourCheck[i].addItemListener(this);
          middle.add(colourCheck[i]);
        }
      colourChoice = new Choice ();
        colourChoice.addItem("Красный");
        colourChoice.addItem("Голубой");
        colourChoice.addItem("Белый");
        colourChoice.addItem("Черный");
        colourChoice.addItem("Зеленый");
        colourChoice.addItem("Оранжевый");
        colourChoice.addItem("Желтый");
        colourChoice.addItemListener(this);
        middle.add(colourChoice);
      flagChoice = new Choice ();
        flagChoice.addItem("Гориз. полосы");
        flagChoice.addItem("Верт. полосы");
        flagChoice.addItem("Гор. полосы с изображением");
        flagChoice.addItemListener(this);
        middle.add(flagChoice);
      motifChoice = new Choice();
        motifChoice.addItem("без изображения");
        motifChoice.addItem("Круг");
        motifChoice.addItem("Звезда");
        motifChoice.addItem("Звезда Давида");
        motifChoice.addItemListener(this);
        middle.add(motifChoice);
      nameField = new TextField("     Флаг ",20);
        nameField.addActionListener(this);
        middle.add(nameField);
      drawButton = new Button("Начертить");
        drawButton.addActionListener(this);
        middle.add(drawButton);
    add("Center",middle);
    setTitle("Разработчик флагов");
    setVisible(true);
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
      System.exit(0);
      }
    });
  }

  // Переменные для обработчиков событий.
  int      flagPicked = 0;
  int      checkBoxPicked = 0; // цвет № 1
```

```
    int       motifPicked = circle;
    Color [] colour = {Color.red, Color.white, Color.blue, Color.black};
    String     name = " Флаг ";

    public void actionPerformed (ActionEvent e) {
      if (e.getSource() == drawButton) {
        switch (flagPicked) {
          case stripes : makeFlag (
               new FlagCanvas (name, colour[0], colour[1], colour[2]));
               break;
          case bars : makeFlag (
               new FlagCanvasVerti (name, colour[0], colour[1],
               colour[2]));
               break;
          case motif : makeFlag (
               new FlagCanvasMotif (name, colour[0], colour[1],
               colour[2],
               motifPicked, colour[3]));
        }}
      else if (e.getSource() == nameField)
        name = nameField.getText();
    }

    public void itemStateChanged(ItemEvent e) {
      Object picked = e.getSource();

      // Нахождение компонента - источника события.
      // Выбор цвета.
      if (picked == colourChoice) {
        int selectedColour = ((Choice) picked).getSelectedIndex();
        colour[checkBoxPicked] = listedColours[selectedColour];
      }
      // Выбор флага.
      else if (picked == flagChoice)
        flagPicked = ((Choice) picked).getSelectedIndex();
      // Выбор изображения.
      else if (picked == motifChoice)
        motifPicked = ((Choice) picked).getSelectedIndex();
      else
      // Кнопки выбора (по умолчанию).
      for (int i=0; i<4; i++) {
        if (picked == colourCheck[i]) {
          checkBoxPicked = i;
        }
      }
    }

    public static void main (String [] args) {
      new FlagMakerGUI();
  }

    public static void makeFlag (FlagCanvas canvas) {
      Frame f = new Frame();
      f.add(canvas);
      f.setTitle(canvas.country);
      f.setSize(300,220);
      f.setLocation(200,200);
```

```
    f.setVisible(true);
    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        e.getWindow().dispose();
      }
    });
  }
}

// = = = = = = = = = = = = = = = = = =
class FlagCanvas extends Canvas {

String country;
  Color bar1, bar2, bar3;

  FlagCanvas (String c, Color b1, Color b2, Color b3) {
    country = c;
    bar1 = b1;
    bar2 = b2;
    bar3 = b3;
  }

  public void paint (Graphics g) {
    // Чертит флаг из цветных прямоугольников.
    g.setColor (bar1);
    g.fillRect (40, 40, 200, 40);
    g.setColor (bar2);
    g.fillRect (40, 80, 200, 40);
    g.setColor (bar3);
    g.fillRect (40, 120, 200, 40);
    // Вывод названия рисунка (страны).
    g.setColor (Color.black);
    g.drawString(country,70,180);
  }
}

class FlagCanvasVerti extends FlagCanvas {
  FlagCanvasVerti (String c, Color b1, Color b2, Color b3) {
    super (c,b1,b2,b3);
  }

  public void paint (Graphics g) {
    // Чертит флаг из цветных прямоугольников.
    g.setColor (bar1);
    g.fillRect (40, 40, 60, 120);
    g.setColor (bar2);
    g.fillRect (100, 40, 60, 120);
    g.setColor (bar3);
    g.fillRect (160, 40, 60, 120);
    // Вывод названия рисунка (страны).
    g.setColor (Color.black);
    g.drawString(country,90,180);
  }
}
class FlagCanvasMotif extends FlagCanvas {

  int motif;
```

```
Color motifColor;

FlagCanvasMotif (String c, Color b1, Color b2, Color b3, int m,
        Color mc){
  super (c,b1,b2,b3);
  motif = m;
  motifColor = mc;
}

public void paint (Graphics g) {

  int [] xpoints = {120,150,122,135,142,120};
  int [] ypoints = {90,90,115,80,115,90};
  // Чертит флаг из цветных прямоугольников.
  g.setColor (bar1);
  g.fillRect (40, 40, 200, 40);
  g.setColor (bar2);
  g.fillRect (40, 80, 200, 40);
  g.setColor (bar3);
  g.fillRect (40, 120, 200, 40);
  // Вывод названия рисунка (страны)..
  g.setColor (Color.black);
  g.drawString(country,70,180);
  if (motif != 0) {
    g.setColor (motifColor);
    switch (motif) {
    case 1: g.fillOval(125,90,20,20); break;
    case 2: g.fillPolygon(xpoints,ypoints,6); break;
    case 3: g.drawPolygon(xpoints,ypoints,6); break;
    default : break;
    }
  }
 }
}
```

Тестирование

Тестирование программы занимает довольно много времени, поскольку пользователь должен проверить все возможные комбинации, все взаимодействия одного компонента с другим. (Результат работы программы показан на рис. 11.1).

11.4. Работа с последовательностями событий

Событиями являются единичные изменения состояния. Взятые вместе, они могут формировать цепочку событий. Другими словами, конкретное событие не всегда приводит к одному ответному действию. Это могут быть два или три ответных действия, в зависимости от событий, происшедших ранее.

Как же мы фиксируем последовательность событий? В большинстве случаев используются названные подходящим образом булевы переменные, которые указывают, произошло или нет предыдущее событие. Так, в примере 11.1, где рассмотрена панель, представляющая возможность выбора цвета, а затем ввода новой длительности включенного состояния, мы должны обеспечить выполнение этих двух событий именно в таком порядке. Мы можем также не допустить активизации (изменения на экране) этих компонентов до тех пор, пока не будет нажата кнопка **Изменить**.

Соблюдайте главное условие: какой бы алгоритм ни был выбран, вы должны четко представлять себе последовательность событий во избежание зацикливания программы или возникновения непредусмотренных ситуаций. Кроме того, программа должна иметь понятный пользователю интерфейс.

Пример 11.2. Настройка светофора

Задача. Мы продолжим работу с интерфейсом из примера 10.5 и добавим возможность изменения интервала времени, когда горит определенный свет светофора.

Решение. В примере 10.5 мы изменили конструктор и вывели на экран все необходимые компоненты. Теперь нужно соединить со слушателями те из них, которые участвуют в диалоге, и сконструировать обработчики событий для их активизации. Это три различных компонента — раскрывающийся список для выбора цвета, текстовое поле для назначения длительности "горения" света и кнопка **Идите**. В настоящий момент мы ничего не можем сделать для активизации указанной кнопки, поэтому с ней работать не будем.

Чтобы сымитировать изменение длительности горения определенного света в ответ на введенное значение, мы просто выведем это значение рядом с соответствующим цветом. Это означает, что каждый раз, когда вводится новое значение длительности, центральный рисунок должен быть перечерчен. Следовательно, метод `actionPerformed` отвечает за вызов метода `repaint`.

Алгоритм. Алгоритм для новой программы приведен на рис. 11.3.

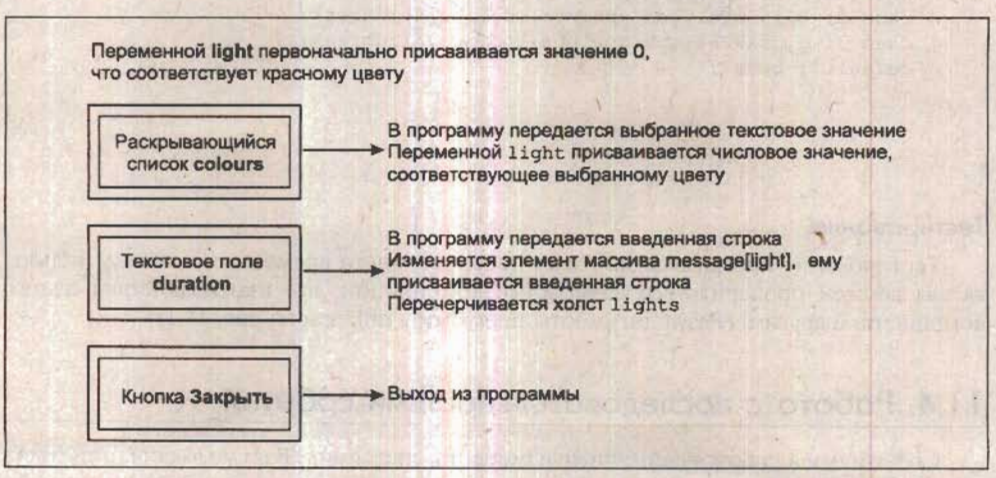


Рис. 11.3. Обработчики событий для программы "Настройка светофора", версия 1

Создадим схему усовершенствованной программы.

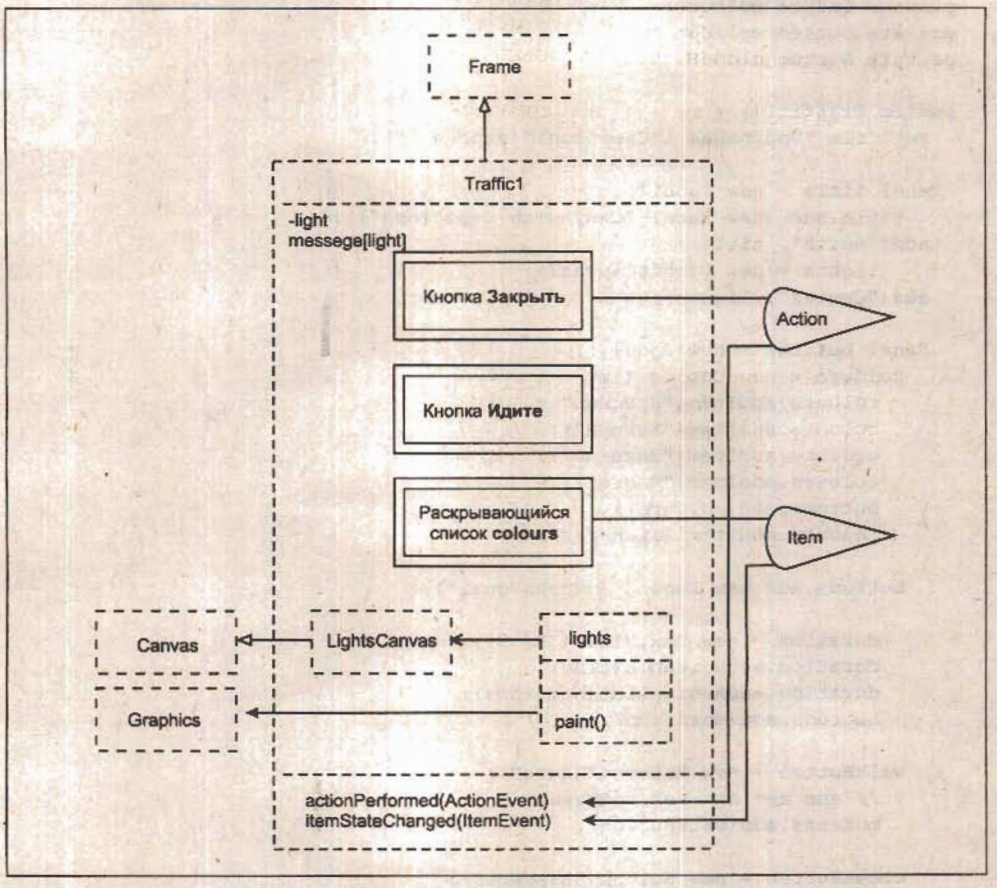


Рис. 11.4. Схема для программы "Настройка светофора"

Программа

```
import java.awt.*;
import java.awt.event.*;

public class Traffic1 extends Frame
        implements ActionListener, ItemListener {
   /* Вторая программа для симулятора светофора -
    *  "Настройка светофора"          Автор Дж.М. Бишоп, октябрь 1996 г.
    *                          Java 1.1 версия Т.Абботта, октябрь 1997 г.
    *                расширена и исправлена  Дж.М. Бишоп, октябрь 1997 г.
    *                                обновлена Дж.М. Бишоп, август 2000 г.
    *
    * Добавляет опцию, позволяющую назначать длительность включенного
    * состояния для выбранного цвета, но на данном этапе это значение
    * длительности просто выводится на экран.
    */
```

```
private LightsCanvas lights;
private TextField duration;
private Choice colours;
private Button walkButton;
private Button closeButton;

public Traffic1() {
  setTitle("Программа \"Светофор\" версия 1");

 Panel title = new Panel();
    title.add (new Label("Симулятор светофора"));
  add("North", title);
      lights = new LightsCanvas();
  add("Center", lights);

  Panel buttons = new Panel();
    colours = new Choice ();
      colours.addItem("Красный");
      colours.addItem("Желтый");
      colours.addItem("Зеленый");
      colours.addItem("Идите");
      buttons.add(colours);
      colours.addItemListener(this);

    buttons.add(new Label("Длительность"));

      duration = new TextField("", 3);
      duration.setEditable(true);
      duration.addActionListener(this);
      buttons.add(duration);

    walkButton = new Button("Идите");
      // еще нет никаких действий
      buttons.add(walkButton);

    closeButton = new Button("Закрыть");
      closeButton.addActionListener(this);
      buttons.add(closeButton);
  add("South", buttons);

  // Создание фрейма (окна).
  setSize(350, 210);
  setVisible(true);
  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
      System.exit(0);
    }
  });
}

public void actionPerformed(ActionEvent e) {
  if (e.getSource() == closeButton) {
  setVisible(false);
  dispose();
```

```
      System.exit (0);
    } else if (e.getSource() == duration) {
      message[light] = duration.getText();
      lights.repaint();
    }
  }

  public void itemStateChanged(ItemEvent e) {
    if (e.getItemSelectable() == colours) {
      String s = (String) e.getItem();
      if (s == "Красный") {light = 0;} else
      if (s == "Желтый") {light = 1;} else
      if (s == "Зеленый") {light = 2;} else
      if (s == "Идите") {light = 3;}
    }
  }

  public static void main (String [] args) {
    new Traffic1();
  }

  private int light = 0;
  String [ ] message = {"по умолчанию "," по умолчанию "," по умолчанию ",
                        " по умолчанию "};

  class LightsCanvas extends Canvas {
    public void paint (Graphics g) {
    g.drawOval(87, 10, 30, 68);
    g.setColor(Color.red);
    g.fillOval(95, 15, 15, 15);
    g.setColor(Color.yellow);
    g.fillOval(95, 35, 15, 15);
    g.setColor(Color.green);
    g.fillOval(95, 55, 15, 15);
      // Цвет кнопки "Идите" также зеленый.
    g.fillOval(95, 85, 15, 15);
    g.setColor(Color.black);
    g.drawString("Красный", 15, 28);
    g.drawString("Желтый", 15, 48);
    g.drawString("Зеленый", 15, 68);
    g.drawString("Идите", 15, 98);
    g.drawString(message[0], 135, 28);
    g.drawString(message[1], 135, 48);
    g.drawString(message[2], 135, 68);
    g.setColor(Color.black);
    g.drawString(message[3], 135, 98);
    }
  }
}
```

Тестирование. Результат работы программы показан на рис. 11.5. Здесь два значения длительности изменены, а два оставлены в качестве используемых по умолчанию.

Разработка симулятора светофора преднамеренно не доведена до конца, поскольку еще не ясно, как будет отсчитываться длительность горения определенного света, прежде чем он будет выключен, и как эти включения затем должны меняться. Для создания действительно полноценной модели светофора нам понадобятся средства Java, позволяющие работать с несколькими подпроцессами (*multithreading*). Подпроцессы рассматриваются в главе 13, где мы вновь вернемся к этому примеру.



Рис. 11.5. Результат работы программы "Настройка светофора"

Давайте рассмотрим другую программу из примера 6.8 — "Конвертер валют". В ней мы использовали для операций ввода/вывода интерфейс класса `Display`, но программа имела ограниченные возможности. Теперь мы можем создать более профессиональную программу, взяв на вооружение адаптированный GUI.

Пример 11.3. Конвертер валют с GUI

Задача. Программа по конвертированию валюты, разработанная в примере 6.8, должна выйти за рамки класса `Display`. Мы хотим создать более удачную версию GUI и расширить программу таким образом, чтобы она могла выполнять конвертирование любых валют.

Компоновка элементов GUI. Разработка любой GUI-программы начинается с вывода на экран ее компонентов. Мы представили свой вариант внешнего вида GUI-программы на рис. 11.7. Из рисунка видно, что область для выводимого текста (объект `TextArea`) находится в центре фрейма, а по обоим его краям расположены раскрывающиеся списки. В верхней части текстового окна находится текстовое поле **Сумма**, а в нижней — кнопка **Конвертировать**, которая активизирует процесс конвертирования и вывод данных.

Для создания соответствующей программы мы должны распределить компоненты интерфейса по панелям и решить, какой менеджер размещения использовать для каждой из них. Предлагаемая разбивка панелей на области показана на рис. 11.6. Общим менеджером размещения (для фрейма) является `BorderLayout`, поэтому три главные панели можно расположить в западной (`West`), центральной (`Center`) и восточной (`East`) областях. Южная и северная области фрейма не используются, но *могут содержать заголовок или инструкции.*

Западная и восточная панели базируются на менеджере `BorderLayout` и имеют метку в северной области, а также раскрывающийся список в центральной. Центральная панель аналогична, но ее северная область представляет собой еще одну панель, включающую в себя метку и текстовое поле для суммы.

Конструкция программы. Отказавшись от использования класса `Display`, мы придадим структуре программы новую форму. В соответствии с определением класса `Display`, мы были привязаны к операциям ввода, нажатия кнопки **Готово** и вывода полученных данных. Но поскольку мы обладаем полным контролем над GUI, то можем реагировать на события в любое время. Кроме того, ситуация, когда имеются две выбранные страны, еще не означает, что должна происходить транзакция. Нам нужно организовать обработку последовательности событий таким образом, чтобы выполнение транзакции было позволено только в том случае, если будет выбран какой-то элемент из каждого раскрывающегося списка.

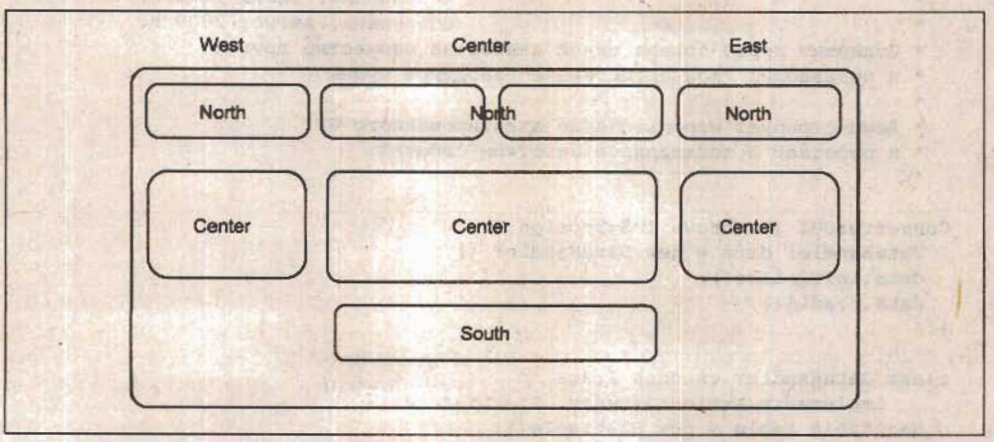


Рис. 11.6. Размещение компонентов по областям фрейма в GUI-программе "Конвертер валют"

Логично предположить, что нужно ввести две булевы переменные, по одной для каждого раскрывающегося списка. Они обе вначале инициализируются как `false`, но после выбора из раскрывающегося списка названия страны булевой переменной присваивается значение `true`. Например:

```
public void itemStateChanged(ItemEvent e) {
  String s = (String) e.getItem();
  if (e.getItemSelectable() == fromChoice) {
    fromSelected = true;
    fromCountry = s;
  }
  else {
    toSelected = true;
    toCountry = s;
  }
}
```

Булевы переменные проверяются при попытке конвертирования (после нажатия кнопки **Конвертировать**), и в случае успешного выполнения им присваивается значение `false`.

Главная программа в основном инициализирует GUI и считывает из файла данные с курсами валют. В программе отсутствует бесконечный цикл, вызывающий метод, который выполняет транзакции. Метод, выполняющий конвертирование, вызывается методом `actionPerformed` после нажатия кнопки **Конвертировать**.

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import javagently.*;

class ConverterGUI extends Frame {

   /*     Программа              Автор Дж. М. Бишоп, декабрь 1998 г.
    * "Конвертер валют"      версия с классом Display, июль 1999 г.
    *                                  GUI-версия,  июль 1999 г.
    *                                  обновлена, август 2000 г.
    * Содержит курсы обмена одной валюты на множество других
    * и производит "конвертирование" вводимой суммы.
    *
    * Демонстрирует использование адаптированного GUI
    * и работает с последовательностями событий.
    */

  ConverterGUI () throws IOException {
    DataHandler data = new DataHandler ();
    data.initialize();
    data.readIn();
  }

  class DataHandler extends Frame
       implements ActionListener, ItemListener {
    Hashtable table = new Hashtable();

    // Вводит каждую строку данных и сохраняет их в
    // хеш-таблице с названием страны в качестве ключа.

    Choice fromChoice, toChoice;
    boolean fromSelected, toSelected;
    TextField amountField;
    TextArea resultField;
    String toCountry, fromCountry;
    Button goButton;

    void initialize () {


        // Панель в левой области фрейма.
        Panel q = new Panel();
          q.add ( new Label ("Из"));
          fromChoice = new Choice();
              fromChoice.addItemListener(this);
            q.add(fromChoice);
          add ("West",q);
        // Панель в правой области фрейма.
        q = new Panel ();
          q.add ( new Label ("В"));
```

```
        toChoice = new Choice();
          toChoice.addItemListener(this);
          q.add(toChoice);
        add ("East",q);
      // Центральная панель.
      q = new Panel (new BorderLayout());
        Panel r = new Panel();
          r.add(new Label("Сумма"));
          amountField = new TextField("1000 ");
            amountField.addActionListener(this);
            r.add(amountField);
            q.add("North",r);
          resultField = new TextArea(8,20);
            q.add("Center",resultField);
            resultField.append("Вначале выберите страны\n");
          goButton = new Button("Конвертировать");
            goButton.addActionListener(this);
            q.add("South",goButton);
       add("Center",q);

    setTitle("Конвертер валют");
    setSize(610,300);
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });
}

public void actionPerformed(ActionEvent e) {
  int amount = 1000;
  if (e.getSource() == amountField)
    amount = (int) Integer.parseInt(amountField.getText());
  else
  if (e.getSource() == goButton) {
    if (fromSelected && toSelected)
      transaction(amount);
    else
      resultField.append("Вначале выберите страны\n");
    }
}

public void itemStateChanged(ItemEvent e) {
  String s = (String) e.getItem();
  if (e.getItemSelectable() == fromChoice) {
    fromSelected = true;
    fromCountry = s;
  }
  else {
    toSelected = true;
    toCountry = s;
  }
}

void transaction (int amount) {
  Rates fromRate = (Rates) table.get(fromCountry);
  Rates toRate = (Rates) table.get(toCountry);
```

```
    resultField.append(amount+" "+fromRate.country+ " "+
        fromRate.currency+
        "\n в "+toRate.country+" "+toRate.currency+
        "\n равны "+
         Stream.format(amount/fromRate.conversion*
            toRate.conversion,10,3)+"\n\n");
    fromSelected = false;
    toSelected = false;
  }

  void readIn() throws IOException {
    Stream fin = new Stream("Rates.dat", Stream.READ);
    Rates rate;
    try {
      for (int i = 0;  ; i++) {
        rate = new Rates();
        rate.setRate(fin);
        table.put(rate.country, rate);
        toChoice.addItem(rate.country);
        fromChoice.addItem(rate.country);
      }
    }
    catch (EOFException e) {}
    fromSelected = false;
    toSelected = false;
    setVisible(true);
    }
  }

  public static void main (String [] args) throws IOException {
    new ConverterGUI();
  }

}
```

Тестирование. Результат нескольких выполнений GUI-программы "Конвертер валют" показан на рис. 11.7. Заметьте, что программа идентифицирует некорректные попытки конвертирования.

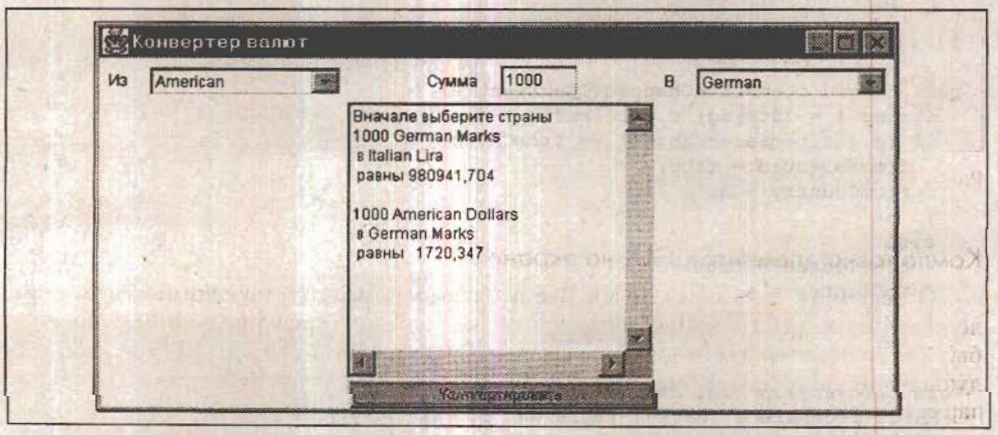


Рис. 11.7. Графический пользовательский интерфейс для программы "Конвертер валют"

11.5. Практикум 6. Касса супермаркета

Для решения следующей задачи будут использованы различные компоненты GUI, а также обработка событий и последовательности событий.

Задача

Управление торговли Саванны решило создать программу для вычисления стоимости заданного веса продукции. Это должна быть программа, которая выводит на монитор кнопки различного назначения и текстовое поле для ввода веса продукции. На текущий момент принимается, что весы не связаны непосредственно с кассой, и специальное устройство для печати чеков отсутствует, а стоимость покупки и название каждой единицы продукции вводится в консольном окне Java.

Решение

Последовательность событий показана на рис. 11.8. Если клубника весит 3 кг, мы вводим в текстовое поле **Вес** цифру 3. Стоимость одного килограмма каждой единицы продукции дается в грацах, скажем, для земляники G12 за 1 кг. Общая стоимость указанного количества продукции, которая в данном случае равна G36, выводится в текстовом поле **Стоимость**, а в завершение будет напечатана строка с информацией о произведенной транзакции.

Решение задачи состоит из двух частей:

- компоновка элементов GUI на экране
- организация обработки событий.



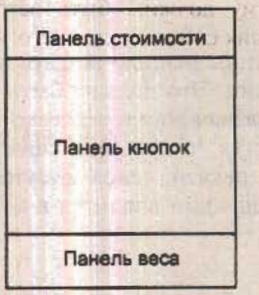
Рис. 11.8. Схема работы кассы

Компоновка элементов GUI на экране

Отдельно от текстовых полей **Вес** и **Стоимость** с сопутствующими им метками должны находиться кнопки, соответствующие каждому виду продукции. Они могут быть расположены рядами и колонками, как кнопки калькулятора. Для этого случая лучшим менеджером размещения будет не потоковый, а `GridLayout`. В качестве параметров для этого менеджера мы укажем предполагаемое количество рядов и колонок, и Java будет выводить компоненты в соответствующие области по мере их

добавления. В результате получится более фиксированное и симметричное расположение, чем при реализации `FlowLayout`.

В любом случае мы будем иметь на экране три панели, которые в `BorderLayout` будут размещены следующим образом:



Панель кнопок содержит по одной кнопке для каждого вида продукции плюс три другие, предназначенные для управления кассой: **Взвесить**, **Распечатать** и **Закрыть**. Особой необходимости в кнопке **Закрыть** нет, поскольку программа прекращает свою работу после закрытия окна, но ею можно воспользоваться, например, для выведения на печать общего текущего объема продажи.

Обработка событий

Мы начнем с перечисления возможных событий.

Нажать кнопку Взвесить
Нажать кнопку какого-нибудь вида продукции (например, Горох)
Ввести вес
Нажать кнопку Распечатать
Нажать кнопку Закрыть
Закрыть окно

Однако не во всех случаях эти события могут быть инициированы. Нам нужно, чтобы при взвешивании продукции события происходили в установленном порядке, согласно которому системе должны быть переданы все необходимые данные, прежде чем нажатие кнопки **Распечатать** приведет к требуемому действию.

В любом порядке должны быть выполнены три пункта:	**Нажать кнопку Взвесить**
	Нажать кнопку любого вида продукции
	Щелкнуть мышью в поле Вес, ввести значение веса продукции, нажать клавишу [Enter]
Затем:	**Нажать кнопку Распечатать**

Теперь опишем действия, выполняемые при каждом событии.

Нажатие кнопки Взвесить	**Открывает текстовое поле Вес**
Нажатие кнопки с названием продукции	**Передает в программу значение стоимости продукции и изменяет значение булевой переменной `chosen` (можно перевести как "выбор сделан")**
Ввод текста в поле Вес	**Передает в программу значение веса продукции и изменяет значение булевой переменной `weighed` (можно перевести как "вес указан")**

Нажатие кнопки **Распечатать**	Выводит на экран или на печать значение стоимости продукции указанного веса и присваивает булевым переменным значения `false`

Однако эти действия не будут выполнены при нарушении последовательности событий. Поэтому в программу должны быть встроены операторы, проверяющие последовательность происшедших событий, для чего можно использовать пару булевых переменных (рис. 11.9).

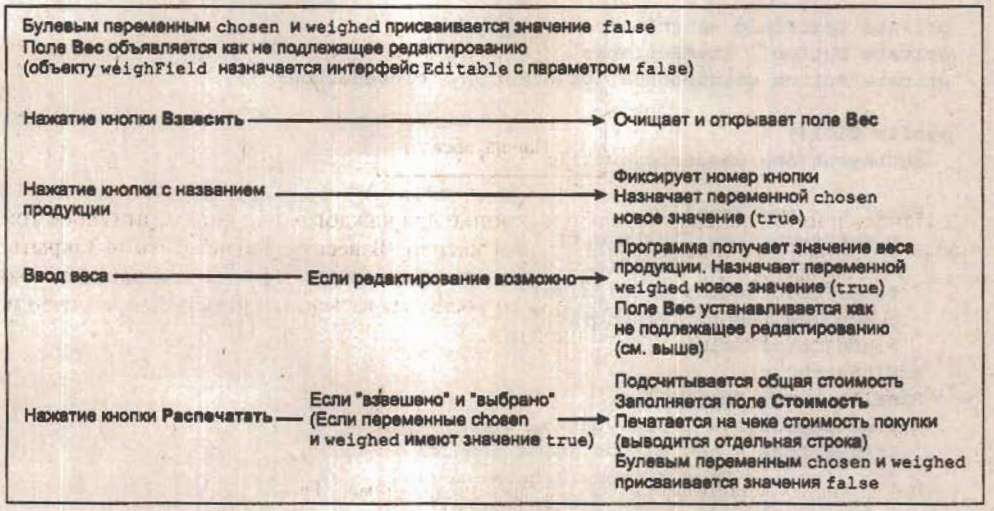


Рис. 11.9. Обработка событий в программе "Касса супермаркета"

Ввод данных

Остается вопрос, как внедрить в программу базовые данные, т.е. названия каждой единицы продукции и ее стоимость (за 1 кг). Поскольку ассортимент продукции, а тем более ее цена, могут каждый день изменяться, имеет смысл сохранять эту информацию в файле. Но для упрощения программы мы изначально внедрим в нее несколько элементов (названия продукции и цены). Читатели могут закрепить приобретенные навыки, расширив диапазон возможностей программы до работы с файлом.

Программа

Далее следует текст программы. Заметьте, что в конструкторе мы объявляем одну панель `p`, но создаем три ее экземпляра — для каждой из трех различных, нужных нам панелей.

```
import java.awt.*;
import java.awt.event.*;

public class Till extends Frame implements ActionListener {

    /* Программа "Касса супермаркета" Автор Дж. М. Бишоп,  октябрь 1996 г.
     * - - - - - - - - -   Java 1.1 Т.Абботт и Дж.М. Бишоп,  октябрь 1997 г.
     *                                          обновлена, август 2000 г.
```

```
   *
   * Имитирует работу кассы овощного магазина, торгующего продукцией 12-ти
   * видов по определенной цене.
   *
   * Демонстрирует использование панелей, различных менеджеров
   * размещения (включая GridLayout), пользовательских обработчиков
   * событий и непосредственно процесс обработки событий, которые
   * происходят в определенном порядке.
   */

private TextField weighField, totalField;
private Button[] itemButtons;
private Button weighButton, printButton, closeButton;

public Till() {
  setLayout(new BorderLayout());

  Panel p = new Panel();
    p.setLayout(new FlowLayout());
    totalField = new TextField(6);
    totalField.setEditable(false);
    p.add(new Label("Стоимость"));
    p.add(totalField);
  add("North",p);
  Panel q = new Panel();
    q.setLayout(new GridLayout(5, 3));
    itemButtons = new Button[items.length];
    for (int i = 0; i < items.length; i++) {
      itemButtons[i] = new Button(items[i]);
      itemButtons[i].addActionListener(this);
      q.add(itemButtons[i]);
    }
    weighButton = new Button("Взвесить");
    weighButton.addActionListener(this);
      q.add(weighButton);
    printButton = new Button("Распечатать");
    printButton.addActionListener(this);
      q.add(printButton);
    closeButton = new Button("Закрыть");
    closeButton.addActionListener(this);
      q.add(closeButton);
  add("Center", q);

  p = new Panel();
    p.setLayout(new BorderLayout());
   Panel  g = new Panel();
      weighField = new TextField(4);
      weighField.setEditable(false);
      weighField.addActionListener(this);
      g.add(weighField);
      g.add(new Label("Вес"));
  p.add("Center",g);
  p.add("South",new Label("После ввода веса нажмите клавишу \"Enter\""));
```

```
    add("South", p);

    // Вывод фрейма.
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });
    setTitle("Супермаркеты Саванны");
    setVisible(true);
    pack();
  }

  public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source == closeButton) {
      System.exit(0);
    } else
    if (source == printButton) {
      printItems();
    } else
    if (source == weighButton) {
      resetWeighField();
    } else
    if (source == weighField) {
      readWeighField();
    } else
      selectItem(e.getActionCommand());
    }

   /* Далее следует код, предписывающий основные действия,
    * выполняемые программой.
    * Обработчики событий и действий, указанные выше, вызывают
    * методы, которые гарантируют, что ответные действия будут
    * выполнены только при соблюдении определенных условий.
    */

  public void resetWeighField() {
    weighField.setText("");
    weighField.setEditable(true);
  }

  public void readWeighField() {
    if (weighField.isEditable()) {
      weighField.setEditable(false);
      weighField.selectAll();
      kg = (double)Integer.valueOf(
      weighField.getText().trim()).intValue();
        weighed = true;
    }
  }
```

```
  public void printItems() {
    if (weighed && chosen) {
      total = kg*unitCosts[select];
      totalField.setText("G "+total);
      System.out.println(kg+"кг "+items[select]+" @ G"+
         unitCosts[select]+" = G"+total);
      kg = 0;
      weighed = false;
      chosen = false;
      total = 0;
      weighField.select(0,0);
    }
  }

  public void selectItem(String item) {
    select = 0;
    while (!item.equals(items[select]))
      select++;
    chosen = true;
  }

  public static void main (String [] args) {
    new Till();
  }

  private String[] items = {"Яблоки", "Груши", "Апельсины",
          "Картофель", "Лимоны", "Арбузы", "Лук", "Чеснок",
          "Авокадо", "", "", ""};
  private double[] unitCosts = {6.00, 5.00, 7.00, 3.00, 10.00, 4.00, 4.00,
                      12.00, 15.00, 0, 0, 0};
  private double total;
  private double kg;
  private boolean chosen = false;
  private boolean weighed = false;
  private int select =1;

}
```

Метод `actionPerformed` содержит четыре оператора `if`, соответствующих пяти событиям (событие закрытия окна обрабатывается в конструкторе отдельно). Три события — нажатие кнопок **Закрыть**, **Взвесить** и кнопки с названием продукции — не зависят от выполнения каких-нибудь условий. Эти события всегда приведут к запрограммированным для них действиям. Остальные два требуют выполнения предварительных событий. Например, в текстовое поле `weighField` можно ввести данные только тогда, когда оно доступно для редактирования, а нажатие кнопки **Распечатать** приведет к соответствующим действиям лишь в том случае, если переменным `chosen` и `weighed` было присвоено значение `true`.

Обратите внимание на использование в конце конструктора нового метода `pack`. В отличие от метода `setSize`, в котором нужно точно указать ширину и высоту фрейма в пикселях, метод `pack` "ждет", пока не будут позиционированы все компоненты, а затем подбирает необходимый размер фрейма.

Тестирование

На рис. 11.10 изображено окно программы после выбора кнопки **Лук**. Вы можете выполнить несколько проверочных прогонов программы. В консольном окне Java будет выведена на печать следующая информация:

```
12.0кг Лук @ G4.0 = G48.0
```



Рис. 11.10. Интерфейс программы "Касса супермаркета"

11.6. Практикум 7. Программа "Конвертер валют", использующая пакет Swing

Теперь, после того как мы ознакомились с большей частью классов пакета `awt`, можно приступить к рассмотрению пакета `Swing`. Как говорилось в разделе 10.1, пакет `Swing`, являющийся платформо-независимым, представляет собой альтернативу пакету `awt`. Программированию в `Swing` посвящены целые отдельные книги, но на данном этапе вам достаточно знать основные принципы работы этого пакета. К наиболее важным особенностям `Swing` относятся:

- большее число доступных компонентов, отдельные из которых незначительно отличаются от соответствующих компонентов пакета `awt` (как правило, компоненты `Swing` идентифицируются с помощью префикса `J`);
- способность изменить вид и функции всего GUI одним оператором;
- расширенные средства для обработки текста на экране, в первую очередь текста, написанного на языке разметки гипертекста HTML (Hypertext Markup Language);
- внутреннее представление структуры данных, отличное от ее представления в GUI.

Последнее означает, что пакет `Swing` содержит средства, позволяющие GUI представлять данные последовательно и в единообразной форме (структуре), например в виде адресной книги, но при этом действительная структура представления данных остается для GUI неизвестной. Но этой темой мы в данной книге заниматься не будем, а сосредоточимся на первых двух пунктах. Чтобы продемонстрировать, как используется `Swing`, мы перепишем программу "Конвертер валюты" (Пример 11.3).

Компоненты Swing

Перечислим основные компоненты Swing:

```
JButton          JProgressBar
JCheckBox        JRadioButton
JColorChooser    JScrollBar
JComboBox        JScrollPane
JComponent       JSlider
JDesktopPane     JSplitPane
JDialog          JTabbedPane
JEditorPane      JTable
JFileChooser     JTextArea
JFrame           JTextField
JLabel           JTextPane
JLayeredPane     JToggleButton
JList            JToolBar
JMenu            JToolTip
JOptionPane      JTree
JPanel           JViewport
JPopupMenu       JWindow
```

Многие из них, в частности `JButton` и `JLabel`, имеют аналогичные классы в `awt`. Другие, такие как `JSlider` и `JColorChoser`, представляют собой новые классы. При преобразовании программы "Конвертер валюты" мы будем использовать новый компонент `JComboBox`.

Класс `JComboBox` в основном заменяет компонент `Choice` пакета `awt` и представляет более общий способ вывода в окно GUI объектов, находящихся в списках. В комбинированном списке (*combo box*) весь объект сохраняется как элемент списка, а методом `toString` считываются только значения, которые выводятся на экран. Мы хотим, чтобы в программе "Конвертер валюты" названия стран выводились на экран в двух раскрывающихся списках. Вместо добавления названий стран как элементов и последующего хранения отдельной структуры данных, содержащей все данные о конвертировании валюты (т.е. курсы обмена для валюты каждой страны), сохраняем весь объект `Rates` в комбинированном списке. Определяем метод `toString` так, чтобы на экран выводилось название страны, а затем, если потребуется, можно было бы извлечь остальную информацию.

Обновление класса Rates

Поскольку к методу `toString` имеется определенное требование, очевидно, что мы должны обновить класс `Rates`. Для этого существует два пути: просто создать другой класс с таким же именем, но содержащий метод `toString`, или объявить новый класс, наследующий класс `Rates` и имеющий добавленный метод `toString`. Теоретически второй способ является лучшим решением. Но расширяющемуся классу должно быть присвоено новое имя. Например в программе "Ветеринарные бирки" (Практикум 4) это мог бы быть класс `XRates` либо даже `SRates` или `JRates`. В таком случае класс `Rates` во всех строках программы, где он упоминается, должен быть заменен новым классом.

Однако только ради определения такого стандартного метода, как `toString`, эта "революция" не нужна. Если новая версия класса `Rates` будет использоваться в одной из старых программ по конвертации, она не создаст никаких проблем. По этой причине мы решили оставить старое имя `Rates`, и теперь новая версия класса будет выглядеть так:

```
import java.io.*;
import javagently.*;

class Rates {
   /* Класс Rates      Автор Дж. М. Бишоп, декабрь 1998 г.
    * - - - - - -      обновление, автор Б. Воролл, август 2000 г.
    * Содержит название страны, название валюты, ее код и курс.
    */

  String country;
  String code;
  String currency;
  double conversion;

  void setRate (Stream in) throws IOException {
    code = in.readString();
    country = in.readString();
    currency = in.readString();
    // Первый курс не нужен, поэтому мы его
    // игнорируем, вводя новое значение.
    conversion = in.readDouble();
    conversion = in.readDouble();
  }

  public String toString () {
    return country;
  }

}
```

Метод `toString` возвращает только название страны — значение, которое будет использоваться в комбинированном списке.

Структура данных

В ранних версиях программы по конвертированию название страны (как ключ) и объекты `Rates` (как значения) хранились в хеш-таблице. Поскольку теперь весь объект хранится в комбинированном списке и выбор делается непосредственно щелчком мыши на названии нужной страны, необходимость в отдельной структуре данных отпадает. Комбинированный список и *есть* структура данных.

Все части программы, отвечающие за ввод данных и выполнение транзакции, остаются почти идентичными. Объявление двух раскрывающихся списков будет выглядеть следующим образом:

```
JComboBox fromChoice, toChoice;
```

Цикл помещения объектов в списки будет таким:

```
for (int i = 0; ; i++) {
  rate = new Rates();
```

```
    rate.setRate(fin);
    toChoice.addItem(rate);
    fromChoice.addItem(rate);
  }
```

а так будут выглядеть операторы, предназначенные для извлечения данных о курсе обмена:

```
void transaction () {
  // Курсы "в" и "из" загружаются непосредственно из списка
  // JComboBox и используются для вывода результатов работы программы.
  Rates fromRate = (Rates) fromChoice.getSelectedItem();
  Rates toRate = (Rates) toChoice.getSelectedItem();
  resultField.append(amount+" "+fromRate.country+ " "+
  fromRate.currency+"\n в "+toRate.country+" "+
  toRate.currency+"\n будут равняться "+
  Stream.format(amount/fromRate.conversion*
  toRate.conversion,10,3)+"\n\n");
}
```

Вид и функции

Поскольку программы Java являются переносимыми и могут выполняться на любом компьютере, в пакете `Swing` предусмотрены средства, с помощью которых вид и функции GUI могут или зависеть от компьютера, на котором программа выполняется, или быть абсолютно от него не зависимыми. В первом случае GUI будет знаком пользователю данного компьютера, но на другом компьютере он примет иной вид. Во втором случае один и тот же GUI на разных компьютерах будет выглядеть одинаково. Эти два варианта можно получить при вызове соответствующего метода со следующими параметрами:

```
UIManager.setLookAndFeel
     (UIManager.getSystemLookAndFeelClassName());

 UIManager.setLookAndFeel
     (UIManager.getCrossPlatformLookAndFeelClassName ());
```

В дополнение к этим операторам могут быть выбраны специфические виды и функции, идентифицированные классами, которые их реализуют. Например:

```
"javax.swing.plaf.metal.MetalLookAndFeel"
"javax.swing.plaf.mac.MacLookAndFeel"
"com.sun.java.swing.plaf.windows.Win dowsLookAndFeel"
"com.sun.java.swing.plaf.motif.MotifLookAndFeel"
```

Поскольку вид и функции, свойственные платформе Macintosh, будут иметь место только на компьютере Macintosh, при их назначении в программе должна присутствовать опция `default`. Примеры этих GUI приведены далее, при тестировании программы.

Пиктограммы

Принципы работы с изображениями рассматриваются в разделе 12.3. В пакете `Swing` они более интегрированы, чем в `awt`. Объект изображения создается, когда объекту в качестве параметра передается имя файла, файл считывается и изображение (если в файле было изображение) выводится на экран. Объект `JLabel` позволяет

размещать изображение рядом с меткой, поэтому мы используем его в приведенной ниже программе для создания заголовка GUI-программы по конвертированию валют. Применяемые при этом операторы имеют следующий вид:

```
JLabel label = new JLabel ("Обмен валют в Саванне",
       new ImageIcon("Elephant.gif"),CENTER);
  p.add("North", label);
```

Программа

Теперь мы можем привести весть текст программы. Благодаря использованию комбинированного списка она короче предыдущей своей версии. Эта программа вводит параметр из командной строки для указания вида и функций GUI с помощью чисел из диапазона от 1 до 6. Ниже приводятся названия возможных вариантов внешнего вида GUI.

1. `CrossPlatform` (назначает заданные по умолчанию собственные для Java вид и функции — этот пункт аналогичен пункту 3).
2. `System` (выбирает один из системных видов GUI, т.е. 4, 5 или 6).
3. `Metal` (вид и функции, присущие Java).
4. `Windows` (свойственен Microsoft).
5. `Motif` (свойственен Sun и Unix).
6. `Macintosh` (свойственен Apple).

Логика программы не совсем такая, как в примере 11.6. Это выражается в том, что опущена обработка событий. В программе гораздо сложнее обнаружить изменения, вызванные использованием комбинированных списков.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import javagently.*;

class ConverterSwing extends JFrame {

   /*      Программа          Автор Дж.М. Бишоп, декабрь 1998 г.
    * "Конвертер валюты"      версия с классом Display, июль 1999 г.
    *                         GUI-версия, июль 1999 г.
    *                         Swing-версия Б. Воролл
    *                         и Дж.М. Бишоп, Август 2000 г.
    * Содержит курсы обмена одной валюты на ряд других
    * и производит "конвертирование" вводимой суммы.
    *
    * Демонстрирует использование настроенного Swing-GUI.
    */


  public static void main(String[] args) throws IOException {
    int look = 0;
    if (args.length > 0)
      look = Integer.parseInt(args[0]);
    new ConverterSwing(look);
  }
  ConverterSwing (int look) throws IOException{
```

```
    try {
      switch (look) {
        case 1: UIManager.setLookAndFeel
          (UIManager.getCrossPlatformLookAndFeelClassName()); break;
        case 2: UIManager.setLookAndFeel
          (UIManager.getSystemLookAndFeelClassName()); break;
        case 3: UIManager.setLookAndFeel
          ("javax.swing.plaf.metal.MetalLookAndFeel"); break;
        case 4: UIManager.setLookAndFeel
          ("com.sun.java.swing.plaf.windows.WindowsLookA ndFeel");
          break;
        case 5: UIManager.setLookAndFeel
          ("com.sun.java.swing.plaf.motif.MotifLookAndFeel"); break;
        case 6: UIManager.setLookAndFeel
          ("javax.swing.plaf.mac.MacLookAndFeel"); break;
      }
    }
    catch (Exception c) {
      System.out.println("Could not use the specified look and feel"
        + "- defaulting to cross-platform");
    }

      DataHandler data = new DataHandler ();
      data.initialise();
      data.readIn();
    // Теперь контроль передается пользователю, а события
    // обрабатываются с помощью
    // методов actionPerformed и transaction()
  }

  class DataHandler extends JFrame
                  implements ActionListener, SwingConstants {

    JComboBox  fromChoice, toChoice;
    JTextField amountField;
    JTextArea  resultField;
    JButton    goButton;

    void initialise () {
      JPanel p = new JPanel (new BorderLayout());
      // Заголовок
      JLabel label = new JLabel ("Обмен валют в Саванне",
        new ImageIcon("Elephant.gif"),CENTER);
        p.add("North", label);
        // Левая панель.
        JPanel q = new JPanel();
          q.add ("North",new JLabel ("Из"));
          fromChoice = new JComboBox();
            q.add("Center",fromChoice);
          p.add ("West", q);
        // Правая панель
        q = new JPanel();
          q.add ("North",new JLabel ("В"));
          toChoice = new JComboBox();
            q.add("Center",toChoice);
          p.add ("East",q);
        // Центральная панель
```

```
    q = new JPanel(new BorderLayout());
      JPanel r = new JPanel();
        r.add(new JLabel("Сумма"));
        amountField = new JTextField("1000  ");
          amountField.addActionListener(this);
          r.add(amountField);
          q.add("North",r);
        resultField = new JTextArea(8,20);
          q.add ("Center",resultField);
        goButton = new JButton ("Конвертировать");
          goButton.addActionListener(this);
          q.add("South",goButton);
      p.add("Center",q);
  getContentPane().add(p);
  setTitle("Конвертер валют");
  setSize(610,300);

  setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}

public void actionPerformed (ActionEvent e) {
  if (e.getSource() == goButton)
    transaction();
}

void transaction () {
  // Вначале считываем последнее значение переменной amount
  int amount = (int)
      Integer.parseInt(amountField.getText().trim());
  // Курсы обмена "из" и "в" загружаются непосредственно из
  // списка JComboBox и используются для формирования
  // результатов работы программы.
  Rates fromRate = (Rates) fromChoice.getSelectedItem();
  Rates toRate   = (Rates) toChoice.getSelectedItem();
  resultField.append(amount+" "+fromRate.country+" "+
                     fromRate.currency+
                     "\n в "+toRate.country+" "+
                     toRate.currency+"\n составляют "+
  Stream.format(amount/fromRate.conversion*
                toRate.conversion,10,3)+"\n\n");
}

void readIn() throws IOException {
  Stream fin = new Stream ("Rates.data", Stream.READ);
  Rates rate;
  try {
    for (int i = 0; ; i++) {
      rate = new Rates();
      rate.setRate(fin);
    // Здесь мы сохраняем данные о курсах обмена в комбинированных
    // списках JComboBox. Заметьте, что сохраняется весь объект
    // Rate, но только элемент, указанный в методе toString
    // класса Rates, будет показан в поле списка.
      toChoice.addItem(rate);
      fromChoice.addItem(rate);
    }
   }
```

```
            catch (EOFException e) {}
            setVisible(true);
        }
    }
}
```

Тестирование

Программа не нуждается в тестировании, но результаты применения различных видов и функций GUI желательно просмотреть. Мы приводим три копии экрана (рис. 11.11 — 11.13), которые демонстрируют:

- свойственный Java межплатформный вид, называемый `Metal` (`CrossPlatfor` и `Metal`);
- вид, присущий данной системе, который совпадает с видом, полученным при прогоне программы на платформе `Windows` (`System` или `Windows`);
- вид, который называется `Motif` (`Macintosh`).

Как мы упоминали ранее, `Swing` — это коллекция пакетов, которые могут предложить еще многое для программирования GUI. Но это тема для отдельной книги.

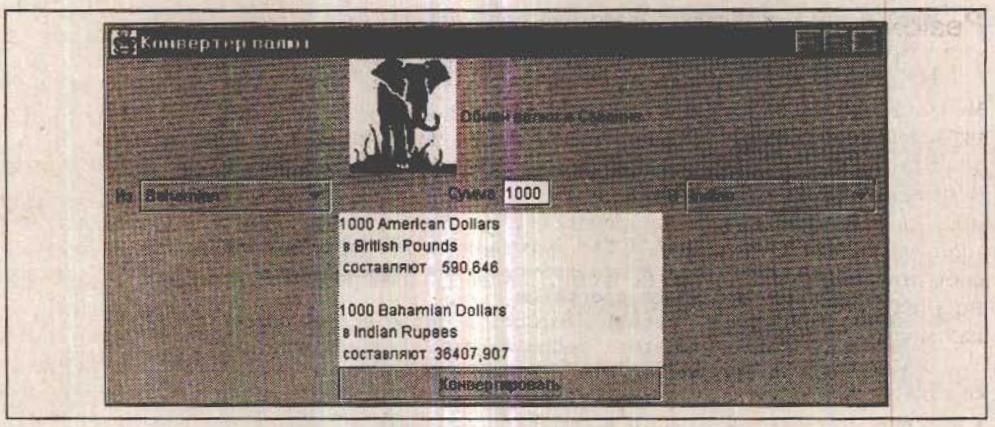


Рис. 11.11. Swing-программа "Конвертер валюты" (вид Metal)

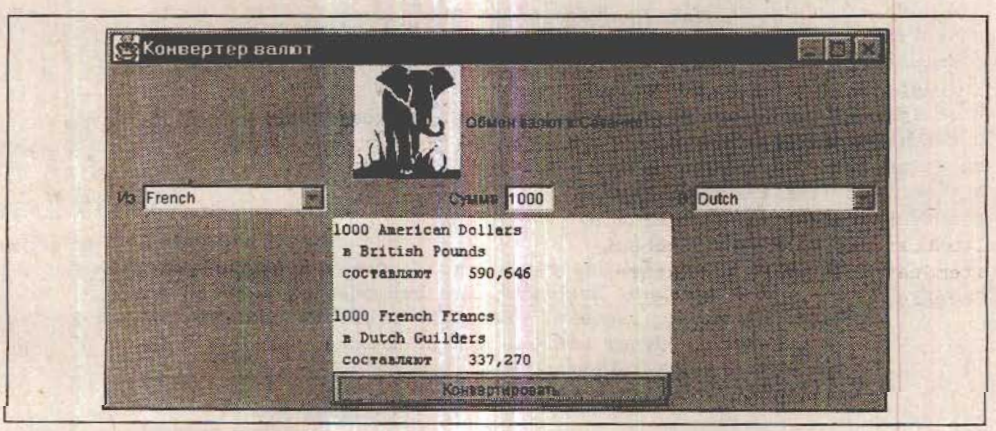


Рис. 11.12. Swing-программа "Конвертер валюты" (вид, присущий Windows)

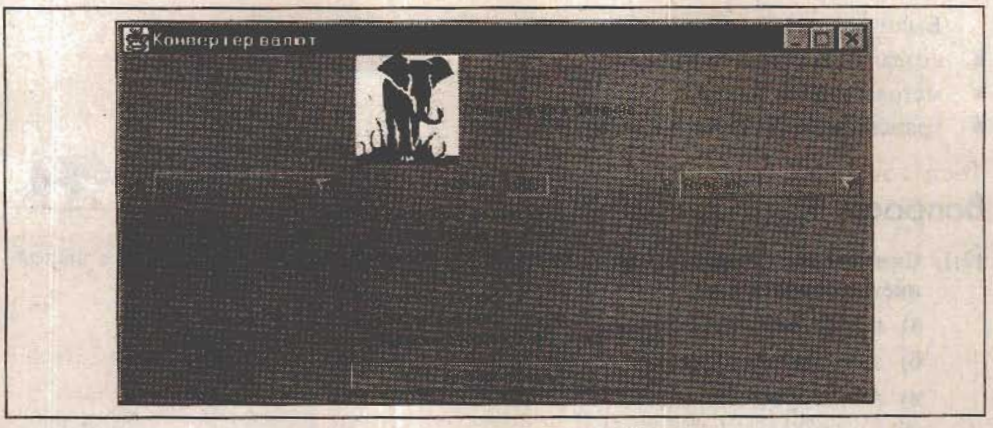


Рис. 11.13. Swing-программа "Конвертер валюты" (вид Motif)

Резюме

Из этой главы мы узнали, как данные вводятся в текстовые поля с помощью меню, кнопок выбора, раскрывающихся списков и т.д. Раскрывающиеся списки ограничивают выбор пользователя, но способны ускорить работу программы.

Взаимодействие с пользователем начинается после выведения на экран компонентов GUI. Передвижение мыши, щелчок мышью на кнопке или ввод текста представляют собой события. События связаны ссылками со слушателями, а слушатели вызывают обработчики, которые выполняют нужные действия. Управляемое событиями программирование требует внимательной разработки алгоритма, особенно в тех случаях, когда необходима определенная последовательность событий перед каким-то действием.

В этой главе не использовались новые ключевые слова, но мы рассмотрели новые составные части awt API. (Кроме приводимых ниже, в табл. 11.1 указан ряд других классов и методов.)

```
TextField           Choice
  getText             getSelectedIndex
  setText             getItemSelectable
  selectText        getSource
  selectAll         getActionCommand
  setEditable       TextArea
  isEditable          append
parseInt            javax.swing.*
ActionEvent           JButton
ItemEvent             JTextField
ItemListener          JComboBox
itemStateChanged      JTextArea
CheckBox              JLabel
CheckBoxGroup       UIManager
```

Были представлены три новых описания:

- создание текстового поля;
- методы класса `TextField`;
- транслирование строки в целое число.



Вопросы

11.1. Считывание объекта `s` класса `String` из текстового поля `TextField t` выполняется оператором:

а) `s = t.readString();`

б) `s = readString(t);`

в) `s = t.getText();`

г) `t.getText(s);`

11.2. Если в программе "Светофор" к выведенным на экран компонентам GUI должен быть добавлен еще один, соответствующий свету с зеленой стрелкой, какой оператор(ы) потребуется, чтобы это было сделано без активизации данного компонента?

а) `buttons.add(walkButton);`

б)
```
walkButton = new Button("Идите");
        buttons.add(walkButton);
```

в) `walkButton = new Button("Идите");`

г) `buttons.add(walkButton("Идите"));`

11.3. Лучшим способом контроля последовательности событий конкретных компонентов, является:

а) сделать все компоненты, кроме требуемого, невидимыми

б) активизировать только требуемый компонент

в) определить булевы переменные, которые назначаются и проверяются после указанных событий соответствующих компонентов

г) поместить каждый из компонентов на отдельную панель

11.4. Какое событие произойдет при введении текста в поле `TextField`?

а) `TextEvent`

б) `ActionEvent`

в) `FieldEvent`

г) `MouseClickedEvent`

11.5. Какой метод используется в методе `actionPerformed` для определения типа произошедшего события?

а) `getEvent()`

б) `getSource()`

в) `getText()`

г) `getAction()`

11.6. Событие в терминологии Java — это:

а) объект

б) класс

в) переменная

г) компонент

11.7. При конструировании новый объект `Frame` появляется на экране:

а) немедленно

б) когда выполняется метод `setVisible(true)`

в) после выполнения конструктора

г) всегда, когда выполняется метод `paint`

11.8. Действия оператора

```
closeButton.addActionListener(this);
```

лучше всего объясненяются так:

а) хранит ссылку на слушатель из текущего объекта (в данном случае из кнопки `closeButton`), причем событие компонента `closeButton` приведет к активизации метода `actionPerformed`, который определен в текущем объекте, наследующем `Frame`

б) хранит ссылку на текущий объект из объекта `amountField`, причем нажатие кнопки `closeButton` приведет к вызову интерфейса `ActionListener`

в) хранит ссылку на объект `closeButton` из объекта `ActionListener`, причем обработчик события, в котором определены действия для `closeButton`, может быть вызван при нажатии этой кнопки

г) хранит ссылку на компонент `closeButton` из текущего объекта, причем при нажатии кнопки `closeButton` интерфейс `ActionListener` может вызвать обработчик события, в котором определены действия для этого объекта

11.9. В новой программе "Конвертер валюты" нам не понадобится хеш-таблица, поскольку вместо нее данные будут храниться в:

а) массиве

б) компонентах `Swing`

в) файле

г) связанном списке

11.10. Для того чтобы в программе "Касса супермаркета" кнопка с названием выбранной продукции после нажатия стала зеленого цвета, следует использовать оператор:

а) `itemButtons[select].setBackground(green);`

б) `itemButtons[i].setBackground(green);`

в) `(Color.green) itemButtons[select];`

г) `itemButtons[select].setBackground(Color.green);`

Задачи

11.1. **Переключатель для рисования фигур.** Учителю необходимо, чтобы дети младшего дошкольного возраста рисовали простые фигуры (круг, квадрат, треугольник), используя три цвета (красный, голубой, желтый). Дети не умеют читать. Обеспечьте подходящий GUI-интерфейс для выбора фигуры и цвета, который бы позволял чертить объект на экране в области, указанной щелчком мыши.

11.2. **Расширение ассортимента продукции.** Измените программу "Касса супермаркета" так, чтобы названия продукции и цены хранились в файле и вводились в начало программы до активизации окна. Должен ли быть изменен размер компонентов в менеджере размещения `GridLayout`, или этот менеджер автоматически разместит большее количество кнопок? Реализуйте новую программу.

11.3. **Чек кассы.** Создайте в отдельном окне чек, который можно было бы распечатать и выдать покупателю овощного магазина. Используйте в качестве образца обычный чек супермаркета.

11.4. **GUI для работы с текстом.** Реализуйте класс, соответствующий классу `Stream` (разделы 4.2 и 7.3) для ввода из текстовых полей `TextField`.

11.5. **Программа "Кофейный магазин Нельсона" с GUI.** Создайте конструкцию полноэкранной версии программы инвентаризации фондов кофейного магазина Нельсона. Перепишите классы так, чтобы был возможен доступ к GUI-компонентам. Выясните, какие могут происходить события, и назначьте соответствующие обработчики событий.

11.6. **Скандинавские флаги.** Четыре скандинавские страны (Норвегия, Швеция, Дания и Финляндия) имеют флаги с похожим дизайном: крест с двойной окантовкой, смещенный относительно центра флага. Расширьте программу "Разработчик флагов" из "Практикума 5" так, чтобы в нее была включена возможность выбора нужного дизайна.

11.7. **Схема предыдущей программы.** В "Практикуме 5" широко использовалось наследование. Нарисуйте полную блок-схему для этой программы, показывая момент, когда уже выбраны все детали для создания флага.

11.8. **Программа "Касса супермаркета", написанная в Swing.** Перепишите программу "Касса супермаркета" из "Практикума 6", используя классы пакета `Swing`. Выясните, оправдывает ли себя использование менеджера размещения `BoxLayout`, свойственного пакету `Swing`. Детали реализации менеджера `BoxLayout` вы можете найти в документации по API Java.

11.9. **GUI, написанные в Swing.** Перепишите все GUI-программы, которые вам предлагалось написать в разделе "Задачи" главы 10, воспользовавшись пакетом `Swing` или программой "Разработчик флагов" из "Практикума 5".

11.10. **Закрепление навыков работы с "холстами".** В программах "Светофор" (примеры 10.5 и 11.2) мы объяснили, как может использоваться дополнительный класс для помещения группы графических объектов в один "холст" и выведения их с помощью одного компонента в окно GUI. Следуя этой технологии, преобразуйте программу "Разработчик флагов" из "Практикума 5" так, чтобы флаг выводился в том же окне, где выводится GUI.

ГЛАВА 12



Апплеты в действии

12.1. От приложения к апплету

Ажиотаж вокруг Java был в большой степени связан с возможностью интеграции этого языка в World Wide Web. Вы убедитесь, что, изучив механизмы доступа к программам и средствам Java через Web и овладев средствами, которые предоставляют стандартные пакеты Java, можно легко создавать апплеты. Причем эти возможности будут ограничены только вашим воображением.

Апплет — это написанная на языке Java программа, которая работает в рамках броузера и, следовательно, может находиться на Web-странице. Апплет, даже если он был доставлен издалека, работает на компьютере пользователя.

Факторы, которые делают возможным такое выполнение программ, — это **интерпретация** (а не компиляция) программ и наличие **встроенной виртуальной Java-машины** (Java Virtual Machine — JVM) во всех броузерах. Благодаря интерпретации приложения Java, которыми мы занимались в предыдущих главах, можно передавать от одного Web-узла к другому и выполнять на разных машинах. Использование броузеров, поддерживающих язык Java, расширяет наши возможности и позволяет работать с апплетами. Апплет — это обычно небольшая программа, которая выполняет одну задачу на одной странице, загруженной в броузер. В качестве примера вы можете посмотреть схемы и копии экрана на рис. 1.1—1.6, где апплеты были представлены в этой книге впервые.

Использование апплетов в Web-страницах дает следующие преимущества.

- Работа выполняется непосредственно на том компьютере, где нужны результаты, т.е. результаты не пересылаются, что **сокращает трафик** в сети.
- Ресурсы компьютера пользователя могут быть полностью **выделены** апплету. Это позволяет выполнять работу быстрее, чем на сервере, где размещена Web-страница, так как последний выделяет для этого лишь часть ресурсов.
- Нам доступны* все средства языка программирования Java. В частности, для взаимодействия с пользовательским интерфейсом используется стандарт пакета `awt`. Это важно, поскольку некоторые специально разработанные для Web языки имеют ограниченные возможности в плане вычислений.

Для того чтобы лучше понять, чем апплет отличается от приложения, вернемся к первому примеру главы 9 (в котором применяется пакет `awt`) и преобразуем его в апплет.

---

* За исключением некоторых ограничений, касающихся безопасности, которые будут описаны ниже.

Преобразование приложения в апплет

Чтобы преобразовать приложение в апплет, нужно выполнить следующие действия.

1. Обеспечить, чтобы все операции **ввода/вывода**, относящиеся к пользователю, проходили через интерфейс, созданный с помощью пакета `awt`. Например, нужно заменить

   ```
   System.out.println("Это предупреждение");
   ```

   на

   ```
   g.drawString("Это предупреждение", 0,0);
   ```

2. Удалить все средства, предназначенные для останова программы (например, кнопку **Закрыть**). Апплет завершает свою работу при закрытии окна программы просмотра или при закрытии в броузере страницы, из которой он был вызван. Апплеты не должны вызывать метод `System.exit()`.
3. **Менеджером размещения, назначенным по умолчанию** для апплетов, является `FlowLayout`. Поэтому, если для фрейма по умолчанию назначен менеджер `BorderLayout`, то нужно явно добавить его вызов (реализацию). Например, добавить оператор

   ```
   setLayout (new BorderLayout ());
   ```

4. Импортировать пакет `applet`, а в главном **окне** вместо `Frame` расширять класс `Applet`*. Например, заменить объявление

   ```
   class GraphicWarning extends Frame {
   ```

   на

   ```
   class AppletWarning extends Applet {
   ```

5. Заменить **конструктор** класса переопределенным методом `init`, который будет вызываться классом `Applet` для одноразовых инициализаций. Например,

   ```
   GraphicWarning ()
   ```

   станет

   ```
   void init ()
   ```

6. Удалить из программы метод **main**, поскольку пакет `applet` возьмет на себя его функции, такие как создание окна и назначение его размеров и видимости (т.е. указание метода `setVisible()` с соответствующим параметром — `false` или `true`).
7. Создать HTML-файл, который будет содержать ссылку на файл класса этого апплета, или включить HTML-инструкции, называемые **тегами**, в готовую Web-страницу (см. приведенное ниже описание).
8. **Вызвать** HTML-файл с помощью программы просмотра апплетов или Web-броузера.

Необходимость некоторых действий вызвана тем, что апплету нельзя импортировать (расширять) класс `Frame` (см. пункт 4). Так, должен быть удален вызов метода `setTitle` (апплет использует имя файла в качестве заголовка, назначенного по

---

* Слово "applet", начинающееся с маленькой буквы — это имя импортируемого пакета; а слово "Applet", начинающееся с большой буквы, обозначает имя класса, который расширяется.

умолчанию), а все методы, которые были ориентированы на класс `Frame`, необходимо заменить соответствующими методами класса `Applet`. Например, вместо метода `dispose` будет применяться метод `destroy`.

HTML

HTML — это **язык разметки гипертекста (HyperText Markup Language)**. Чтобы организовать запуск апплета, достаточно владеть только основами этого языка. Фактически активизация апплета выполняется при помощи пары тегов:

Теги HTML

```
<APPLET code="name" width=n height=m>
</APPLET>
```

При создании HTML-страницы в редакторе броузера могут быть сгенерированы дополнительные теги, указывающие начало и конец HTML-файла и описывающие вывод результатов работы апплета. Однако непосредственно с запуском апплета они не связаны.

В пункте 8 указано, что существует два способа выполнения апплета. Поэтому вначале исследуем, как это сделать с помощью программы просмотра апплетов, поставляемой вместе с Java. Затем, в следующих разделах, рассмотрим, как интегрировать апплеты с броузером.

Пример 12.1. Апплет, предупреждающий о вирусе

Задача. Программа, предупреждающая о возможном наличии вируса, стала популярной, и другие пользователи также хотят ее применять.

Решение. Если программу преобразовать в апплет, его можно будет загрузить на любую машину и там запустить на выполнение.

Конструкция. Выполняя описанные выше действия, мы можем преобразовать приложение из примера 10.1 в апплет. Кроме того, необходимо создать HTML-файл, который содержит следующие теги:

```
<APPLET code="WarningApplet.class" width=200 height=200>
</APPLET>
```

Программа. Программа очень проста и подходит для того, чтобы начать с нее изучение темы, поскольку нам не требуется выполнять некоторые из описанных выше изменений (в частности, действия, описанные в пунктах 2, 3 и 5). Заметьте, что как апплет эта программа короче, чем ее первоначальная версия, так как в апплете отсутствует код, отвечающий за создание и закрытие фрейма. Эти функции возьмет на себя исполняющая система языка Java в броузере или программа просмотра.

```
import java.awt.*;
import java.applet.*;

public class WarningApplet extends Applet {

   /* Программа "Окно предупреждения",
    * преобразованная в апплет          Автор Дж. М. Бишоп, октябрь 1996 г.
    * = = = = = = = = = = = = = =       Java 1.1
    * Апплет должен вызываться при помощи соответствующего HTML-файла
    * в броузере или программе просмотра апплетов.
    */
```

```
  static private final int line = 15;
  static private final int letter =7;

  public void paint (Graphics g) {
    setBackground(Color.white);
    g.drawRect(2*letter, 2*line, 33*letter, 6*line);
    g.drawString("ПРЕДУПРЕЖДЕНИЕ", 9*letter, 4*line);
    g.drawString("Обнаружен возможный вирус", 4*letter, 5*line);
    g.drawString("Перезагрузите компьютер", 5*letter, 6*line);
    g.drawString("и уничтожьте вирус", 7*letter, 7*line);
  }
}
```

Тестирование. Окно с сообщением, полученное в результате работы апплета, выглядит почти так же, как полученное при исполнении приложения. Отличие состоит лишь в том, что заголовок окна устанавливается программой просмотра апплетов и представляет собой имя файла (это является верным признаком того, что окно создано апплетом). На рис. 12.1 показано окно броузера с загруженным HTML-файлом.

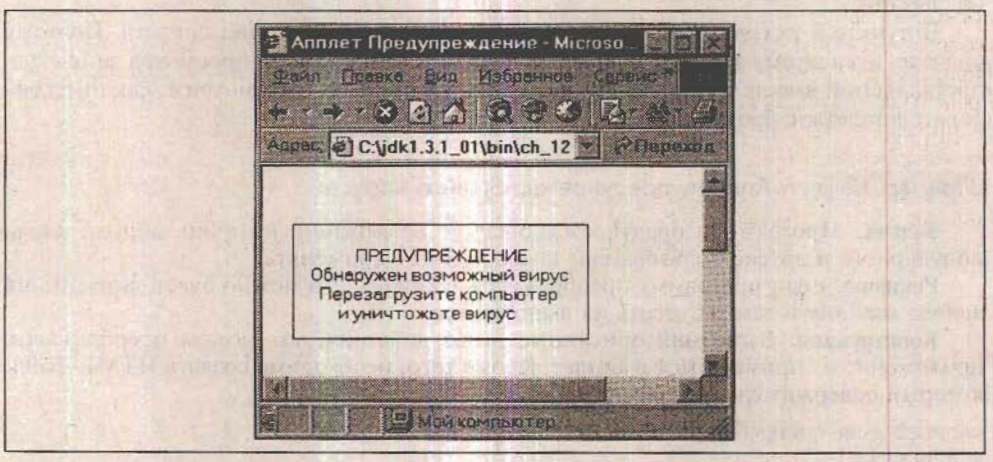


Рис. 12.1. Результат работы апплета — программы `WarningApplet`

Как работает апплет

Поскольку апплет не имеет главной программы, возникают вопросы, как он стартует и как заканчивает свою работу. Апплет стартует и выполняется исполняющей системой Java в рамках броузера (наподобие Netscape или Explorer) либо специальной программой, которую называют программой просмотра апплетов. Виртуальная Java-машина (JVM) просматривает апплет и в соответствующее время вызывает один из приведенных ниже четырех методов, конечно, при условии, что эти методы определены в апплете.

На рис. 12.2 показано отношение между апплетом и классом `Applet`, выполняющимся в Web-броузере. Метод `init` вызывается вместо конструктора, причем тогда, когда броузер впервые обнаружит тег апплета. Этот метод отвечает за обеспечение первоначальных установок апплета, его пользовательского интерфейса, кнопок, меню и любых дополнительных подпроцессов управления. (Подпроцессы рассматриваются в следующей главе.) Метод `start` вызывается всегда, когда Web-страница,

содержащая апплет, вновь появляется на экране. Например, метод `start` может возобновить воспроизведение анимации.

Методы апплета

`init()`
`destroy()`
`start()`
`stop()`

Все или некоторые из этих четырех методов должны быть реализованы каким-либо подклассом класса `Applet`. При необходимости они вызываются JVM. Метод `init` инициализирует любой GUI, относящийся к апплету, а также начало любого подпроцесса (см. главу 13), а метод `destroy` уничтожает их и прекращает работу апплета. Метод `start` инициализирует такие действия, как анимация (вывод на экран движущегося изображения), каждый раз, когда апплет вновь стартует на Web-странице, а метод `stop` останавливает эти действия.

После вызова метода `start` система может вызывать любые другие методы, переопределенные в программе. В примере 12.1 присутствует только метод `paint`, поскольку для данного апплета не нужно определять другие методы. Затем апплет возвращается в пассивное состояние и ожидает, когда произойдет какое-либо событие. Пользователь может "пролистать" его (т.е. посредством полос прокрутки вывести за пределы видимой области Web-страницы) или инициализировать обычное GUI-событие (например, нажатие кнопки мыши). В первом случае апплет становится невидимым, что приводит к вызову метода `stop`, который отвечает за приостановку любого воспроизведения. Когда апплет вновь станет видимым, вызывается метод `start`. Любое другое событие обрабатывается обычным образом с помощью методов пакета `awt` (таких как `actionPerformed`), переопределенных в первоначальной программе.



Рис. 12.2. Отношение между апплетом и классом `Applet`

Кроме того, может быть вызван метод `destroy`, в результате чего апплет освободит занимаемые им ресурсы раньше, чем закончит работу программа просмотра или броузер перейдет на другую страницу. Если мы возвратимся на Web-страницу с апплетом, он вновь будет запущен с "нуля". При этом выполнится последовательность методов, начиная с `init`.

Приведенные выше четыре метода определяются в классе `Applet`, но переопределяются пользователем, который обеспечивает функциональность программы. Кроме того, в классе `Applet` существует группа методов, которые можно использовать без переопределения. Это методы, предназначенные для загрузки изображений, аудиоданных и HTML-параметров, а также методы, позволяющие выяснить, где находится апплет и откуда он поступил. Они будут рассмотрены далее.

12.2. Апплеты в броузерах

До появления языка Java Web-броузеры были в основном статическими и предоставляли информацию в том же виде, в каком она хранилась на сервере. Существовали также средства для передачи информации на хост при помощи электронной почты. Концепция, реализованная в языке Java, позволяет производить вычисления на хосте и открывает новые возможности. Теперь, включив апплеты в Web-страницу, можно предоставить пользователю динамический интерфейс (предлагаем читателю вернуться к рис. 1.1). По окончании работы пользователя ее результаты могут быть возвращены на сервер.

Где находятся апплеты

Если апплет хранится на хост-машине, к нему можно получить доступ с другой машины в Internet. Для этого нужно использовать соответствующий протокол для работы с WWW и указать URL*. Например, URL апплета `WarningApplet` выглядит так:

```
http://www.cs.up.ac.za/javagently/Examples/Ch12/WarningApplet.html
```

Этот HTML-файл необходимо поместить в каталог, доступный для Web. Файл класса, к которому обращается HTML-файл, должен располагаться в том же каталоге, если вы хотите, чтобы для его вызова достаточно было указать только имя:

```
<APPLET code="WarningApplet.class" width=200 height=200>
```

Если апплет находится в другом каталоге, то в теге должен быть указан его полный URL. Теперь мы рассмотрим апплет, который взаимодействует с пользователем, и сравним результаты работы в локальной программе просмотра апплетов и в броузере на удаленном компьютере.

Пример 12.2. Апплет "Касса супермаркета"

Задача. Директора овощных магазинов Саванны рассматривают возможность хранения главной версии программы "Касса супермаркета" в одном месте, чтобы при необходимости ее могли загрузить все кассовые аппараты. Это даст большие преимущества в случае обновления программы, расширения ассортимента или замены компьютеров.

---

* URL — это аббревиатура от Universal Resource Locator (универсальный указатель ресурсов).

Решение. Преобразовать программу в апплет согласно инструкции. Создать HTML-файл и при помощи броузера загрузить его на каждый компьютер (кассу).

Программа. Изменения будут минимальными. Необходимо изменить имя конструктора на `init` и удалить метод `main`. Поскольку апплеты не должны себя закрывать, нужно удалить кнопку **Закрыть** и соответствующий обработчик событий, определенный в методе `actionPerformed`.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class TillApplet extends Applet implements ActionListener {

   /* Программа "Касса супермаркета"   Автор Дж. М. Бишоп, октябрь 1996 г.
    * - - - - - - - - -    Java 1.1 Т.   Абботт и Дж. М. Бишоп, октябрь 1997 г.
    *                                    обновлена август 2000 г.
    *
    * Имитирует работу кассы овощного магазина, имеющего до 12-ти
    * видов продукции; оперирует с ценами за 1 кг.
    *
    * Демонстрирует использование панелей, различных менеджеров
    * размещения (включая GridLayout), пользовательские обработчики
    * событий и непосредственно обработку событий, которые должны
    * происходить в определенном порядке.
    */

  private TextField weighField, totalField;
  private Button[] itemButtons;
  private Button weighButton, printButton;

  public void init() {
    setLayout(new BorderLayout());
    setBackground(Color.cyan);
     Panel p = new Panel();
      p.setLayout(new FlowLayout());
      totalField = new TextField(6);
      totalField.setEditable(false);
      p.add(new Label("Стоимость"));
      p.add(totalField);
    add("North",p);
    Panel q = new Panel();
      q.setLayout(new GridLayout(5, 3));
      itemButtons = new Button[items.length];
      for (int i = 0; i < items.length; i++) {
        itemButtons[i] = new Button(items[i]);
        itemButtons[i].addActionListener(this);
        q.add(itemButtons[i]);
      }
      weighButton = new Button("Взвесить");
        weighButton.addActionListener(this);
        q.add(weighButton);
      printButton = new Button("Распечатать");
        printButton.addActionListener(this);
        q.add(printButton);
    add("Center", q);
    p = new Panel();
```

```
    p.setLayout(new BorderLayout());
    Panel  g = new Panel();
      weighField = new TextField(4);
      weighField.setEditable(false);
      weighField.addActionListener(this);
      g.add(weighField);
      g.add(new Label("Вес"));
    p.add("Center",g);
    p.add("South",new Label("После ввода веса нажмите клавишу \"Enter\""));
  add("South", p);
}

public void actionPerformed(ActionEvent e) {
  Object source = e.getSource();
  if (source == printButton) {
    printItems();
  } else
  if (source == weighButton) {
    resetWeighField();
  } else
  if (source == weighField) {
    readWeighField();
  } else
    selectItem(e.getActionCommand());
  }

 /* Далее следует основное управление программой.
  * Обработчики событий и действий, указанные выше, вызывают
  * методы, которые гарантируют, что ответные действия будут
  * выполнены только при соблюдении определенных условий
  * (например, других предыдущих событий).
  */

public void resetWeighField() {
  weighField.setText("");
  weighField.setEditable(true);
}

public void readWeighField() {
  if (weighField.isEditable()) {
    weighField.setEditable(false);
    weighField.selectAll();
    kg = (double)Integer.valueOf(
    weighField.getText().trim()).intValue();
      weighed = true;
  }
}

public void printItems() {
  if (weighed && chosen) {
    total = kg*unitCosts[select];
    totalField.setText("G "+total);
    System.out.println(kg+"кг "+items[select]+" @ G"+
       unitCosts[select]+" = G"+total);
    kg = 0;
    weighed = false;
```

```
        chosen = false;
        total = 0;
        weighField.select(0,0);
      }
    }

    public void selectItem(String item) {
      select = 0;
      while (!item.equals(items[select]))
        select++;
      chosen = true;
    }

    private String[] items = {"Яблоки", "Груши", "Апельсины",
            "Картофель", "Лимоны", "Арбузы", "Лук", "Чеснок",
            "Авокадо", "", "", ""};
    private double[] unitCosts = {6.00, 5.00, 7.00, 3.00, 10.00, 4.00, 4.00,
                      12.00, 15.00, 0, 0, 0};
    private double total;
    private double kg;
    private boolean chosen = false;
    private boolean weighed = false;
    private int select =1;

}
```

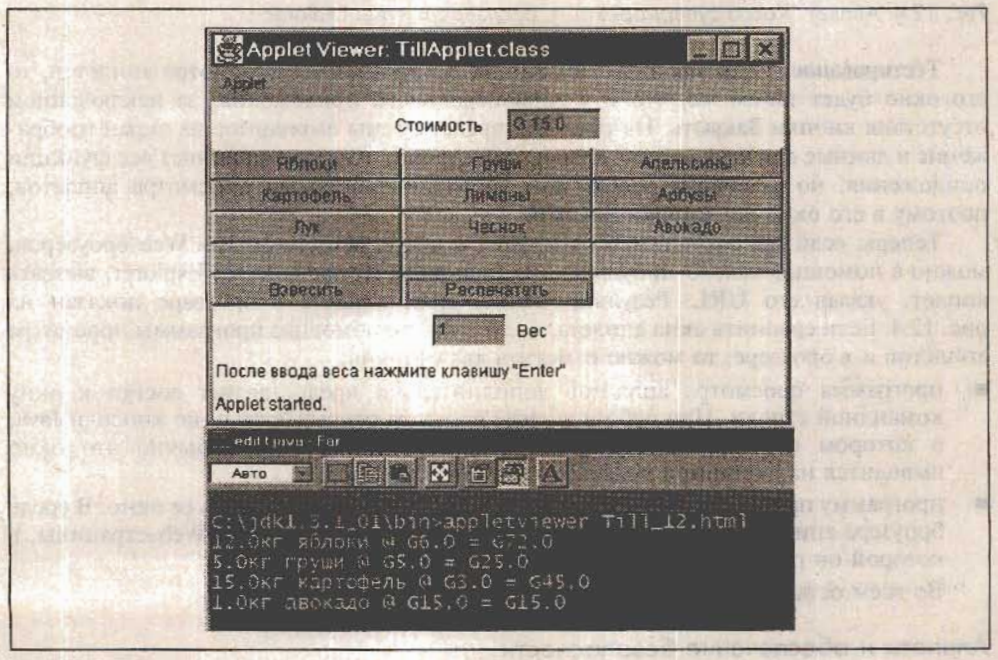


Рис. 12.3. Апплет "Касса супермаркета" в программе просмотра апплетов

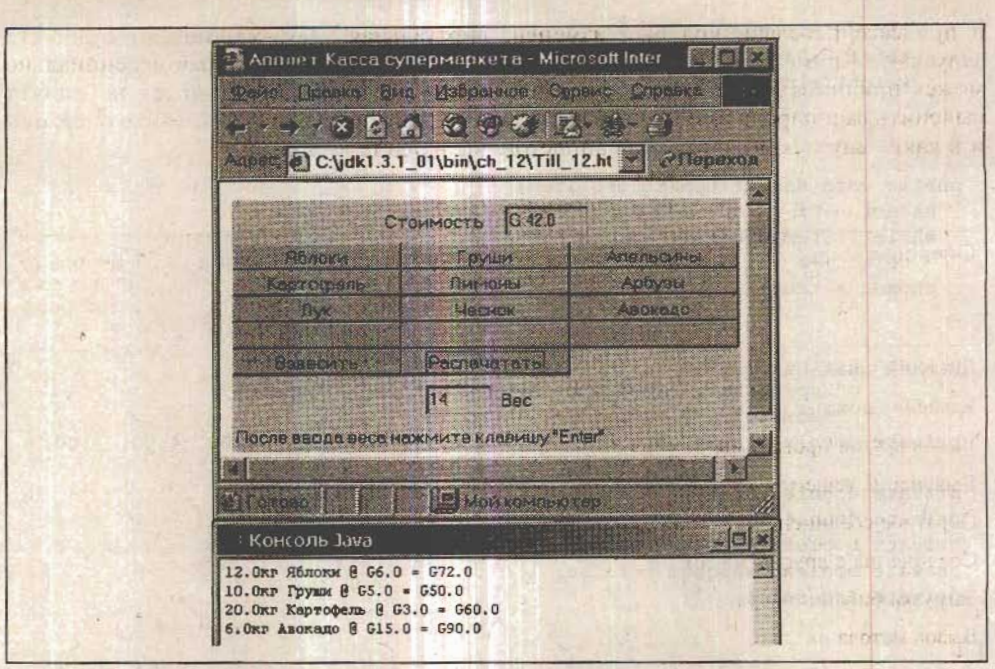


Рис. 12.4. Апплет "Касса супермаркета" в броузере Internet Explorer

Тестирование. Если мы запустим апплет из программы просмотра апплетов, то его окно будет таким же, что и у первоначального приложения, за исключением отсутствия кнопки **Закрыть**. На рис. 12.3 представлены выводимое на экран изображение и данные для чека (в окне командной строки). Апплет выполняет все функции приложения, но завершает работу при закрытии программы просмотра апплетов, поэтому в его окне нет кнопки **Закрыть**.

Теперь, если мы сохраним HTML-файл в месте, доступном для Web-броузеров, можно с помощью какого-нибудь из них (в данном случае Internet Explorer) вызвать апплет, указав его URL. Результат выполнения апплета в броузере показан на рис. 12.4. Если сравнить окна апплета, полученные с помощью программы просмотра апплетов и в броузере, то можно отметить два отличия:

- программа просмотра апплетов дополнительно предоставляет доступ к окну командной строки. При работе в среде броузера открывается окно консоли Java, в котором отображаются результаты работы программы (обычно это окно выводится на экран при помощи меню **Вид** броузера);
- программу просмотра апплетов можно остановить, если закрыть ее окно. В среде броузера апплет прекращает свою работу только при закрытии Web-страницы, в которой он работает.

Во всем остальном апплет ведет себя одинаково во всех средах.

Апплеты и обеспечение безопасности

Если апплет запускается из Web-броузера, существует вероятность того, что он *окажется поврежденным или зараженным вирусом* и навредит вашей системе. Java всеми возможными способами защищает систему от этого. В первую очередь, осуществляется проверка целостности байт-кода, поступающего на компьютер. Если

в процессе передачи код был изменен, виртуальная Java-машина не будет его запускать. Кроме того, JVM не станет выполнять операцию, которая потенциально может причинить вред вашей системе. Например, прибывший апплет не сможет выяснить ваш пароль или удалить ваши файлы. Информацию о том, какие операции и в каких случаях доступны для апплетов, вы найдете в табл. 12.1.

Таблица 12.1. Выполнение апплетами опасных операций

Операция	Приложение Java	Апплет в программе просмотра	Локальный (собственный) апплет в броузере	Прибывший апплет в броузере
Доступ к локальным файлам	✓	✓		
Удаление локальных файлов	✓			
Вызов другой программы	✓	✓		
Выяснение вашего имени	✓	✓	✓	
Обратное соединение с хостом	✓	✓	✓	✓
Соединение с другим хостом	✓	✓	✓	
Загрузка библиотеки Java	✓	✓	✓	
Вызов метода `exit`	✓	✓		
Создание всплывающего окна	✓	✓	✓	✓

Давайте рассмотрим, что представляет собой каждый элемент таблицы. Приложение Java — это полноценная программа и поэтому может делать на вашем компьютере все, что потребуется. Что касается апплетов, то здесь существуют следующие варианты. Если апплет работает в программе просмотра, он может выполнять все операции, доступные приложению, за исключением удаления файлов. Когда апплет выполняется в броузере, его возможности значительно уменьшаются.

В броузере апплет не завершает работу сам, она прекращается, когда страница, в которой он находится, сменяется другой страницей. Следовательно, апплет не вызывает метод `exit`, а на панели пользовательского интерфейса отсутствуют кнопки его закрытия. Тот факт, что апплеты не могут осуществлять чтение и запись в локальной файловой системе, может показаться недостатком, но при необходимости он может быть устранен пользователем посредством определения специального менеджера безопасности. Наконец, из последнего столбца таблицы видно, что к апплетам, загруженным из удаленных хостов, применяются дополнительные ограничения. Все они введены с хорошими намерениями, но, конечно, усложняют жизнь программистам.

Дополнительные действия, которые придется выполнить в случае, когда апплет не может считать локальный файл, описаны далее и продемонстрированы в примере 12.3.

Использование тега PARAM

Вы уже знаете, что ввод данных в апплет не всегда нужно осуществлять путем считывания информации из файла. Если апплет выполняется в броузере, можно воспользоваться средствами ввода данных в Web-страницу и способностью апплета извлекать их оттуда. Однако механизм этот довольно громоздкий, поэтому применять его лучше для одноразового ввода данных.

Взаимодействие между апплетом и Web-страницей достигается путем использования тега PARAM. При этом предусматривается определение в HTML-документе именованных параметров, которые апплет считывает при помощи метода getParameter, определенного в классе Applet. Параметры размещаются между тегами <APPLET> и </APPLET>. Каждый параметр должен быть указан согласно следующему описанию:

Параметр HTML
<PARAM NAME = "*formal*" VALUE = "*actual*">

Атрибут NAME указывает, что следующая за ним строка является именем искомого параметра. В рамках одного комплекта угловых скобок, формирующих пару, есть значение VALUE, которое должно быть назначено этому параметру. Значение — это также строка, но она может быть подвергнута анализу как целое число или значение другого типа. Возможным применением тегов PARAM в программе "Касса супермаркета" будет перемещение процедуры инициализации цен продуктов из апплета в Web-страницу. Они должны быть указаны следующим образом:

```
<PARAM NAME= "Чеснок" VALUE="12">
```

Для доступа к параметрам мы используем метод работы со строками getParameter:

```
unitCosts[i] = Integer.parseInt(getParameter(items[i]));
```

Здесь items[i] — это строка, например "Чеснок". Метод getParameter должен найти параметр с таким именем и вернуть соответствующее значение, которым в данном случае является строка "12". Теги PARAM используются в примере 12.3 и в "Практикуме", а также необходимы в некоторых задачах, приведенных в конце главы.

Подписанные апплеты

К коду, который вызывает сомнения и загружается извне, следует применять все указанные выше ограничения. Однако даже в рамках одной компании может возникнуть потребность ввести некоторые правила безопасности, например, в случае, если апплеты пишутся в главном офисе и рассылаются филиалам. Меры безопасности понадобятся также и при необходимости считывать локальные файлы, расположенные в филиалах. JDK обеспечивает возможность создания апплетов с цифровыми подписями. Это делается при помощи специального инструмента — программы, которая называется keytool. Компания должна вести небольшую базу данных доверенных апплетов. Ввод апплетов в базу осуществляется с использованием открытых и закрытых ключей. Апплет сохраняется как архивный файл (JAR*), и затем JAR-файл передается программе keytool. HTML-файл, в котором вызывается апплет, должен содержать дополнительный тег для доступа к подписанному архивному файлу.

Сказанное выше применимо к JDK, другие IDE (Integrated Development Environment — интегрированные среды разработки) и броузеры могут иметь иные способы объявления и обработки апплетов, помеченных как безопасные.

---

* JAR — это аббревиатура от Java archive (архив Java).

JAR-файлы

Сейчас мы рассмотрим, каким образом апплет может получить доступ к пакетам, которые определены пользователем, таким как `javagently` и `jgeslib`. Загрузчик класса апплета не в состоянии найти нужные ему классы, если они находятся в других каталогах. Однако мы не можем просто скопировать все необходимые файлы классов в рабочий каталог, поскольку при этом в программе нарушатся структура пакета и процедура импорта. В результате программу нельзя будет скомпилировать.

К счастью, Java предлагает решение этой проблемы. Мы можем создать архив пакета Java (JAR) и затем ссылаться на него в HTML-файле, содержащем апплет. Перейдите в каталог, расположенный выше того, в котором находится пакет, и вызовите команду `jar`, как описано ниже.

Команда jar

```
jar cf packagename.jar packagename\*.class
```

В результате будет создан архив, включающий все файлы классов пакета в сжатом виде. Например:

```
jar cf javagently.jar javagently\*.class
```

Чтобы получить доступ к архиву, в HTML-файл, из которого вызывается апплет, нужно добавить следующий тег:

```
archive = javagently.jar
```

Кроме возможности сохранения структуры пакета на удаленном компьютере, использование JAR-файлов позволяет получить меньший объем файла апплета и, следовательно, загрузить его быстрее.

Пример 12.3. Счастливое имя

Задача. Газета *Savanna News* устраивает еженедельный отбор участников для розыгрыша ценного приза. Отбор базируется на именах и фамилиях читателей газеты. Каждой букве алфавита присваивается числовое значение, а для имени и фамилии назначается проходная сумма. Если вы читатель газеты и сумма числовых значений букв ваших имени и фамилии равна проходной сумме, то вы получаете право участвовать в розыгрыше главного приза. Для того чтобы уравнять возможности читателей, имена расширяются или сокращаются до 20 символов, при этом пробелы и знаки пунктуации удаляются. Отбор участников проходит каждую неделю, причем числовые значения символов и проходная сумма каждый раз меняются. Те, кто прошел в заключительный тур, имеют шанс выиграть еженедельный приз в 1000 грац.

Пример

```
Алфавит          a b c d e f g h i j k l m n o p q r s t u v w x y z
Значения         5 1 3 6 7 9 2 3 7 5 4 1 3 8 9 7 5 4 3 2 1 3 2 4 5 4
Проходная сумма  100

Имена                    Используемые буквы          Сумма
Judy Bishop              JudyBishopJudyBishop        94
John Smith               JohnSmithJohnSmithJo        100
Roelf van den Heever     RoelfvandenHeeverRoe        118
Timothy Fox-Fox-Robinson TimothyFoxFoxRobinso        116
```

Из примера видно, что на этой неделе в заключительный тур вышел Джон Смит. Остальные могут вновь участвовать в отборе на следующей неделе, когда изменятся числовые значения букв и проходная сумма.

Редактор газеты хочет реализовать отбор претендентов через Web, чтобы читатели, проверив сумму числовых значений своих имен, могли немедленно узнать результат.

Решение. Конечно, написать Java-апплет! Такая процедура отбора претендентов на главный приз идеально подходит для апплета, поскольку в данном случае он будет содержать информацию, назначаемую централизованно (числовые значения для букв). Эта информация должна быть защищена, но при этом каждый должен иметь возможность выполнить вычисления. Кроме того, подсчитывать сумму вручную — занятие довольно скучное, поэтому большим плюсом для апплета будет наличие функции, производящей вычисления.

Разработка экрана. Полезно начать с разработки экрана для апплета и определить, какие поля для ввода и вывода данных нужно создать, чтобы обеспечить обмен информацией с пользователем. Кроме того, мы должны решить, что еще будет размещаться на Web-странице и какие данные будут поступать в апплет. Текст (заголовки) будем выводить на экран средствами HTML, а не из апплета, поскольку так проще. Один из возможных вариантов компоновки экрана представлен на рис. 12.5.

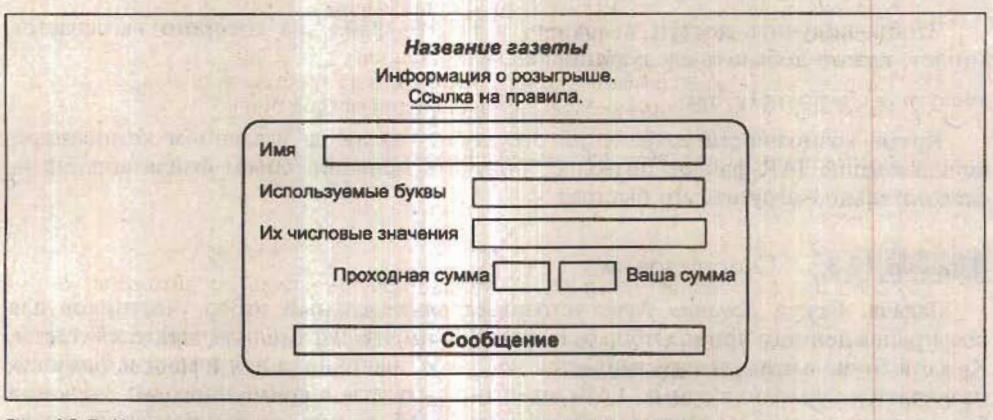


Рис. 12.5. Компоновка экрана для проведения розыгрыша призов

Алгоритм. Код апплета можно разделить на две части. Первая часть занимается выводом элементов экрана и назначением данных; на вторую возложена организация ввода имени и вычисление результата. Вид экрана мы разработали. Теперь настало время подумать над организацией данных.

Нам нужна таблица, соотносящая символы с целыми числами. К сожалению, как мы знаем из раздела 6.4, Java позволяет применять для индексирования элементов массива только целые числа. Поэтому вместо массива мы должны использовать хеш-таблицу. Символы и соответствующие им целые числа помещаются в таблицу при помощи метода `put`, а извлекаются оттуда методом `get`.

В этом случае возникает вопрос, как вводить значения в апплет. Нам нужно, чтобы значения могли быть легко изменены персоналом газеты, т.е. людьми, которые не обязательно являются программистами. Если ввести эти значения как фиксированные константы, то каждую неделю, когда их нужно будет изменять, потребуется перекомпилировать программу, что не очень удобно.

Однако существуют и другие варианты.

1. Хранить значения в Web-странице как параметры и затем извлекать их оттуда.
2. Соединяться с файлом на главном компьютере газеты и считывать значения.

Оба эти варианта реализуемы. О том, как соединиться с удаленным источником, рассказывается в главе 14, поэтому здесь мы будем использовать теги PARAM. Именами параметров будут строка **Сумма** и символы `a`, `b`, `c`, `d` и т.д., т.е. каждая буква латинского алфавита.

Интерфейс имеет одно поле ввода с меткой **Ваше имя**, кнопку **Новое имя** и пять текстовых полей, в которых выводятся следующие данные: используемые буквы, их числовые значения, проходная сумма, сумма введенного имени и сообщение.

При вводе метод `actionPerformed` должен будет обрабатывать события, происходящие в объектах `TextField` и `Button`. Щелчок мышью на кнопке **Новое имя** должен привести к установке полей в первоначальное состояние (удалению из них всех строк), что сделать не трудно; намного сложнее обработать имя. Далее мы приводим требуемый алгоритм.

Проверка суммы вводимого имени

Ввести текст.
Удалить пробелы, которые могут быть в конце имени.
Перевести буквы в нижний регистр.
Удалить из имени все пробелы и знаки пунктуации.
Сократить имя до 20 символов или повторить его до требуемой длины (20).
Извлечь из хеш-таблицы числовые значения для каждой буквы,
подсчитать сумму.
Сумма имени совпадает с проходной суммой?

Да — Сообщение с пожеланием удачи.

Нет — Стандартное пожелание "Играйте еще!"

Для обработки имени будем использовать несколько методов, предоставляемых классом `String`. Это методы `trim`, `toLowerCase` и `toCharArray`. Последний метод преобразует строку в массив символов. Это удобно, поскольку числовые значения для каждого символа также хранятся в виде массива и мы можем использовать для них один алгоритм вывода.

Web-страница

Web-страница должна иметь заголовки и хранить параметры. Текст страницы, созданной в редакторе броузера, приведен ниже.

```
<HTML>
<HEAD>
  <TITLE>Розыгрыш </TITLE>
</HEAD>
<BODY>
<CENTER>
<H1>
<B><FONT COLOR="#3366FF"><FONT SIZE=+2>
<I>Savanna News </I>Розыгрыш </FONT></FONT></B>
</H1>
```

```
</CENTER>

<CENTER>
<P>
 <B>Если ваше имя наберет указанную сумму,<BR>вы можете выиграть 1000G </B>
</CENTER>
</P>
<CENTER>
<B><A HREF="Rules.html">Правила </A> розыгрыша.</B>
</CENTER>

<CENTER>
<APPLET CODE="Competition.class"  WIDTH=450 HEIGHT=180>
<PARAM  NAME="Сумма"  VALUE="100">
<PARAM  NAME="a"  VALUE="5"><PARAM NAME="b" VALUE="1">
<PARAM  NAME="c"  VALUE="3"><PARAM NAME="d" VALUE="6">
<PARAM  NAME="e"  VALUE="7"><PARAM NAME="f" VALUE="9">
<PARAM  NAME="g"  VALUE="2"><PARAM NAME="h" VALUE="3">
<PARAM  NAME="i"  VALUE="7"><PARAM NAME="j" VALUE="5">
<PARAM  NAME="k"  VALUE="4"><PARAM NAME="l" VALUE="1">
<PARAM  NAME="m"  VALUE="3"><PARAM NAME="n" VALUE="8">
<PARAM  NAME="o"  VALUE="9"><PARAM NAME="p" VALUE="7">
<PARAM  NAME="q"  VALUE="5"><PARAM NAME="r" VALUE="4">
<PARAM  NAME="s"  VALUE="3"><PARAM NAME="t" VALUE="2">
<PARAM  NAME="u"  VALUE="1"><PARAM NAME="v" VALUE="3">
<PARAM  NAME="w"  VALUE="2"><PARAM NAME="x" VALUE="4">
<PARAM  NAME="y"  VALUE="5"><PARAM NAME="z" VALUE="4">
</APPLET>
</CENTER>
</BODY>
</HTM L>
```

Апплет. Обратите внимание на фрагмент кода апплета, в котором имя извлекается из текстового поля и преобразуется в массив символов. Все это выполняет следующий оператор:

```
c = nameField.getText().trim().toLowercase().toCharArray();
```

Метод `getText` возвращает строку, которая затем передается двум методам класса `String` — `trim` и `toLowerCase`, также возвращающим строку. Метод `toCharArray` получает строку и преобразует ее в массив символов.

После этого производится удаление пробелов и перевод символов в нижний регистр.

Затем для каждого элемента этого массива извлекается соответствующее ему числовое значение из хеш-таблицы. Хэш-таблица хранит строки, поэтому результатом выполнения оператора

```
String s = (String) values.get(String.ValueOf(d[i]));
```

будет строка. Чтобы построить другой массив символов, состоящий из числовых значений букв (все эти значения меньше 10, т.е. являются одноразрядными числами), мы можем просто извлечь первый символ строки, используя метод `charAt`. Однако для вычисления суммы мы должны преобразовать строку в целое число. Эти действия выполняют приведенные ниже операторы.

```
e[i] = s.charAt(0);
total += Integer.parseInt(s);
```

Обратите внимание на то, что мы используем в апплете несколько шрифтов. Обычный шрифт указывается так:

```
Font g = new Font("SansSerif",Font.PLAIN,12);
```

Для букв и их числовых значений применяем шрифт с фиксированной шириной, который задается следующим оператором:

```
Font h = new Font("Monospaced",Font.PLAIN,12);
```

Выводимое на экран сообщение будет представлено шрифтом большего размера:

```
Font f = new Font("Serif",Font.BOLD,20);
```

Код Java-апплета приведен ниже. По объему он меньше, чем код приложения, но выполняет те же функции.

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Competition extends Applet implements ActionListener {

   /* Апплет "Розыгрыш"    Автор Дж. М. Бишоп, октябрь 1996 г.
    * = = = = = = = = = =    Java 1.1 январь 1998 г.
    * Создает GUI, улучшающий проведение розыгрыша.
    * Демонстрирует взаимодействие между апплетом и Web-страницей.
    */

  private static int winningScore;
  private TextField targetField, nameField,
          lettersField, valuesField, scoreField, resultField;
  private Button againButton;
  private Hashtable values = new Hashtable ();
  private String openingMessage = "Ваше имя может сделать вас победителем!";
  private static final int lettersCounted = 20;
  String letters [] = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j",
         "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u" ,"v", "w",
         "x", "y", "z"};

  public void init () {

    // Вначале генерируются числовые значения букв и
    // создается хеш-таблица. Для этого используются значения в
    // тегах PARAM, которые считываются из Web-страницы.
    // И ключ, и значения являются строками.

    winningScore = Integer.parseInt(getParameter("Сумма"));
    for (int i = 0; i<26; i++)
      values.put(letters[i], getParameter(letters[i]));
    // Выводит на экран элементы пользовательского интерфейса.
    setLayout (new BorderLayout ());
    Panel p = new Panel ();
      Font g = new Font("SansSerif",Font.PLAIN,12);
```

```
    p.setFont(g);
  nameField = new TextField ("",40);
    p.add (new Label ("Ваше имя:"));
    p.add (nameField);
    nameField.addActionListener(this);
  Panel r = new Panel ();
    Font h = new Font("Monospaced",Font.PLAIN,12);
    r.setLayout(new GridLayout(2,2));
    lettersField = new TextField("",20);
      lettersField.setEditable(false);
      r.add(new Label("Используемые буквы:"));
      r.add (lettersField);
      lettersField.setFont(h);
    valuesField = new TextField("",20);
      valuesField.setEditable(false);
      r.add(new Label("Их числовые значения:"));
      r.add(valuesField);
      valuesField.setFont(h);
  p.add(r);
  targetField = new TextField(Integer.toString(winningScore),4);
    targetField.setEditable(false);
    p.add (new Label ("Проходная сумма"));
    p.add(targetField);
  scoreField = new TextField ("",4);
    p.add (scoreField);
    p.add (new Label("Ваша сумма"));
    scoreField.setEditable(false);
   againButton = new Button("Новое имя");
    againButton.addActionListener(this);
    p.add (againButton);
add("Center", p);

Panel q = new Panel ();
  resultField = new TextField (openingMessage);
    resultField.setEditable(false);
    Font f = new Font("Serif",Font.BOLD,20);
    resultField.setFont(f);
    q.add(resultField);

add("South", q);
}

public void actionPerformed (ActionEvent evt) {
  Object source = evt.getSource();

  // При нажатии кнопки Новое имя удаляются все данные,
  // находящиеся в четырех текстовых полях, а в пятом выводится
  // первоначальное сообщение.
  if (source == againButton) {
    nameField.setText(" ");
    scoreField.setText(" ");
    valuesField.setText(" ");
    lettersField.setText(" ");
    resultField.setText(openingMessage);
```

```
    }

    // Было введено имя.
    else
      handleName();
  }

  void handleName(){
    char c [];
    char d [] = new char[lettersCounted];
    char e [] = new char[lettersCounted];

    // Переводит буквы имени в нижний регистр и удаляет пробелы.
    c = nameField.getText().trim().toLowerCase().toCharArray();
    int n=0;
    for (int i=0; i<c.length & n < lettersCounted; i++) {
      if (Character.isLetter(c[i])) {
        d[n] = c[i];
        n++;
      }
      if (i==c.length-1) i=-1;
    }

    // Символы массива теперь находятся в промежутке между d[0]
    // и d[n], а мы помещаем соответствующие значения в массив e[].
    int total = 0;
    for (int i=0; i<n; i++) {
      String s = (String) values.get(String.valueOf(d[i]));
      e[i] = s.charAt(0);
      total += Integer.parseInt(s);
    }

    // Подсчитывает и сравнивает суммы. Выводит сообщение.
    scoreField.setText(Integer.toString(total));
    lettersField.setText(String.valueOf(d));
    valuesField.setText(String.valueOf(e));
    if (total == winningScore)
        resultField.setText("Желаем удачи в розыгрыше приза!");
    else
        resultField.setText("Увы, но мы ждем вас на следующей неделе! ");
  }

}
```

Тестирование. Апплет можно протестировать, используя имена, приведенные в начале примера, или любые другие. На рис. 12.6 показан именно тот случай, когда сумма введенного имени совпала с проходной суммой.

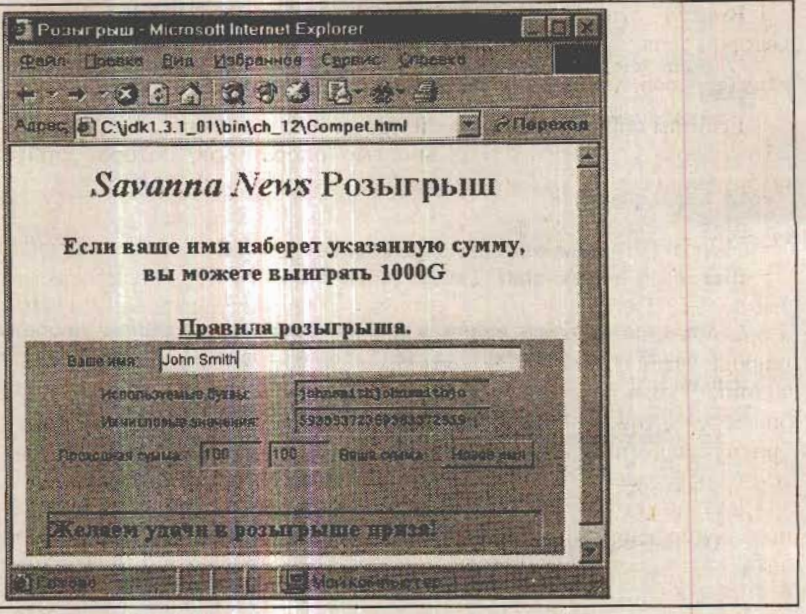


Рис. 12.6. Апплет Competition в броузере Internet Explorer

12.3. Звук и изображения

Одним из важных достоинств языка Java является возможность легко и эффективно внедрять в программу звук и изображения.

Звук

Интерфейс пакета applet, называемый AudioClip, имеет три метода: play, stop и loop. Метод getAudioClip класса Applet возвращает объект, который реализует этот интерфейс, после чего мы можем воспроизвести данный объект.

Объявление и воспроизведение объекта AudioClip

```
AudioClip name;
name = getAudioClip (getCodeBase (), filename);
name.play();
```

Метод getCodeBase класса Applet выясняет, из какого каталога запущен апплет, чтобы затем искать в этом каталоге звуковой файл. Воспроизводимый файл должен иметь расширение .au, но не .wav.

Предположим, что у нас есть файл с именем ouch.au и мы хотим его воспроизвести. Выполняется это с помощью следующих операторов:

```
AudioClip ouch;
ouch = getAudioClip (getCodeBase (), "ouch.au");
ouch.play();
```

Вместо этой группы операторов можно применять сокращенный вариант, в котором используется анонимный класс:

```
play(getCodeBase (), "ouch.au");
```

Если вы собираетесь в своей программе запускать клип несколько раз, то лучше воспользоваться последовательностью операторов, которая приведена в таблице. (Применение звука продемонстрировано в примере 12.4.)

Изображения

В данном случае мы рассматриваем изображения, состоящие из пикселей, т.е. растровые. Изображения можно взять из файла на диске, получить по сети либо создать в режиме реального времени графической программой или видеокамерой. Java имеет широкий набор средств для обработки этого многообразия источников изображений, в частности, при работе в сети. Вероятно, вы уже сталкивались с тем, что изображения в броузере загружаются с различной скоростью. Одним из наиболее удобных свойств броузера является возможность автоматической загрузки изображений. Это означает, что текст продолжает поступать во время вывода изображения на экран.

Java может обеспечить аналогичный способ управления изображениями с помощью интерфейсов, которые называются `Observer` (наблюдатель). Мы можем "оживлять" изображение, контролировать его мерцание и фильтрацию, а также воспроизводить видео с различной скоростью. Однако подробно эти возможности мы описывать не будем. Расскажем только, как извлечь изображение и вывести его на экран, а также как организовать его перемещение по экрану.

Как и в случае с аудиоданными, для изображений Java предоставляет соответствующий класс. Способ объявления и воспроизведения этого класса совпадает с приведенным выше для звука, с той только разницей, что интерфейс `AudioClip` заменяется интерфейсом `Image`. Поэтому оператор, выполняющий извлечение изображения, выглядит так:

```
Image me;
me = getImage (getCodeBase (), "bishop.gif");
```

Для вывода изображения на экран используется метод `drawImage`, определенный в классе `Graphics` пакета `awt`. Описание вывода изображения на экран выгладит следующим образом:

Вывод изображения на экран

```
g.drawImage (Image imageid, int x, int y,
       ImageObserver observer);
```

Координаты `x` и `y` указывают координаты верхнего левого угла области, в которую выводится изображение. В большинстве случаев наблюдателем (`observer`) является апплет, поэтому в качестве четвертого параметра используем специальную переменную `this`. Например:

```
g.drawImage (me, 0, 0, this);
```

Отслеживание (tracking) изображения

Наблюдатели позволяют нам выполнять определенные действия во время загрузки изображений. Однако может возникнуть ситуация, когда требуется противоположное: удостовериться, что изображение загружено, прежде чем выполнять другой

оператор. Это бывает важно, когда апплет загружает изображение в своем методе `init`. Если изображение указано как подлежащее загрузке, а апплет "спрашивает", насколько оно большое, то он может получить неверный ответ. Поэтому мы передаем апплет в качестве параметра объекту класса `MediaTracker` для отслеживания загрузки изображения. Это выполняют методы `addImage` и `waitForID`, приведенные в следующем описании.

Применение объекта класса `MediaTracker` для отслеживания загрузки изображения

```
MediaTracker tracker = new MediaTracker(applet);
tracker.addImage(Image imageid, 0);
 try {tracker.waitForID(0);}
catch (InterruptedException e) {}
```

Объект `MediaTracker` отслеживает изображение, и когда возвратится значение 0 (которое мы выбрали), его можно перехватить и таким образом определить, что изображение загружено. Этот метод успешно применяется в "Практикуме".

Перемещение изображения

Технология перемещения изображения по экрану хорошо известна, Java же делает ее еще проще. Во многих других системах при перемещении изображения в другое место экрана вы должны позаботиться о том, чтобы удалить изображение в прежнем месте. В Java при использовании метода `paint` это не требуется. Но вот что вы обязательно должны делать, так это задавать координаты для перемещения объектов при помощи переменных, а не констант. Как это делается, показано в примере 12.4.

Реагирование на события, связанные с мышью

Различные события, слушатели и обработчики событий, предоставляемые языком Java, рассмотрены в разделе 11.2. После этого мы главным образом имели дело с классом `ActionEvent`, который описывает любое действие, производимое пользователем над `awt`-компонентом (т.е. событие), и соответствующими действию слушателем (методом `ActionListener`) и обработчиком событий (методом `actionPerformed`). Интерфейс `ActionListener` является особым слушателем, поскольку имеет только один метод обработки. Если вы посмотрите в табл. 11.1, то увидите, что большинство других событий имеют несколько методов.

Тонкость заключается в том, что слушатели — это абстрактные интерфейсы, а при использовании интерфейса мы должны обеспечить реальные версии *всех* его абстрактных методов, даже если их не применяем. Поэтому, чтобы обнаружить нажатие мыши, необходимо определить метод `mousePressed`, а также пустые версии всех остальных относящихся к мыши четырех методов, которые присутствуют в классе `MouseListener`. Чтобы сэкономить время, в пакет `java.awt.event` для каждого абстрактного интерфейса-слушателя включена его реализация, называемая **адаптером (adapter)**. Абстрактный класс-адаптер предоставляет пустые версии всех методов слушателя. Затем, вместо реализации слушателя, мы наследуем адаптер и переопределяем все методы, которые нам действительно нужны. На рис. 12.7 показано, как эта последовательность действий применяется к двум слушателям мыши, которые используются для связанных с мышью событий. Во всех методах в качестве параметра указывается класс `MouseEvent`.

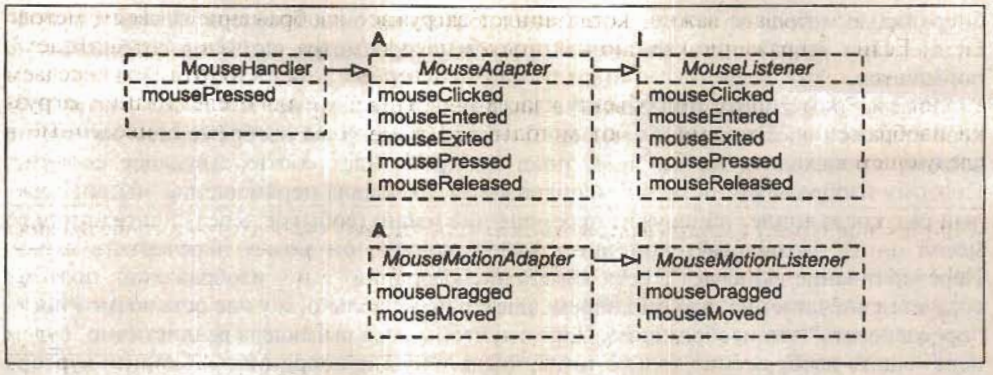


Рис. 12.7. Схема, иллюстрирующая отношения между слушателями и адаптерами, предназначенными для работы с мышью

Поэтому, если вы хотите, чтобы апплет "слушал мышь" (т.е., чтобы в ответ на события мыши он выполнял определенные действия), в его метод `init` необходимо включить оператор

```
this.addMouseListener(new MouseHandler());
```

а также внутренний класс `MouseHandler`, в котором определяется метод `mousePressed`, обрабатывающий событие (нажатие кнопки мыши). В следующем примере демонстрируется использование обоих слушателей, предназначенных для работы с мышью, `MouseListener` и `MouseMotionListener`.

Пример 12.4. Поймать гнома

Задача. Создать простую игру, которая состоит в том, что пользователь "преследует" и "ловит" (щелкая мышью) изображение, перемещающееся по экрану. В игре должны присутствовать звуковые эффекты.

Решение. В первую очередь нам нужны один `gif`-файл и два `au`-файла, один из которых предназначен для озвучивания "поимки" гнома, а другой — для неудачной попытки. Из опыта можем порекомендовать следующее: `gif`-файлы должны быть не больше 10 Кбайт, а `au`-файлы (для нормального воспроизведения) не меньше 6 Кбайт. Это минимальные требования, но вполне возможно, что ваш компьютер может оперировать файлами гораздо больших объемов. Мы должны вывести изображение на экран и позволить игроку преследовать его с помощью мыши. Если пользователю удалось поймать гнома, это сопровождается одним звуком, если не удалось и он "промазал" — другим. Чтобы усилить впечатления, выведем соответствующие сообщения.

Алгоритм. Метод апплета, предназначенный для перемещения изображения, работает следующим образом. После того как изображение извлечено и выведено на экран, мы можем получить его длину и ширину. Когда пользователь щелкает кнопкой мыши, метод `getID` возвращает константу `MOUSE_PRESSED` (в некоторых книгах она называется `MOUSE_DOWN`), и можно проверить, находятся ли текущие координаты мыши в области, определяемой текущими координатами изображения. Отреагировав на событие соответствующим звуком и сообщением, мы обновляем экран, заставляя изображение переместиться. Перемещение изображения выполняется путем добавления к координатам произвольного значения, которое передается в метод `drawImage`.

С помощью метода showStatus в строку состояния окна броузера выводится сообщение. При выполнении апплета в программе просмотра апплетов сообщение не появляется.

Рассмотрим работу программы в динамическом режиме. Когда же фактически перемещается изображение? Очевидно, что с выведенным изображением ничего не должно происходить до тех пор, пока не произойдет соответствующее событие. Поэтому изображение остается неподвижным до начала перемещения мыши. Каждый раз, когда апплет обнаружит перемещение мыши (событие, в результате которого метод getID возвращает константу MOUSE_MOVED), он может перечертить экран. Перечерчивание включает в себя изменение координат x и y изображения, поэтому создается впечатление, что оно перемещается произвольно, и у нас есть возможность "преследовать" это изображение. Для того чтобы игра выглядела реалистично, будем перемещать изображение только тогда, когда текущие координаты позиции курсора мыши кратны 3.

Программа. Программа соответствует алгоритму и описаниям, которые приведены выше.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class CatchM extends Applet {

   /* Программа "Поймать гнома"   Автор Дж. М. Бишоп, октябрь 1996 г.
    * = = = = = = = = = = = = =   Java 1.1 январь 1998 г.
    *
    * Пользователю предлагается поймать гнома
    * щелчком левой кнопки мыши на его изображении.
    * Демонстрирует использование звука, изображения
    * и обработки событий,
    * связанных с мышью - перемещение мыши и нажатие кнопки.
    */

  int mx, my, limitx, limity ;
  int wins;
  int boardSize;
  Image duke;

  public void init() {
    wins = 0;
    boardSize = getSize().width - 1;
    duke = getImage(getCodeBase(),"duke.gif");
    this.addMouseListener (new mousePressHandler());
    this.addMouseMotionListener (new mouseMotionHandler());
  }

  class mousePressHandler extends MouseAdapter {

    public void mousePressed (MouseEvent e) {
      int x = e.getX();
      int y = e.getY();
      requestFocus();
      if (mx < x && x < mx+limitx &&
```

```
          my < y && y < my+limity) {
        wins++;
        getAppletContext().showStatus("Попался!  Всего " + wins);
        AudioClip ouch;
        ouch = getAudioClip (getCodeBase (), "jahoo1.au");
        ouch.play();
      }
      else {
        getAppletContext().showStatus("Снова промазал.");
        AudioClip haha;
        haha = getAudioClip (getCodeBase (), "beep.au");
        haha.play();
      }
      repaint();
    }
  }

  public class mouseMotionHandler extends MouseMotionAdapter {
    public void mouseMoved(MouseEvent e) {
      if (e.getX() % 3 == 0 && e.getY() % 3 == 0)
        repaint();
    }
  }

  public void paint(Graphics g) {
    // Перед определением размера изображения ждет, пока изображение
    // будет загружено.
    // Эти операторы нельзя поместить в метод init.
    limitx = duke.getWidth(this);
    limity = duke.getHeight(this);
    int change = boardSize-limitx;

    // Чертит окно.
    g.drawRect(0, 0, boardSize, boardSize);

    // Вычисляет новое местоположение гнома
    // и выводит его изображение на экран.
    mx = (int)(Math.random()*1000) % change;
    my = (int)(Math.random()*1000) % change;
    g.drawImage(duke, mx, my, this);
  }
}
```

Тестирование. По вполне очевидным причинам очень трудно показать тестирование этой программы в книге. На рис. 12.8 представлена копия экрана, полученная при выполнении* апплета в броузере Internet Explorer. Программу можно найти на сайте *JavaGently*. Это как раз тот случай, когда вы должны указать адрес (расположение) апплета в Web, чтобы увидеть и услышать результаты его выполнения.

---

* В апплете были использованы аудиофайлы jahoo1.au и beep.au, которые находятся в каталоге C:\jdk1.3.1_01\demo\applets\TicTacToe\audio, разумеется, если вы инсталлировали JDK на ваш диск C. — *Прим. ред.*

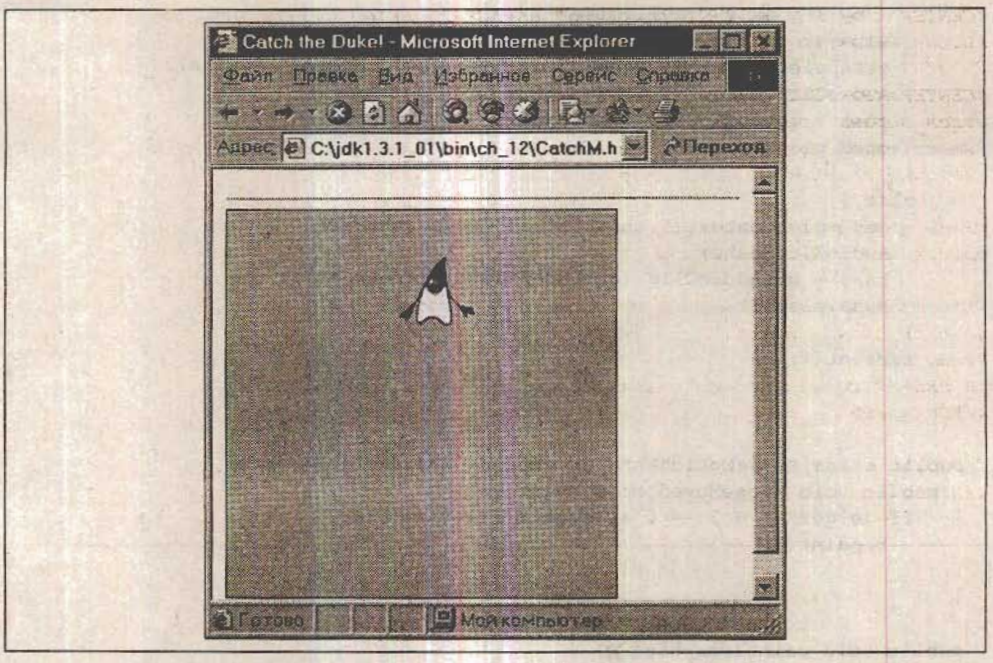


Рис. 12.8. Программа "Поймать гнома" в Internet Explorer

12.4. Практикум 8. Проект по охране природы

В главе 1 был описан проект по созданию Web-узла для демонстрации фотоснимков животных, птиц и деревьев саванны. Теперь мы обладаем достаточными знаниями для того, чтобы исследовать компоновку такой системы и определить связи между HTML-сценариями, апплетами Java и изображениями, демонстрирующимися в сайте.

Web-страницы

На рис. 1.3 (здесь представлена первая страница сервера) мы видим несколько компонентов: заголовок, текст, изображение и две ссылки (на страницу с фотографиями животных и страницу с фотографиями деревьев). Третья ссылка (на страницу с фотографиями птиц) была обещана, но еще не реализована. Для создания этой страницы нужно использовать интерактивный редактор броузера или написать HTML-файл. В любом случае у нас должен быть HTML-документ, который в наиболее простой форме выглядит так, как приведенный ниже.

```
<HTML>
<BODY>

<CENTER><P><FONT COLOR="#0000FF"><FONT SIZE=+3>
Проект по охране природы Саванны
</FONT></FONT></P></CENTER>

<P></P>
```

```
<CENTER><P><IMG SRC="Elephant.jpg" HSPACE=20 HEIGHT=117 WIDTH=178
ALOGN=LEFT></P></CENTER>

<CENTER><P><FONT COLOR="#3366FF"><FONT SIZE=+1>
Отдел охраны природных ресурсов Саванны <BR>
 имеет своей целью помочь вам выяснить информацию о животных, <BR>
 которые населяют равнины Африки.</FONT></FONT></CENTER></P>

<P>Вы можете также сотрудничать с нами, зарегистрировать ваши интересные
фотоснимки и интерактивно добавить их к нашим данным.</P>

<P>Версия 1 нашей домашней страницы имеет разделы по темам:
<A HREF="Animals.html">животные</A>,
птицы и
<A HREF="Trees.html">деревья</A>.
</FONT></P>

</BODY>
</HTML>
```

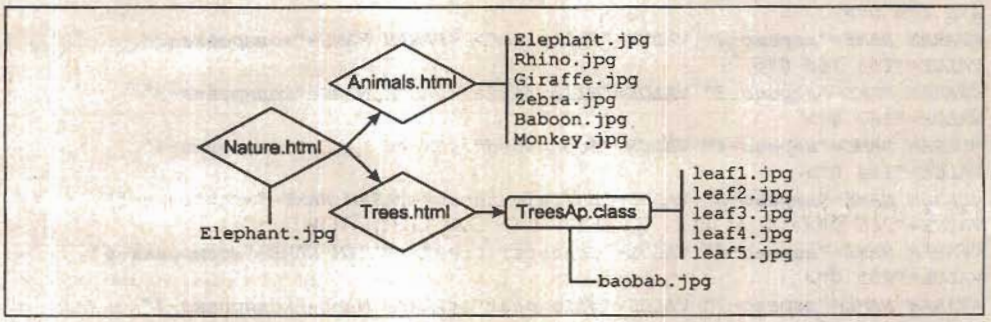


Рис. 12.9. Организация информации на Web-узле проекта по охране природных ресурсов

Документ начинается с информации, содержащейся в заголовке. Она печатается шрифтом, размер которого больше размера обычно используемого шрифта. Предполагается, что изображение слона (файл `Elephant.jpg`) находится в том же каталоге, что и HTML-файл, и сохранено в JPG-формате. Параметры изображения указывают, что оно будет размещено слева от текста, который может за ним следовать*. Следующие три абзаца содержат текст, причем последний включает в себя также две ссылки на другие страницы с именами файлов `Animals.html` и `Trees.html`.

Файл `Animals.html` создан по такому же принципу, за исключением того, что содержит шесть изображений, выводимых на экран. Схема, иллюстрирующая организацию информации на нашем узле, представлена на рис. 12.9. Стрелками показаны ссылки, а простые линии указывают на изображения и апплеты, включаемые в HTML-файлы.

---

* Это означает, что если бы в этом блоке HTML-кода присутствовал еще и текст, то он был бы размещен справа от изображения. — *Прим. ред.*

Включение апплета в HTML-файл

Документ `Trees.html` представляет Web-страницу, показанную на рис. 1.5. Ниже приведена сокращенная версия его HTML-кода.

```
<HTML>
<BODY>
<CENTER><P><FONT COLOR="#008000"><FONT SIZE=+2>
Деревья Саванны</FONT></FONT></P></CENTER>

<CENTER><P> Введите в соответствующие поля размеры листа в миллиметрах <BR>
и укажите его тип.<BR> Программа может идентифицировать перечисленные ниже
деревья.</P>
</CENTER>
<CENTER><TABLE BORDER=1 WIDTH="70%">

<TR>
<TD><APPLET CODE="TreesAp.class" WIDTH=360 HEIGHT=300>
<PARAM NAME="записей" VALUE="19">

<PARAM NAME="дерево-1" VALUE="Protea"> <PARAM NAME="кодировка-1" VALUE="165
175 176 0">
<PARAM NAME="дерево-2" VALUE="Milkplum"> <PARAM NAME="кодировка-2"
VALUE="156 166 0">
<PARAM NAME="дерево-3" VALUE="Wild fig"> <PARAM NAME="кодировка-3"
VALUE="167 0">
<PARAM NAME="дерево-4" VALUE="Beechwood"><PARAM NAME="кодировка-4"
VALUE="164 0">
<PARAM NAME="дерево-5" VALUE="Buffalo thorn"<PARAM NAME="кодировка-5"
VALUE="265 0">
<PARAM NAME="дерево-6" VALUE="Confetti tree"><PARAM NAME="кодировка-6"
VALUE="255 0">
<PARAM NAME="дерево-7" VALUE="Wild pear"><PARAM NAME="кодировка-7"
VALUE="277 0">
<PARAM NAME="дерево-8" VALUE="Smelly shepherd"><PARAM NAME="кодировка-8"
VALUE="266 267 0">
<PARAM NAME="дерево-9" VALUE="Monkey orange"><PARAM NAME="кодировка-9"
VALUE="355 0">
<PARAM NAME="дерево-10" VALUE="Powder puff"><PARAM NAME="кодировка-10"
VALUE="377 588 0">
<PARAM NAME="дерево-11" VALUE="Baobab"><PARAM NAME="кодировка-11"
VALUE="367 0">
<PARAM NAME="дерево-12" VALUE="Sausage tree"><PARAM NAME="кодировка-12"
VALUE="588 0">
<PARAM NAME="дерево-13" VALUE="Worm cure albizia"><PARAM
NAME="кодировка-13" VALUE="578 0">
<PARAM NAME="дерево-14" VALUE="Weeping boer-bean"><PARAM
NAME="кодировка-14" VALUE="577 0">
<PARAM NAME="дерево-15" VALUE="Pride of De-Kaap"><PARAM NAME="кодировка-15"
VALUE="567 0">
<PARAM NAME="дерево-16" VALUE="Fever tree"><PARAM NAME="кодировка-16"
VALUE="667 0">
<PARAM NAME="дерево-17" VALUE="Camel thorn"><PARAM NAME="кодировка-17"
VALUE="677 0">
<PARAM NAME="дерево-18" VALUE="Umbrella thorn"<PARAM NAME="кодировка-18"
```

```
VALUE="654 0">
<PARAM NAME="дерево-19" VALUE="Candle acacia"><PARAM NAME="кодировка-19"
VALUE="666 0"></APPLET></TD>

</APPLET>

</TD> </CENTER>

<TD ALIGN=RIGHT><FONT COLOR="#0080FF"><FONT SIZE=-1>
Баобаб<BR>
Корковый дуб<BR>
Молочная слива<BR>
Дикое фиговое дерево<BR>
Малярийное дерево<BR>
Верблюжий шип<BR>
Шип-зонтик<BR>
Свечевидная акация<BR>
... и другие деревья<BR>
</FONT></FONT>
</TD>
</TR>
</TABLE>
<FONT COLOR="#0080FF"></FONT>
</BODY>
</HTML>
```

Сначала определим заголовок страницы и разместим на ней небольшой фрагмент текста. Затем выведем окно апплета и рядом с ним список названий деревьев. Один из простых способов организации этих компонентов на экране — создание таблицы, в одной ячейке которой располагается апплет, а в другой — список названий деревьев. Учтите, что побочным эффектом этого метода наличие рамок таблицы.

В теге апплета указано, что апплет находится в файле `TreesAp.class` и окно, в котором он будет выведен, имеет определенную высоту и ширину. Затем следует длинный список названий деревьев и коды размеров их листьев (это мы рассмотрим далее). Список из девятнадцати деревьев содержится также и в HTML-коде (этим задается его появление в правом столбце таблицы).

Ввод данных в апплет

Рассмотрим еще раз апплет `TreesAp`, окно которого показано на рис. 1.5. Задача этого апплета — получив три параметра, характеризующих дерево (длину листа, его ширину и тип листорасположения), вывести на экран информацию о дереве, отвечающем заданным критериям. На рис. 1.5 показано, какие данные вводятся для баобаба, а на рис. 1.6 представлен результат поиска. Числовые данные передаются в апплет путем ввода их в поля, а листорасположение выбирается щелчком мышью на соответствующей кнопке.

Однако этот апплет должен также хранить список названий деревьев и соответствующих каждому дереву параметров. Как это сделать? Существует три способа:

- инициализировать массив данными, как это было сделано в программе "Касса супермаркета";
- использовать тег `PARAM`;
- считывать данные из файла, используя удаленное соединение.

Для наших целей, вероятно, самым подходящим средством будет использование тега PARAM. Чтобы иметь параметры, формирующие массив (в нашем случае это 19 деревьев), мы должно определить 19 имен. Если назвать параметры похожими именами (например, дерево-1, дерево-2 и т.д.), можно эффективно запрограммировать процесс их извлечения. Соответствующий оператор выглядит так:

```
String tree = getParameter("дерево-"+(i+1));
```

Кодирование размеров листьев

Из данных, введенных пользователем, апплет конструирует трехзначный код. Каждое дерево может иметь несколько (скажем, до 4-х) таких кодов, поскольку существует определенный диапазон размеров листа, который должен учитываться. Поэтому для каждого дерева нужно иметь его название, как было описано выше, и комплект кодов для его идентификации. Например, второе дерево (дерево-2), молочная слива, имеет листья, длина которых варьируется от 40 до 80 см, а ширина от 20 до 30 см. Эти значения представлены двумя кодами (согласно системе кодов, которая предложена Юджином и Глен Молл и Никки Пэйдж*), а именно 156 и 166. Поэтому параметр кодировки (кодировка-2) для второго по списку дерева (дерево-2) будет выглядеть так:

```
<PARAM NAME="кодировка-2" VALUE="156 166">
```

Конечно, природу закодировать трудно, и в реальном мире вполне могут существовать другие деревья, которые по своим данным подойдут под код 156 или 166. Но в основном предложенная система кодировки работает, поэтому мы можем продолжить заполнение списка названиями деревьев, ввести их вместе с кодами в HTML-файл и активизировать апплет.

Внутри апплета

В завершение предлагаем вам обратить внимание на следующие вопросы, касающиеся работы апплета:

- как достигается нужное расположение изображений моделей листьев;
- как имена деревьев и их коды хранятся в хеш-таблице и как к ним осуществляется доступ;
- как осуществляется обработка событий, необходимая для передачи в программу введенных пользователем данных и возвращения изображения.

Все необходимые пояснения вы найдете в комментариях, которые располагаются перед соответствующими блоками кода.

Еще одной особенностью апплета является то, что он реагирует на ввод пользователя, окрашивая текстовые поля в зеленый цвет. По завершении ввода данных пользователь нажимает кнопку **Старт**, и если HTML-файл с изображением этого дерева существует, соответствующая фотография будет выведена на экран. Если щелкнуть на кнопке **Новый поиск**, все поля ввода вновь станут белыми и пользователь может вновь производить поиск. Аналогичный метод можно применить также в апплетах "Касса супермаркета" и "Розыгрыш".

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
```

---

* Eugene и Glen Moll и Nicci Page, *Common Trees*, Struik, 1989.

```
import java.util.*;

public class TreesAp extends Applet implements ActionListener {

   /* Апплет "Деревья"    Автор Дж. М. Бишоп январь 1997 г.
    * ================    Java 1.1 январь 1998 г.
    * Запрашивает информацию о листьях дерева, кодирует ее
    * и переходит к URL, где хранится дальнейшая информация о дереве.
    * Получает названия деревьев и их коды из HTML-страницы.
    * Демонстрирует взаимодействие с броузером, работу с массивами кнопок,
    * отслеживание вывода изображения; использование менеджеров
    * размещения для вложенных панелей; применение разделителей
    * строки на лексемы.
    */

   // Текстовые поля, булевы переменные и целые числа,
   // необходимые для указания параметров листа.
   private TextField widthField, lengthField, messageField;
   private Button againButton, startButton;
   private boolean gotChoice = false;
   private boolean gotWidth = false;
   private boolean gotLength = false;
   private boolean start1 = false;
   private int c, w, l;

   private Hashtable table = new Hashtable ();

   private String patterns [] = {
     "Чередующееся",   "Противоположное",
     "Вееровидное",    "Пальмовидное",
     "Комплексное",    "Хвоеподобное"};

   private Button patternButton [] = new Button[6];

   public void init () {
   //===================
   // Вначале вводятся параметры для HTML-таблицы.
      setUpTable();

   // Вывод на экран компонентов изображения: двух текстовых полей
   // вверху и одного внизу. В центре менеджером GridLayout выводятся
   // шесть моделей расположения листьев на ветке. Каждая модель
   // представляет собой самостоятельную панель, использующую менеджер
   // размещения BorderLayout с кнопкой в области
   // South и холстом, занимающим остальную площадь. Рисование холста
   // выполняется в классе leafPattern. Назначен 10-й размер шрифта.
   Font f = new Font ("SansSerif",Font.PLAIN,10);
   this.setFont (f);
   setLayout(new BorderLayout(0,0));
   Panel p = new Panel ();
     p.add (new Label ("Ширина листа"));
     widthField = new TextField (8);
       p.add (widthField);
       widthField.addActionListener(this);
     p.add (new Label ("Длина листа "));
     lengthField = new TextField (8);
       p.add (lengthField);
```

```
    lengthField.addActionListener(this);
add ("North",p);

// Создает панель для шести типов расположения листьев на ветке.
Panel q = new Panel ();
  q.setLayout(new GridLayout(2,3));
  for (int b = 0; b<patterns.length; b++) {
    Panel s = new Panel ();
      s.setLayout(new BorderLayout(0,0));
      s.add ("Center",new leafPattern (this,b));
      patternButton[b] = new Button(patterns[b]);
        patternButton[b].addActionListener(this);
        s.add ("South",patternButton[b]);
    q.add (s);
  }
add ("Center",q);

  // Далее следует вывод текстового поля для сообщения и кнопок
  // "Старт" и "Новый поиск" в область South,
  // т.е. в нижнюю часть апплета.
  Panel r = new Panel();
    messageField = new TextField (30);
      r.add (messageField);
      startButton = new Button ("Старт");
      r.add(startButton);
      startButton.addActionListener(this);
    againButton = new Button ("Новый поиск");
      r.add(againButton);
      againButton.addActionListener(this);
  add("South",r);
}
private void setUpTable() {
//========================
    String s, item;
    StringTokenizer t;
    int code;
    s = getParameter("записей");
    int n = Integer.parseInt(s);
    for (int i = 0; i<n; i++) {
    // Извлекает название дерева из HTML-файла.
      String tree = getParameter("дерево-"+(i+1));
    // Считывает все его коды.
      s = getParameter("кодировка-"+(i+1));
      t = new StringTokenizer (s);
    //Сохраняет в таблице каждую пару код-дерево.
      while (true) {
        try {
          item = t.nextToken();
          code = Integer.parseInt(item.trim());
          table.put(new Integer(code),tree);
        }
        catch (NoSuchElementException e) {
          break;
        }
      }
   }
}
```

```
public void actionPerformed (ActionEvent e) {
//=========================
  Object source = e.getSource();
  String name = e.getActionCommand();
    if (source == widthField) {
    w = readValue(widthField);
    gotWidth = true;
  } else if (source == lengthField) {
    l = readValue(lengthField);
    gotLength = true;
  } else if (source == startButton) {
    startl = true;
  } else if (source == againButton) {
     resetFields();
  } else if (name instanceof String) {
    c = 0;
    while (!name.equals(patterns[c])) c++;
    gotChoice = true;
    patternButton[c].setBackground(Color.green);
  }

  // Следующие 12 строк кода были введены в программу только для
  // того, чтобы пользователь мог прочитать в текстовом поле
  // строку "Идет поиск файла ..." и нажать кнопку Старт.
  if (gotChoice & gotLength & gotWidth){
     int part2=convertLength();
     int part3=convertWidth();
     int code = 100*(c+1)+10*part2+part3;
     String tree = (String) table.get(new Integer (code));
  if (tree == null)
    setMessage("Нет информации по кодировке "+code);
    else {
       String s = tree+".html";
       setMessage("Идет поиск файла "+s);
    }
  }
  if (gotChoice & gotLength & gotWidth & startl)
    findTree ();
}
private void findTree () {
//===============
// Подсчитывает код, состоящий из трех частей.
  int part2=convertLength();
  int part3=convertWidth();
  int code = 100*(c+1)+10*part2+part3;
// Извлекает название дерева из хеш-таблицы,
// если такое название существует.
  String tree = (String) table.get(new Integer (code));
  if (tree == null)
    setMessage("Нет информации по кодированию "+code);
  else
    fetchTree (tree);
}

private void fetchTree (String treeName) {
//=====================
```

```
    // Пытается найти и показать Web-страницу, базируясь на
    // информации о дереве.
      try {
       AppletContext context = getAppletContext ();
        String s = treeName+".html";
        setMessage("Идет поиск файла "+s);
        URL u = new URL (getCodeBase(),s);
        context.showDocument(u,"_self");
      }
      catch (MalformedURLException e) {
        setMessage("По этому дереву нет информации.");
      }
    }

    // Методы преобразования (получения) кода.
    //==================

    private int convertLength () {
    // Возвращает код, базируясь на информации о длине листа,
    // введенной пользователем.
      if (l<=25) return 4; else
      if (l<=50) return 5; else
      if (l<=100) return 6; else
      if (l<=200) return 7; else
                  return 8;
    }
    private int convertWidth () {
    // Возвращает код, базируясь на информации о ширине листа,
    // введенной пользователем.
      if (w<=10) return 4; else
      if (w<=20) return 5; else
      if (w<=40) return 6; else
      if (w<=80) return 7; else
                 return 8;
    }

  // Методы, обслуживающие текстовые поля.
  // ===============
    private int readValue (TextField t) {
        int x = Integer.parseInt(t.getText());
        t.setEditable(false);
        t.setBackground(Color.green);
        return x;
    }
    private void clearField(TextField t) {
      t.setEditable(true);
      t.setBackground(Color.white);
      t.setText("");
    }

    private void resetFields () {
      gotChoice = false;
      patternButton[c].setBackground(Color.lightGray);
      gotWidth = false;
```

```
    clearField(widthField);
    gotLength = false;
    clearField(lengthField);
    clearField(messageField);
    messageField.setEditable(false);
  }

  private void setMessage(String s) {
    messageField.setText(s);
    messageField.setBackground(Color.green);
  }

  class leafPattern extends Canvas {
  //====================
    Image im;

    leafPattern(Applet a, int b) {
      weAre = a;
    // Конструирует имя (адрес) URL
      pattern = "images/leaf"+Integer.toString(b)+".jpg";
    // Извлекает изображение
      im = weAre.getImage(weAre.getCodeBase(),pattern);
    // Гарантирует, что изображение будет получено прежде,
    // чем конструктор продолжит свою работу.
      MediaTracker tracker = new MediaTracker(weAre);
      tracker.addImage(im, 0);
      try { tracker.waitForID(0);}
      catch (InterruptedException e) {}
    }

    public void paint (Graphics g) {
      g.drawImage(im,0,0,this);
    }

    private Applet weAre;
    private String pattern;
  }
}
```

Тестирование. Результаты работы аплета в окне броузера показаны на рис. 12.10. Если вы хотите выбрать конкретное дерево, просмотрите его параметры в файле `Trees.html`. На рисунке (в поле для вывода сообщений) видно, что заданные значения подходят для баобаба.

Некоторые размышления по поводу авторских прав. Написание апплетов, в которых предусматривается применение изображений, довольно интересное занятие. Однако при этом следует учитывать серьезное ограничение: фотографии, сделанные не вами, обычно защищены авторским правом. Из этого следует, что вы не можете свободно использовать их в своих Web-страницах, даже экспериментальных. Поэтому, фотографии для главы 1 сделаны автором этой книги.

Возможности, которые предоставляют мультимедийные средства Java, настолько мощны и просты в применении, что нередко программист забывает, что мультимедийное содержимое его апплета или Web-страницы должно быть предварительно кем-то создано. Ищите изображения, которые можно свободно использовать, а если сомневаетесь в их доступности, спросите разрешение у создателя или издательства.



Рис. 12.10. Апплет "Деревья" (файл `Trees.html`) в броузере Internet Explorer

Резюме

Апплеты похожи на приложения, однако они выполняются в броузерах или программах просмотра апплетов и с учетом контекста класса `Applet`, поэтому не нуждаются в методе `main`. Все процедуры ввода/вывода в апплетах должны осуществляться через пакет `awt`. Апплет должен иметь сопутствующий ему HTML-файл, из которого он будет вызываться. Этот HTML-файл можно загрузить в программу просмотра апплетов или броузер либо вызвать из Web-страницы. Апплеты, выполняемые в броузерах, прекращают свою работу только при закрытии страницы, в которой находятся. Апплеты могут получать данные из Web-страницы с помощью тега `PARAM`. Java предоставляет средства защиты для того, чтобы апплеты, прибывшие по сети, не могли причинить вред вашему компьютеру.

Апплеты, также как и приложения, поддерживают работу со звуком и изображениями. Эти элементы можно включить в Java-программу, используя объекты для соединения с файлами. Последние должны иметь расширения `.au` и `.gif` или `.jpg` соответственно. Вы можете создавать изображения, передвигающиеся по экрану, если при перечерчивании экрана зададите для них новые координаты. Отслеживание загрузки изображений требует применения дополнительных средств.

В этой главе не вводились новые ключевые слова, но были представлены такие новые API-классы и методы:

```
java.applet.*          MediaTracker
Applet                   addImage
  init                   waitForID
```

```
  destroy              Observer
  start                InterruptedException
  stop                 MouseEvent
  getCodeBase          MouseListener
  getAppletContext       mousePressed
  getParameter             mouseMoved
Image                    mouseDown
  getImage             MouseAdapter
  drawImage
```

Следующий перечень включает темы, на которые в главе обращено особое внимание:

- теги HTML для активизации апплета;
- методы апплета;
- параметры в HTML-коде;
- команда `jar`;
- объявление и воспроизведение объекта `AudioClip`;
- вывод изображения на экран;
- применение объекта класса `MediaTracker` для отслеживания загрузки изображения.

Вопросы



12.1. Выберите лучший ответ на вопрос: что такое апплет?

а) программа, которая хранится на локальном жестком диске и запускается из командной строки

б) программа, которая хранится на удаленной системе, передается локальной системе и выполняется в Web-броузере

в) программа, которая хранится и выполняется на удаленной машине, а результаты ее работы передаются локальной системе и выводятся на экран в Web-броузере

г) программа, которая хранится и выполняется на удаленной машине, а результаты ее работы передаются локальной системе и выводятся на экран в консольном окне Java или в окне командной строки

12.2. С позиций программирования на языке Java ответьте на вопрос: что означает термин "интерпретация"?

а) компилятор выясняет, что означает код, затем, используя словарь трансляции, создает исполняемый файл программы

б) компилятор в качестве промежуточного этапа генерирует байт-код, который затем сохраняется на жестком диске как исполняемый файл

в) компилятор генерирует байт-код из кода-источника и сохраняет этот байт-код в двоичной форме в файле класса. Когда вы запускаете программу, модуль, вызываемый виртуальной машиной Java, считывает байт-код и преобразует его в машинный код, который затем выполняется как обычно. Этим гарантируется, что программа может быть выполнена на любом компьютере (на любой платформе)

г) компилятор считывает любые строки, представляющие код программы, и транслирует их во все существующие в мире языки программирования, поэтому программа сможет исполняться везде

12.3. Какие преимущества дает возможность включения и выполнения апплетов в Web-страницах?

а) уменьшение сетевого трафика, более быстрое выполнение кода и возможность доступа ко всем средствам и функциям Java

б) возможность создавать больше нюансов ("наворотов") в Web-странице

в) то же самое, что и в пункте а), плюс повышенная безопасность

г) предоставляет удаленному программисту доступ к машине пользователя

12.4. Апплет прекращает свою работу, когда:

а) в броузер загружается новая Web-страница

б) пользователь щелкает мышью на кнопке **Закрыть**, находящейся в окне апплета

в) вызывается метод `System.exit()`

г) когда в окне броузера пользователь пролистывает страницу и окно апплета становится невидимым

12.5. Менеджером размещения, назначенным по умолчанию для апплета, является:

а) `GridLayout`

б) `BoxLayout`

в) `BorderLayout`

г) `FlowLayout`

12.6. HTML — это аббревиатура от:

а) Hyperspace Marketing for Mobile Links — гиперпространственный маркетинг для мобильных связей

б) HyperText Markup Language — язык разметки гипертекста

в) How To Make Links — как делать ссылки

г) Hyperlinks, Tables, Movies and Lists — гиперссылки, таблицы, кинофильмы и списки

12.7. Что является новшеством в предоставляемом Java способе работы с апплетами?

а) для описания этого способа нужно очень много книг

б) вся работа выполняется на хосте

в) возможность зарабатывать больше денег, используя средства навигации по Web

г) генная инженерия, позволяющая уменьшить некоторые вещи

12.8. Если код апплета содержит оператор `System.out.println();` и обеспечивает вывод строк в консольное окно, то где можно обнаружить этот вывод?

а) при печати на принтере

б) в окне командной строки

в) вывод нигде не будет показан

г) в консольном окне Java, отображение которого предусмотрено в броузере

12.9. Какой из следующих ответов лучше описывает преимущества, предоставляемые апплету, который хранится на удаленной машине и загружается через броузер?

а) возможность обратного соединения с хостом и создания всплывающего окна

б) возможность доступа к локальным файлам, выполнения другой программы, вызова метода `exit` и создания всплывающего окна

в) возможность выяснения имени и паролей локальной машины, форматирования жесткого диска и загрузки библиотеки Java

г) никаких преимуществ

12.10. Что представляют собой наблюдатели в терминологии Java?

а) это предоставляемые языком Java объекты, отслеживающие, какой объем изображения загружен, позволяя апплету выполнять дальнейший код

б) это предоставляемые языком Java объекты, которые отслеживают, какой объем изображения загружен, задерживая выполнение апплета до тех пор, пока изображение не будет полностью загружено и готово к выводу на экран

в) методы Java, которые могут быть вызваны для обработки удаленных событий

г) методы Java, которые автоматически вызываются при выполнении апплета на удаленной машине, когда вызывается соответствующий локальный слушатель

Задачи

12.1. **Добавление деревьев.** Проект по охране окружающей среды находится в начальной стадии и нуждается в значительном расширении. Необходимо предоставить посетителям узла возможность добавлять новые фотографии. Если вы будете посещать Web-узел *Java Gently*, то сможете пользоваться новыми версиями системы, по мере того, как они станут доступными. Одним из усовершенствований, которые вы можете сделать сами, является модификация списка деревьев. Для этого измените только HTML-файлы.

12.2. **Руководство пользователя к программе "Касса супермаркета".** При работе с программой "Касса супермаркета" не сразу можно догадаться, что и в какой последовательности следует делать. Создайте HTML-страницу, на которой слева располагается инструкция по пользованию программой, а справа — окно апплета.

12.3. **Заголовок газеты "Savanna News".** Если у вас есть сканер, отсканируйте заголовок ("шапку") солидной газеты и внедрите его в HTML-страницу, которая вызывает апплет "Розыгрыш". Можете создать большой заголовок, используя интересный шрифт. Ниже окна апплета разместите правила, представленные мелким шрифтом.

12.4. **Много гномов.** Добавьте еще несколько изображений гномов в апплет "Поймать гнома". Пусть каждое из них перемещается (выводится на экран) с различной частотой.

12.5. **Обновление числовых значений букв.** Создайте отдельный апплет, в котором использовался бы пароль. Он может быть выбран из меню или кнопкой в HTML-странице "Розыгрыш". Это должно дать возможность кому-нибудь из персонала газеты интерактивно изменять числовые значения букв.

12.6. **Апплет с программой "Кофейный магазин Нельсона".** Используйте результаты выполнения задачи 11.5 и преобразуйте GUI-версию программы "Кофейный магазин Нельсона" в апплет и HTML-документ.

12.7. **Интерактивные курсы обмена валют.** Усовершенствуйте программу из "Практикума 7", создав апплет с интерфейсом, пригодным для использования в пунктах обмена валют.

12.8. **Изменение цен.** Цены на овощи и фрукты изменяются довольно часто. Поэтому в GUI-программу "Касса супермаркета", в область, где находится кнопка **Закрыть**, добавьте новую кнопку с названием **Новая цена**. После нажатия этой кнопки должно появиться окно со списком продуктов и их текущими ценами. Пользователь должен иметь возможность ввести в этом окне новую цену для любого продукта. Эти цены должны вступить в силу немедленно после закрытия окна. Добавьте в программу необходимость ввода пароля, чтобы предотвратить свободный доступ к этому окну.

12.9. **Домашние "любимцы" теряются и находятся.** Базируясь на системе бирок, созданной для Ассоциации ветеринаров Саванны ("Практикум 4" из главы 9), в газете "*Savanna News*" захотели создать электронную версию своей популярной колонки "Потери и находки". Главная мысль заключается в том, что владельцы потерявшихся животных смогут предоставить (ввести) дополнительную информацию, а также фотографию своего любимца. Люди, которые нашли потерявшееся животное, могут ввести содержащуюся в бирке информацию и получить остальные сведения. Разработайте и реализуйте такую программу, применяя апплеты и Web-страницы. Изображения животных доступны для использования на Web-странице "*Java Gently*" в рубрике "*Other Materials*" ("Другие материалы").

12.10. **Апплет, имитирующий работу сотового телефона.** GUI-программу для сотового телефона, которую вы разрабатывали в задаче 10.3, преобразуйте в апплет. Добавьте обработчики событий, чтобы нажатие клавиш приводило к формированию сообщения в консольном окне Java. В главе 14 мы рассмотрим, как сделать, чтобы один сотовый телефон "общался" с другим.

ГЛАВА 13



Работа с несколькими подпроцессами

13.1. Работа с несколькими подпроцессами. Введение

Исходя из терминов, применяющихся для описания работы операционной системы, выполняющуюся программу называют *процессом*. В любой момент времени операционная система может выполнять несколько процессов для различных пользователей. Не все из них должны быть активными, они могут находиться в состоянии ожидания своей доли рабочего времени процессора, который поочередно выполняет несколько процессов, или могут ожидать некоторой информации, например той, которую должен ввести пользователь.

Точно так же процесс можно разбить на отдельно выполняемые *подпроцессы* (threads). Программа, имеющая такую структуру, называется *программой, работающей с несколькими подпроцессами* (multithreaded). Каждый подпроцесс выглядит и выполняется так, как если бы он был единственным. Он может взаимодействовать с другими подпроцессами этого же процесса, но при разработке программы нужно обеспечить очень точный контроль значений совместно используемых переменных.

Операционная система должна распределять рабочее время процессора между подпроцессами точно так же, как она распределяет его между процессами. Наиболее целесообразным является выделение для каждого подпроцесса определенного промежутка времени, по истечении которого данный подпроцесс приостанавливает работу, а другой подпроцесс, ожидающий своей очереди, получает возможность выполнить свои операторы (сколько успеет). Менее привлекательным является распределение, при котором подпроцесс выполняется до тех пор, пока ему не потребуется информация извне (от другого подпроцесса или пользователя). Проблема реализации этого способа как раз и состоит в том, что один-единственный подпроцесс не может эффективно распорядиться всей мощностью процессора. В настоящее время в большинстве систем используется первый метод распределения.

Для чего нужны подпроцессы

Рассмотрим две ситуации, которые демонстрируют значение подпроцессов в программах.

1. **Пользовательский интерфейс.** Если апплет или графическое приложение занято выведением на экран изображения или текста, то у пользователя может появиться желание остановить это действие. При наличии кнопки **Закрыть** мы можем ею воспользоваться, но программа обнаружит факт нажатия кнопки только тогда, когда достигнет пассивного состояния, т.е. завершит вычисление или выведение чего-либо на экран и будет ждать поступления событий. Однако если события,

связанные с кнопкой, обрабатываются отдельным подпроцессом, то регулярно (через назначенный промежуток времени) у последнего появляется возможность отреагировать на нажатие, даже если программа продолжает выполнять свои задачи. На рис. 13.1 проиллюстрирована работа программы, в которой есть два таких подпроцесса, причем показаны не только этапы выполнения подпроцессов на протяжении некоторого промежутка времени, но и их взаимодействие.

2. **Несколько экземпляров.** Иногда пользователю может потребоваться, чтобы одновременно были доступны идентичные копии изображения, несколько окон или различные результаты вычислений. Например, комплект выводимых на экран компонентов изображения светофора из примера 10.4 может быть продублирован, и мы сможем видеть два или три таких комплекта, работающих одновременно. Каждый будет обрабатываться отдельным подпроцессом. В простом случае подпроцессы являются точными копиями друг друга и выполняются независимо. Java позволяет подпроцессам взаимодействовать друг с другом и синхронизировать свою деятельность, о чем речь пойдет в разделе 13.3.

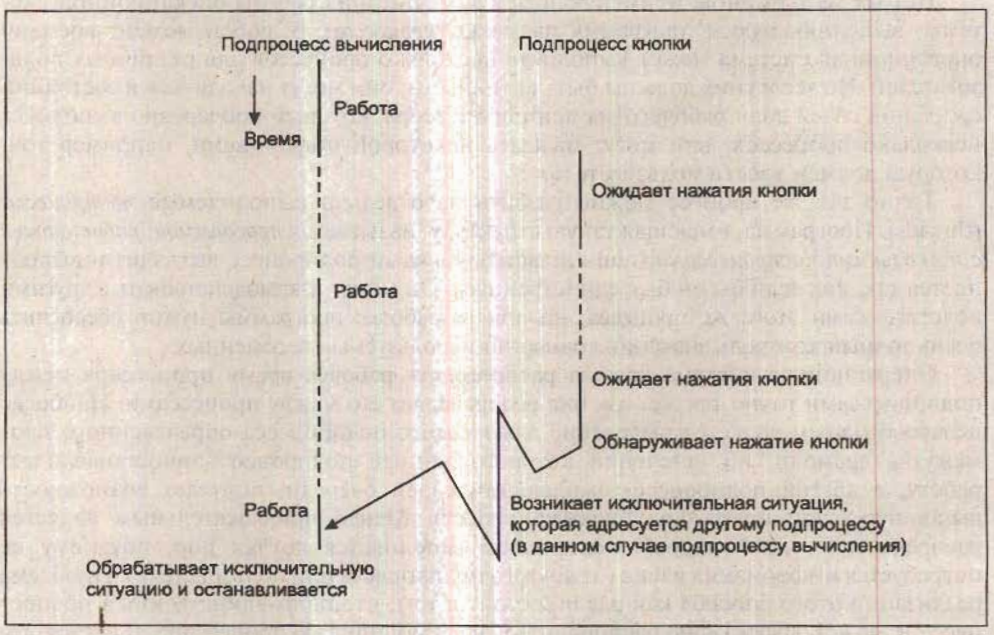


Рис. 13.1. Работа с несколькими подпроцессами в пользовательском интерфейсе

Для пользователя присутствие нескольких подпроцессов должно быть незаметным, следовательно, программисту нужно точно определить время выполнения и завершения каждого создаваемого подпроцесса. В течение времени жизни процесса подпроцессы могут создаваться и уничтожаться. Они являются средством для эффективного использования мощности и памяти процессора.

13.2. Назначение подпроцессов

Для того чтобы "разбить" программу на подпроцессы, мы вначале должны определить, какие методы могут выполняться независимо. Обычно они уже находятся в классе, поэтому можно сказать, что именно класс станет подпроцессом и, возможно, в нескольких своих реализациях.

Класс `Thread` находится в пакете `java.lang`, и это означает, что он всегда доступен и не требует специального импортирования. Любой класс, который "хочет" быть классом-подпроцессом, должен наследовать класс `Thread`. Класс `Thread` реализует интерфейс `Runnable`, который имеет один метод — `run`. Следовательно, вероятный класс-подпроцесс должен реализовать метод `run`. Обычно исполнимая часть класса помещается в метод `run` (конечно же, позволяя этому методу вызывать другие необходимые методы).

Метод `main` по умолчанию является подпроцессом, и из него или из другого активного метода, например из метода `init`, применяющегося в классе `Applet`, начинают свое выполнение другие подпроцессы. Другими словами, подпроцессы (и мы увидим это в последующих примерах) могут быть разбиты на собственные подпроцессы. Давайте рассмотрим простой пример с использованием подпроцессов, а затем вновь вернемся к теории.

Пример 13.1. Рисование точек

Задача. Продемонстрировать работу подпроцессов.

Решение. Мы выберем простой пример, в котором в окно апплета выводятся разноцветные точки. В программе будет три одновременно выполняющихся подпроцесса, каждый из которых чертит точки своим цветом.

Программа. Программа написана как апплет, в котором метод `init` создает экземпляры трех подпроцессов и инициирует их выполнение. Эти подпроцессы являются экземплярами внутреннего класса, который наследует класс `Thread` и называется `Spots`. В каждом из подпроцессов `Spots` фиксируется цвет выводимой точки, после чего запускается в работу цикл метода `run`. Цикл вызывает метод `draw`, для того чтобы отобразить одну точку, затем выжидает установленное время (дословный перевод названия метода sleep — "спит"), после чего отображает следующую точку.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class SpotTest extends Applet {

   /* Рисование точек       автор Дж.М. Бишоп, август 2000 г.
    * = = = = = = = = =
    *
    * Чертит точки различного цвета.
    *
    * Демонстрирует использование простых подпроцессов.
    */

  int mx, my;
  int radius = 10;
```

```
  int boardSize = 200;
  int change;

  public void init() {
    boardSize = getSize().width - 1;
    change = boardSize-radius;

    // Создает и инициирует выполнение трех подпроцессов.
    new Spots(Color.red).start();
    new Spots(Color.blue).start();
    new Spots(Color.green).start();
  }


class Spots extends Thread {

   Color colour;
    // Конструктор фиксирует цвет выводимой точки
    // для этого подпроцесса.
    Spots(Color c) {
      colour = c;
    }

    // Очень простой метод run.
    public void run () {
      while (true) {
        draw();
        try {
          sleep (500); // миллисекунды
        }
        catch (InterruptedException e) {
        }
      }
    }

  public void draw() {
    Graphics g = getGraphics();
    g.setColor(colour);
    // Вычисляет новые координаты для точки
    // и чертит ее.
    mx = (int)(Math.random()*1000) % change;
    my = (int)(Math.random()*1000) % change;
    g.fillOval(mx, my, radius, radius);
  }
 }
}
```

Тестирование. Наблюдая за выполнением программы*, мы можем подумать, что все точки появляются одновременно (рис. 13.2). На самом же деле, хотя подпроцессы и выполняются одновременно, они, фактически, делят между собой процессор и выполняются по очереди. Просто они делают это так быстро, что мы не успеваем этого заметить.

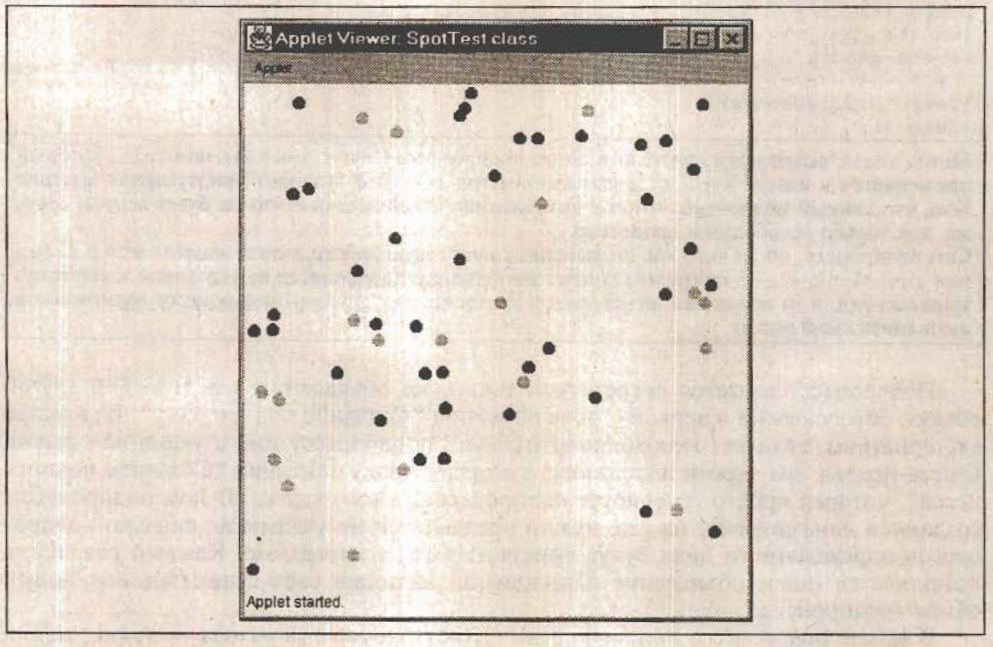


Рис. 13.2. Выход апплета `SpotTest`

Класс Thread

После неформального ознакомления с подпроцессами рассмотрим класс `Thread` более детально. Создание подпроцесса может быть выполнено двумя способами.

Создание подпроцесса
ThreadClass threadid = **new** *ThreadClass* (*parameters*); **new** *ThreadClass* (*parameters*)
В первой строке описания объект-подпроцесс создается с идентификатором `threadid`. Во второй строке объект создается без идентификатора.

---

* Выполнение программы-апплета требует создания простейшего html-файла, текст которого приводится ниже:

```
<html>
<head>
 <title>Апплет Рисование точек</title>
</head>
   <body>
        <applet code="SpotTest.class" width=400 height=400></applet>
   </body>
 </html>
```

— *Прим. ред.*

Среди множества других класс `Thread` имеет следующие важные методы.

Методы класса `Thread`

```
методы экземпляра
- - - - - - - - - - - -
start ();
run ();
методы класса
- - - - - - - -
sleep (milliseconds);
yield ();
```

Метод `start` вызывается из уже активного подпроцесса (такого как `main` или `init`, который применяется в классе `Applet`) и вызывает метод `run`. Это "говорит" виртуальной машине Java, что данный подпроцесс готов к выполнению своей задачи и что он будет вызван сразу же, как только освободится процессор.
Сам подпроцесс, после того как он выполнит некоторую работу, может вызвать метод `sleep` или `run`. Метод `sleep` приостанавливает выполнение подпроцесса на указанное количество миллисекунд, в то время как метод `yield` предоставляет другому подпроцессу возможность выполнить свою задачу.

Подпроцесс создается посредством такого же объявления, как и любой другой объект. Это показано в первой строке описания "Создание подпроцесса". В качестве альтернативы, если нет необходимости давать подпроцессу имя с указанием имени класса-предка, мы можем использовать вторую строку описания "Создание подпроцесса", которая просто генерирует подпроцесс данного класса. В Java подпроцессы создаются динамически, нам не нужно предварительно указывать, сколько подпроцессов определенного вида будут присутствовать в программе. Каждый раз, когда выполняется новое объявление подпроцесса, начинает свое существование новый объект-подпроцесс.

В работе подпроцесса активную роль играют три важных метода — `start`, `run` и `sleep`. После того как подпроцесс был создан посредством своего конструктора, созидающий метод вызывает метод `start`, который определен в классе `Thread`. Метод `start`, в свою очередь, вызывает метод `run`, что приводит к внесению этого подпроцесса в список операционной системы, где перечисляются подпроцессы, ожидающие своей очереди на выполнение. Когда наступит очередь данного подпроцесса, ему будет передан контроль над процессором, и он немедленно начнет выполняться.

Метод `run` обычно состоит из цикла, который продолжается до тех пор, пока не будет выполнено какое-нибудь условие, после чего метод `run` завершит свою работу естественным образом, в результате чего подпроцесс прекратит свое существование.

Методы `sleep` и `yield` являются методами класса. Выполнение любого из них приводит к тому, что подпроцесс оставляет (передает другому подпроцессу) контроль над процессором. При выполнении метода `sleep` подпроцесс выжидает указанное время, по истечении которого он снова будет готов к выполнению. Если в операционной системе предусмотрено распределение времени использования процессора, то метод `yield` не потребуется, однако если такое совместное использование невозможно, то вызов данного метода приостановит выполняющийся подпроцесс и передаст контроль над процессором другим "ожидающим".

Java также поддерживает концепцию приоритетов для подпроцессов. Для них могут быть указаны значения, находящиеся в диапазоне от 1 (низкий приоритет) до 10 (высокий приоритет). Эти значения являются директивой для операционной системы, указывающей, какой подпроцесс должен выполняться следующим, если возможны варианты. Подпроцессы с наиболее высоким приоритетом всегда будут выполняться первыми.

Подпроцессы в новых версиях прежних программ

Давайте попытаемся изменить программу таким образом, чтобы в ней применялись подпроцессы. Возьмем уже знакомый нам пример 11.2, предусматривающий настройку светофора. Мы бы хотели, чтобы цвета светофора загорались по очереди в назначенной последовательности красный-желтый-зеленый, но чтобы кнопки и раскрывающиеся списки, находящиеся внизу экрана, оставались активными. В первоначальной программе конструктор выводит на экран заголовки и кнопки, а затем создает объект класса `Canvas` (холст), чей метод `paint` непосредственно чертит изображение светофора.

Вначале нам нужно преобразовать класс `LightCanvas` в подпроцесс, а именно:

- расширить класс `Thread`;
- определить метод `run`, который бы периодически выполнялся в цикле `while`.

Но класс может наследовать только один другой класс, поэтому мы должны применить следующую стратегию.

1. Укажем, что класс `LightCanvas` будет расширять класс `Thread`.
2. Передадим экземпляр холста новому конструктору класса `LightCanvas`.
3. Сохраним локально ссылку на экземпляр холста из подпроцесса.
4. Вызовем метод `getGraphics` для выведения холста в окне, с которым работает подпроцесс.

Метод `paint` вызывается автоматически через пакет `awt`. Теперь у нас в программе должны присутствовать независимые процессы черчения, выполняющиеся в единственном окне. Выполнение метода `run` должно привести к полному завершению процесса черчения. Описание наиболее простого метода `run`, который будет использоваться в примере 13.2, приводится ниже.

Определение для подпроцесса метода `run`

```
public void run () {
  while (!ended) {
    draw (); // или некоторые другие операторы
    try {
      sleep (500);
    }
    catch (InterruptedException e) {}
  }
}
```

Метод `run` вызывается, когда стартует подпроцесс.
Значение булевой переменной `ended` может устанавливаться (назначаться) другим подпроцессом или главной программой, чтобы уничтожить подпроцесс.
При каждом повторении метода `run` выполняются некоторые операторы (например, `draw`), а затем посредством вызова метода `sleep` подпроцесс опять передает контроль JVM, в результате чего у других подпроцессов появляется шанс выполнить свои методы (операторы). Подпроцесс может вновь возвратить себе контроль над процессором и возобновить выполнение своих действий после перехвата объекта-исключения `InterruptedException`.

В нашем случае подпроцесс повторно вычерчивает изображение и ждет не менее 500 миллисекунд, прежде чем сделать это еще раз. Для того чтобы возобновить работу подпроцесса, после того как пройдет положенное время и наступит его очередь, система должна сгенерировать объект-исключение `InterruptedException`.

Теперь переименованный класс `LightsCanvas` будет выглядеть следующим образом:

```
class SetOfLights extends Thread {

  SetOfLights (Canvas c) {
    Area = c;
  }
  public void run () {
    while (true) {
      draw ();
      try {sleep (500); }
         catch (InterruptedException e) {}
    }
  }

  void draw () {
    Graphics g = area.getGraphics();
    … как и прежде, содержание метода paint
  }
}
private Canvas area;
```

Для создания и запуска подпроцесса в работу мы вызываем его конструктор, а затем метод `start`.

```
SetOfLights lights = new SetOfLights (area);
lights.start();
```

Остановка (уничтожение) подпроцесса должна быть выполнена корректно. Имеет смысл воспользоваться булевой переменной, которая будет проверяться в цикле `while` метода `run`. Когда от пользователя придет сигнал о том, что программа должна завершиться, этой булевой переменной будет присвоено значение `false`, после чего подпроцесс остановит выполнение цикла `while` и прекратит свое существование.

В примере 13.2 приводится полный текст программы, имитирующей работу светофора.

Пример 13.2. Программа "Светофор" с мигающими огнями

Задача. Статическое представление светофора, данное в примере 11.2, должно быть усовершенсвовано и показывать действительно мигающие огни. Кроме того, мы бы хотели, чтобы одновременно можно было видеть несколько работающих комплектов огней (т.е. светофоров) — в таком случае можно будет сравнивать изменения.

Решение. Для создания программы, в которой бы отображались мигающие огни, нужно использовать класс `Graphics` пакета `awt`. Вывести на экран несколько светофоров можно с помощью подпроцессов. Как упоминалось выше, мы создаем для светофора новый класс, указываем, что он расширяет класс `Thread`, и "заставляем" его повторно вычерчивать полный комплект элементов изображения светофора после нажатия кнопки **Новый светофор**. Время, указанное для метода `sleep`, будет соответствовать продолжительности горения одного света, когда остальные будут отключены (выведены серым цветом). Используя оператор `switch`, мы можем реализовать требуемую последовательность — красный-желтый-зеленый.

Разработка класса. Поскольку мы используем подпроцессы, отношения между классами в `awt` и теми, которые мы определили, будут немного различаться. Новая

схема (сравните с приведенной на рис. 11.4) показана на рис. 13.3. Из нее исключены часть кнопок и некоторые другие компоненты, не являющиеся необходимыми для объяснения сути работы программы.

Имеющиеся различия мы, в принципе, уже объяснили. Класс `SetOfLights` наследует класс `Thread`, поэтому он не может одновременно наследовать и класс `Canvas`. Вместо этого холст, называемый `area`, передается как параметр из метода `actionPerformed` класса `Traffic2` в конструктор класса `SetOfLights`. В программе показано, что объекты `lights` (изображения светофоров) создаются в методе `actionPerformed`, поскольку именно здесь в ответ на событие, сгенерированное кнопкой **Новый светофор**, создаются их экземпляры. Ссылка на этот холст используется методом `draw` для успешного доступа к методам класса `Graphics`.

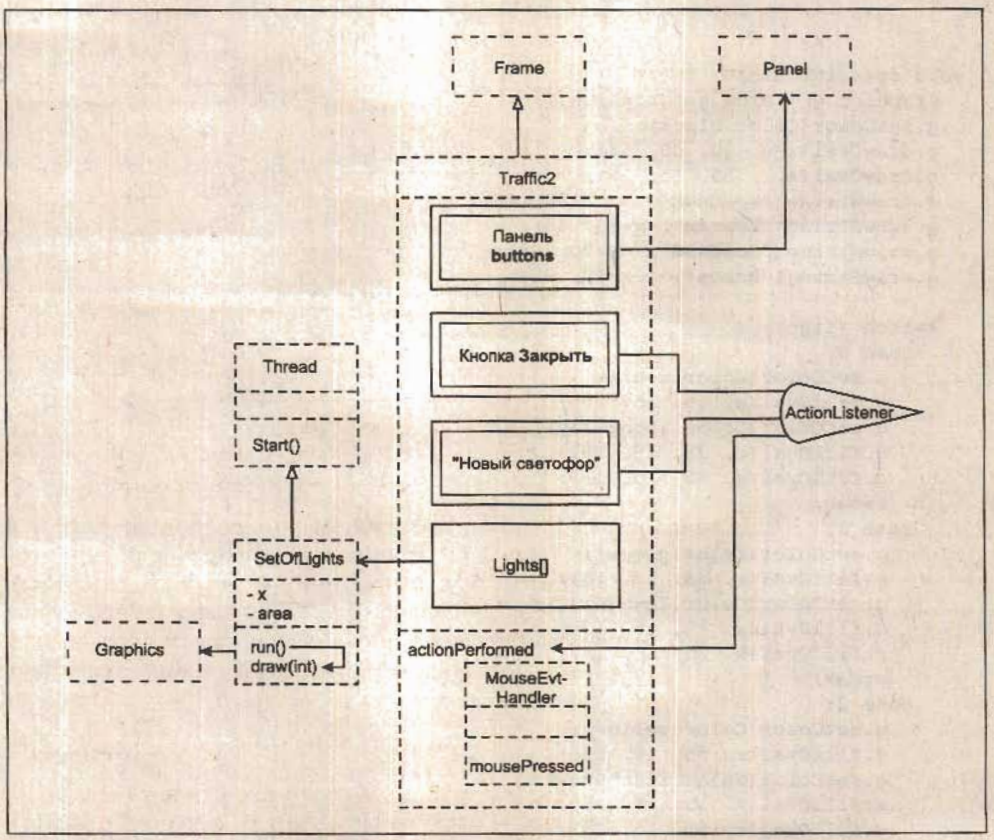


Рис. 13.3. Схема программы с мигающими огнями светофора

Программа. Класс `SetOfLights` выглядит следующим образом.

```
import java.awt.*;
class SetOfLights extends Thread {

  private int red = 0;
  private Canvas area;
  private int x;
```

```
  public boolean alive = true;

  public SetOfLights(Canvas c, int x) {
    area = c;
    this.x = x;
  }
  public void run() {
    while (alive) {
      for (int light = 0; light < 3; light++) {
        draw(light);
        try { sleep(500);}
        catch (InterruptedException e) { }
      }
    }
  }

  void draw(int light) {
    Graphics g = area.getGraphics();
    g.setColor(Color.black);
    g.drawOval(x-8, 10, 30, 68);
    g.drawOval(x,   85, 15, 15);
    g.drawString("Красный",    x-90, 28);
    g.drawString("Желтый", x-90, 48);
    g.drawString("Зеленый",  x-90, 68);
    g.drawString("Идите",    x-90, 98);

    switch (light) {
      case 0:
        g.setColor(Color.red);
        g.fillOval(x, 15, 15, 15);
        g.setColor(Color.lightGray);
        g.fillOval(x, 35, 15, 15);
        g.fillOval(x, 55, 15, 15);
        break;
      case 1:
        g.setColor(Color.green);
        g.fillOval(x, 55, 15, 15);
        g.setColor(Color.lightGray);
        g.fillOval(x, 15, 15, 15);
        g.fillOval(x, 35, 15, 15);
        break;
      case 2:
        g.setColor(Color.yellow);
        g.fillOval(x, 35, 15, 15);
        g.setColor(Color.lightGray);
        g.fillOval(x, 15, 15, 15);
        g.fillOval(x, 55, 15, 15);
        break;
    }

  }
}
```

Метод `run` имеет собственный внутренний цикл, выполнение которого охватывает три фазы работы светофора. В зависимости от того, какой свет должен гореть, оператор `switch` окрашивает один круг определенным цветом, а для двух других

кругов назначает серый цвет. Далее следует программа, которая активизирует эти подпроцессы*.

```
import java.awt.*;
import java.awt.event.*;

public class Traffic2 extends Frame
       implements ActionListener {

   /* Третья версия программы "Светофор"
    *                                  Автор Дж.М. Бишоп, октябрь 1996 г.
    *                         Java 1.1 версия Т. Абботта, октябрь 1997 г.
    *              , расширена и исправлена  Дж.М. Бишоп, октябрь 1997 г.
    *                              обновлена Дж.М. Бишоп, август 2000 г.
    * Дает возможность работать одновременно
    * нескольким светофорам.
    *
    * Демонстрирует использование подпроцессов и графики.
    */

  private Canvas area;
  private int lightsPosition = 105;
  private static final int lightsWidth = 150;
  private SetOfLights[] lights = new SetOfLights[3];
  private int nLights = 0, setWanted = 0;
  private Choice colours;
  private Button newSetButton;
  private Button walkButton;
  private Button closeButton;
  private TextField duration;

  public Traffic2() {
    setTitle("Программа "Светофор" версия 2");
    add("North",
         new Label("Симулятор светофора ", Label.CENTER));
    area = new Canvas();
      area.addMouseListener(new MouseEvtHandler());
      add("Center",area);

    Panel buttons = new Panel();
      newSetButton = new Button("Новый светофор");
        newSetButton.addActionListener(this);
        buttons.add(newSetButton);
      colours = new Choice ();
        colours.addItem("Красный");
        colours.addItem("Желтый");
        colours.addItem("Зеленый");
        colours.addItem("Идите");
        buttons.add(colours);

      buttons.add(new Label("Длительность"));

      duration = new TextField("", 3);
        duration.setEditable(true);
```

* Чтобы запустить программу, нужно весь код класса `SetOfLights` поместить перед методом `main` главной программы (класса `Traffic2`) или же поочередно выполнить компиляцию обоих классов. — *Прим. ред.*

```
        duration.addActionListener(this);
        buttons.add(duration);

      walkButton = new Button("Идите");
        // еще нет никаких действий
        buttons.add(walkButton);

      closeButton = new Button("Закрыть");
        closeButton.addActionListener(this);
        buttons.add(closeButton);
    add("South", buttons);

    // Создание окна (назначение размеров и т.д.).
    setSize(350, 210);
    setVisible(true);
    addWindowListener(new WindowAdapter () {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });

  }

  class MouseEvtHandler extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
      int n = e.getX() / lightsWidth;
      if (n < nLights)
      setWanted = n;
    }
  }

  public void actionPerformed(ActionEvent e) {
    Object event = e.getSource();
    if (event == newSetButton) {
      lights[nLights] = new SetOfLights(area, lightsPosition);
      lights[nLights].start();
      lightsPosition += lightsWidth;
      nLights++;
      if (nLights == 3)
      newSetButton.setEnabled(false);
    } else if (event == closeButton) {
      for (int i = 0; i<nLights; i++)
        lights[i].alive = false;
      setVisible(false);
      dispose();
      System.exit(0);
    }
  }

  public static void main(String[] args) {
    new Traffic2();
  }
}
```

На панели кнопок появился дополнительный компонент — кнопка **Новый светофор**. При ее нажатии стартует подпроцесс, выводящий на экран еще один комплект изображений, составляющих светофор. Чтобы светофоры располагались

вдоль окна, мы предоставили новую стартовую позицию x, значение которой возрастает на 150 пикселей при отображении каждого нового светофора.

Тестирование. На рис. 13.4. приведена копия экрана с изображением трех светофоров, работающих в различных фазах (горит разный свет).



Рис. 13.4. Три светофора, работающих одновременно

Коммуникации между подпроцессами

В приведенном выше примере подпроцессы являются довольно независимыми. Однако часто возникают ситуации, когда подпроцессы должны передавать информацию друг другу, например, результаты вычислений или сигналы о переходе в другой режим работы. Поскольку подпроцессы являются объектами, то для таких коммуникаций могут использоваться стандартные технологии, свойственные объектам, в том числе вызов методов и обновление незакрытых переменных. Может создаться впечатление, что при взаимодействии подпроцессов, осуществляющих коммуникации указанными способами, проблем не существует. Однако конфликты могут происходить и при этих условиях, и последствиями таких конфликтов могут стать неверные результаты работы программы. Затронутая тема будет рассмотрена в параграфе 13.4. Мы же продолжим изучение нашего примера, который имитирует работу светофора, и попытаемся показать, насколько просто обработать в программе коммуникации между подпроцессами при отсутствии конфликтов.

Пример 13.3. Светофор для пешеходов

Задача. В самом начале работы над этой программой (пример 10.5) мы упоминали, что главное ее назначение — исследовать, как измененяется продолжительность горения каждого света светофора при нажатии пешеходами кнопки **Идите**. Кнопка **Идите** находится в контрольной панели. Теперь нам хотелось бы, чтобы свет, соответствующий на изображении кнопке **Идите**, загорался каждый раз, когда на светофоре загорится красный свет. Кроме того, мы бы хотели вновь активизировать раскрывающийся список и текстовое поле **Длительность**, чтобы передавать (вводить в программу) информацию, которая бы изменяла фазы работы светофора, т.е. создать действительно работоспособную имитацию светофора.

Изменения в алгоритме. Вначале решим следующий вопрос: должна ли кнопка **Идите** использоваться для всех трех светофоров или имеет смысл изменить выводимое на экран изображение так, чтоб под каждым светофором была своя кнопка. Последний вариант

кажется более целесообразным, но его гораздо труднее реализовать, поскольку мы используем очень простой менеджер размещения `BorderLayout` и светофоры у нас находятся в центральной панели (область `"Center"`), а кнопки — в области `"South"`. Поэтому, хотя мы отдали предпочтение первому варианту, с одной кнопкой, мы посредством щелчка мышью где-нибудь в центральной области сможем выбрать светофор, к которому хотели бы применить изменения. Для этого будем использовать слушатель `MouseListener` и метод `mousePressed`.

Следующий вопрос: что должно фактически происходить после нажатия кнопки **Идите**? Мы должны будем передать информацию конкретному подпроцессу, который управляет этим светофором. Подпроцессу не нужно реагировать немедленно, поскольку, согласно требованию, он должен продолжать предусмотренный цикл до тех пор, пока не наступит очередь красного света, и только после этого должен загораться зеленым светом круг, соответствующий надписи **Идите**. Поэтому мы можем обеспечить нужную коммуникацию, внедрив в подпроцесс светофора булеву переменную `walk`, которая будет получать значение `true` после нажатия кнопки **Идите**.

А как мы узнаем, какому подпроцессу передавать информацию? Каждый подпроцесс в главной программе должен быть зафиксирован под конкретным именем, и наиболее простым решением этого требования будет определение массива подпроцессов. При нажатии кнопки мыши по координатам ее курсора в центральной области холста мы можем определить, с каким светофором необходимо иметь дело, и следующее нажатие кнопки **Идите** будет относиться именно к этому подпроцессу массива.

Раньше объекты, представляющие собой изображения светофора, создавались внутри метода `actionPerformed`, и имя `lights` использовалось каждый раз, когда нужно было отобразить новый светофор. Теперь, поскольку имена подпроцессов нужно контролировать в течение выполнения всей программы, мы объявляем массив подпроцессов как закрытый. Каждый новый подпроцесс массива активизируется посредством вызова метода `start`.

Заключительным усовершенствованием программы будет использование раскрывающегося списка и текстового поля для изменения времени, в течение которого свет будет гореть. Реализовать его довольно просто. Вместо постоянного значения длительности "спящего" режима (активизации метода `sleep`), равной 500 миллисекундам, мы дадим возможность свету "спать" `time[light]` миллисекунд. Для этого в методе `actionPerformed` при необходимости изменяются значения массива `time`.

На этот раз мы не приводим схему алгоритма, а предоставляем читателю возможность начертить ее самостоятельно, чтобы лучше понять программу.

Программа. В заключительной программе, имитирующей работу светофора, удалены метки для соответствующих цветов, поскольку они заменены значениями текущей длительности горения данного света. Для компилирования этой программы нужно выполнить то же самое, что и в предыдущей версии, — поместить код класса `SetOfLights` в главную программу.

```
import java.awt.*;
import java.awt.event.*;

public class Traffic3 extends Frame
       implements ActionListener, ItemListener {

   /* Четвертая программа "Светофор"
    *                                    Автор Дж.М. Бишоп, октябрь 1996 г.
    *                              Java 1.1 версия Т. Абботта, октябрь 1997 г.
```

```
 *                расширена и исправлена Дж.М. Бишоп, октябрь 1997 г.
 *                                улучшена Дж.М. Бишоп, август 2000 г.
 * Дает возможность вводить в текстовом поле "Длительность"
 * различное время включенного состояния для всех цветов
 * светофора.
 */

private Canvas area;
private int lightsPosition = 105;
private static final int lightsWidth = 150;
private SetOfLights[] lights = new SetOfLights[3];
private int nLights = 0, setWanted = 0;
private TextField duration;
private Choice colours;
private int light;
private Button newSetButton;
private Button walkButton;
private Button closeButton;

public Traffic3() {
  setTitle("Программа \"Светофор\" версия 3");

  add("North",new Label
     ("Симулятор светофора",Label.CENTER));

  area = new Canvas();
  area.addMouseListener(new MouseEvtHandler());
  add("Center",area);

  Panel buttons = new Panel();
    newSetButton = new Button("Новый светофор");
      newSetButton.addActionListener(this);
      buttons.add(newSetButton);
    colours = new Choice ();
      colours.addItem("Красный");
      colours.addItem("Зеленый");
      colours.addItem("Желтый");
      colours.addItem("Идите");
      colours.addItemListener(this);
      light = 0;
      buttons.add(colours);

    buttons.add(new Label("Длительность"));

    duration = new TextField("", 4);
      duration.addActionListener(this);
      buttons.add(duration);
    walkButton = new Button("Идите");
      walkButton.addActionListener(this);
      buttons.add(walkButton);
    closeButton = new Button("Закрыть");
      closeButton.addActionListener(this);
      buttons.add(closeButton);
  add("South",buttons);
```

```
    // Создание окна (определение размеров и т.д.).
    setSize(450, 210);
    setVisible(true);
    addWindowListener(new WindowAdapter () {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });

  }

  public void itemStateChanged(ItemEvent e) {
    String s = (String) e.getItem();
    if (s=="Красный ")    {light = 0;} else
    if (s=="Зеленый")  {light = 1;} else
    if (s=="Желтый") {light = 2;} else
    if (s=="Идите")    {light = 3;}
  }
  class MouseEvtHandler extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
      int n = e.getX() / lightsWidth;
      if (n < nLights)
      setWanted = n;
    }
  }

  public void actionPerformed(ActionEvent e) {
    Object event = e.getSource();
    if (event == newSetButton) {
      lights[nLights] = new SetOfLights(area, lightsPosition);
      lights[nLights].start();
      lightsPosition += lightsWidth;
      nLights++;
      if (nLights == 3)
      newSetButton.setEnabled(false);
    } else if (event == walkButton) {
      lights[setWanted].walk = true;
    } else if (event == duration) {
      lights[setWanted].time[light]=
        Integer.parseInt(duration.getText());
    } else if (event == closeButton) {
      for (int i = 0; i<nLights; i++)
        lights[i].alive = false;
      setVisible(false);
      dispose();
      System.exit(0);
    }
  }

  public static void main(String[] args) {
    new Traffic3();
  }
}
```

При нажатии кнопки **Идите** переменной `walk` в корректном подпроцессе присваивается значение `true`. Нужный подпроцесс предварительно выбирается щелчком мыши возле соответствующего изображения светофора, т.е. значение координаты `x` курсора мыши делится на ширину холста, которая постоянна и равна 150 пикселям.

Логика действий, относящихся к светофору, такова: каждый раз, когда должен включиться красный свет, нужно проверить, присвоено ли переменной `walk` значение `true`. Если да, то подпроцесс входит в отдельный цикл, состоящий из десяти проходов (повторений), во время которых красный свет остается включенным, а свет (круг), соответствующий кнопке **Идите**, окрашивается то в зеленый, то в серый цвет. Используемый для этой цели метод `run` приведен в полном тексте класса `SetOfLights`.

```
import java.awt.*;
class SetOfLights extends Thread {

  private int red = 0;
  private Canvas area;
  private int x;
  private int light;
  boolean alive = true;

  // открытые переменные
  boolean walk = false;
  boolean walkOn = false;
  int time [] = {500, 500, 500, 500};

  public SetOfLights(Canvas c, int x) {
    area = c;
    this.x = x;
  }

  public void run() {
    while (alive) {
      for (int light = 0; light < 3; light++) {
        if (light == red & walk) {
          walkOn = false;
          for (int i = 0; i < 11; i++) {
            draw(light);
            try { sleep(time[3]);}
            catch (InterruptedException e) { }
            walkOn = !walkOn;
          }
          walk = false;
        } else {
          draw(light);
          try { sleep(time[light]); }
          catch (InterruptedException e) { }
        }
      }
    }
  }

  void draw(int light) {
    Graphics g = area.getGraphics();
```

```
        g.setColor(Color.black);
        g.drawOval(x-8, 10, 30, 68);
        g.setColor(Color.cyan);
        g.fillRect(x-90,10,70,100);
        g.setColor(Color.black);
        g.drawString(""+time[0], x-70, 28);
        g.drawString(""+time[2], x-70, 48);
        g.drawString(""+time[1], x-70, 68);
        g.drawString(""+time[3], x-70, 98);

        switch (light) {
          case 0:
            g.setColor(Color.red);
            g.fillOval(x, 15, 15, 15);
            g.setColor(Color.lightGray);
            g.fillOval(x, 35, 15, 15);
            g.fillOval(x, 55, 15, 15);
            break;
          case 1:
            g.setColor(Color.green);
            g.fillOval(x, 55, 15, 15);
            g.setColor(Color.lightGray);
            g.fillOval(x, 15, 15, 15);
            g.fillOval(x, 35, 15, 15);
            break;
          case 2:
            g.setColor(Color.yellow);
            g.fillOval(x, 35, 15, 15);
            g.setColor(Color.lightGray);
            g.fillOval(x, 15, 15, 15);
            g.fillOval(x, 55, 15, 15);
            break;
        }

        if (light == red & walk) {
          if (walkOn)
            g.setColor(Color.green);
          else
            g.setColor(Color.white);
          g.fillOval(x+1, 85, 14, 14);
        } else {
          g.setColor(Color.black);
          g.drawOval(x, 85, 15, 15);
        }
      }
    }
```

Переменная `walkOn` принимает попеременно то значение `true`, то значение `false`, контролируя включение света, соответствующего кнопке **Идите**. После десяти повторений переменной `walk` присваивается значение `false`, и работа светофора сможет перейти в следующую обычную фазу, при которой поочередно загораются три света светофора. Соответствующие данному алгоритму операторы находятся в конце метода `draw`.



Рис. 13.5. Заключительная программа, имитирующая работу светофора

Тестирование. При выполнении этой программы, кнопка **Идите** может быть нажата для каждого светофора в различный момент времени, как это показано на рис. 13.5. После того как в программу была введена возможность назначать различную продолжительность горения света, симулятор действительно стал пригоден для исследования поставленной задачи.

13.3. Синхронизация работы подпроцессов

В приведенном выше примере программа работает с каждым процессом индивидуально. Как выполняются коммуникации между подпроцессами и объектами `Traffic`, показано на рис. 13.6. Если объект `Traffic` вызывает объект `lights[0]`, то невозможно, чтобы этому процессу помешало взаимодействие между объектом `Traffic` и, скажем, объектом `lights[1]`. Однако предположим, как возможный вариант, что каждому из подпроцессов потребуется изменить одну и ту же переменную, принадлежащую классу `Traffic` (например, учитывающую количество включений желтого света). Диаграмма вызовов будет такой, как показано на рис. 13.7.

Теперь необходимо учесть следующее: подпроцессы могут вмешиваться в работу друг друга, что может привести к возникновению неверных результатов. Рассмотрим ситуацию, когда подпроцесс 1 должен увеличивать значение переменной с 52 до 53 (операция инкремента). Предположим, что он успеет выполнить только половину операции, после чего операционная система прервет его работу и передаст управление процессором подпроцессу 2. Подпроцесс 2 считает текущее значение переменной, которое все еще равно 52, увеличит его до 53 и продолжит свою работу. Через некоторое время контроль над процессором вновь вернется подпроцессу 1, у которого выполнена только половина операции инкремента. Он уже (до прерывания) считал значение переменной как 52, поэтому будет увеличивать ее (как он "полагает") до 53. Но в результате двух операций инкремента полученное значение переменной станет равным 54, а вмешательство подпроцесса 2 в данном случае свелось к тому, что подпроцесс 1 перезаписал сделанное им, подпроцессом 2, обновление.

Чтобы избежать подобных коллизий, Java предоставляет средства для подпроцессов, позволяющие синхронизировать их действия. Существует два уровня обеспечиваемой синхронизации:

- защита совместно используемых ресурсов;
- сигнализация об изменениях в состояниях между подпроцессами.

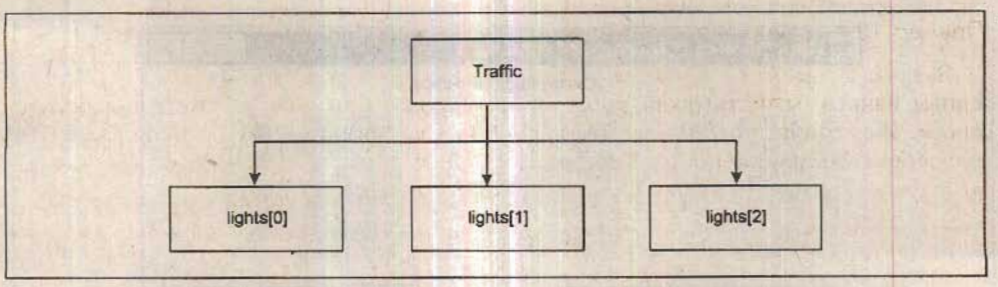


Рис. 13.6. Подпроцессы, которые не могут вмешаться в работу друг друга

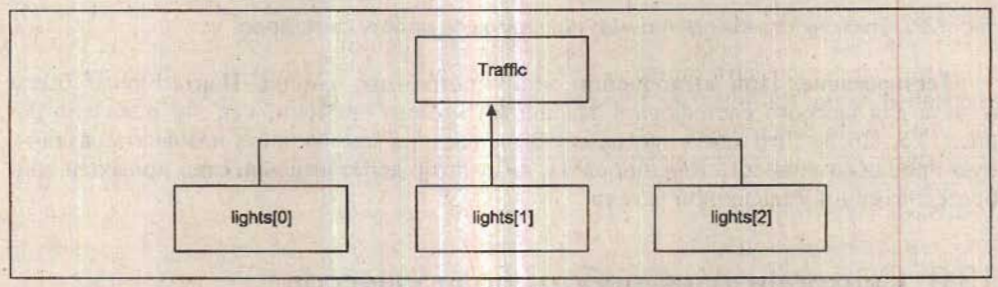


Рис. 13.7. Подпроцессы, у которых есть возможность вмешаться в работу друг друга

Защита совместно используемых ресурсов

Для защиты ресурсов, и в частности переменных, к которым могут получить доступ одновременно несколько подпроцессов, мы осуществляем обновление методами, маркированными специальным модификатором, который описан ниже.

Синхронизированный метод
synchronized *modifiers methodid (parameters)*
В каждое конкретное время этот метод может исполнять только один объект (оператор/ы только одного объекта).

К тому же Java гарантирует, что подпроцесс, получивший доступ к синхронизированному методу, закончит его до того, как любой другой подпроцесс сможет получить доступ к этому методу или любому другому синхронизированному методу данного объекта. Такие подпроцессы помещаются в очередь и ожидают, когда наступит их черед. Следовательно, подпроцесс имеет исключительный доступ к объекту из выполняемого синхронизированного метода. Можно сказать, что объект, имеющий один или более синхронизированных методов, выполняет **мониторинг** своих данных, поэтому он называется **монитором*** **(monitor)**.

Чтобы не задерживать остальные подпроцессы, синхронизированные методы должны выполнять минимальный объем действий и содержать только те операторы, которые действительно восприимчивы к вмешательству извне.

---

* Мониторы представляют собой очень важную концепцию операционных систем, предложенную Пером Бринчем Хансеном (Per Brinch Hansen) и Тони Хоэ (Tony Hoare) в начале 1970-х годов.

Пример 13.4. Автопарки, обслуживающие живописные места

Задача. Существует множество живописных мест, откуда открываются великолепные панорамы. Есть такая точка обзора и возле Саванны, к которой ведут две дороги. Желающих посетить ее обслуживают два автопарка. Мы хотели бы знать количество машин, въезжающих за определенное время на территорию каждого из автопарков и суммарное количество машин, прибывших в город для посещения панорамы.

Решение. Необходимо создать апплет, который бы показывал непосредственно вид, открывающийся с выбранной точки обзора, а также счетчики, отражающие количество автомобилей в трех заданных местах. В ближайшее время администрация Саванны, занимающаяся охраной природы, может принять решение об ограничении количества автомобилей, въезжающих в город. В таком случае автопарки могут использоваться как буферы, и мы должны иметь это в виду при разработке программы. Предлагаем создать апплет (а не приложение) и показать, как подпроцессы взаимодействуют с апплетами. Мы не будем использовать функции броузера, поэтому система рассчитана на просмотр апплетов в программе `appletviewer`.

Разработка. Нам нужно рассмотреть три различных счетчика. Главный, который учитывает машины, посетившие панораму, "принадлежит" непосредственно апплету. Остальные два будут работать независимо и, следовательно, относиться к двум дополнительным подпроцессам. Эти подпроцессы идентичны во всем, за исключением имени (**Запад** и **Восток** — восточный и западный автопарки), поэтому мы определим один класс-подпроцесс и создадим два его экземпляра.

Алгоритм. Учет машин, побывавших на территории панорамы, выполняется посредством вызова метода, который обновляет общее их количество и выводит это значение на экран. Поскольку метод `enter` потенциально может быть вызван одновременно двумя подпроцессами и поскольку он обновляет переменную, мы объявим его как синхронизированный, гарантируя тем самым, что значение переменной будет правильным (вычислено без помех).

Программа. Рассмотрим текст программы.

```
import java.io.*;
import java.applet.*;
import java.awt.*;
import java.util.*;

public class CarPark extends Applet {

  /*  Апплет "Автопарк"          Автор Дж.М. Бишоп, январь 1998 г.
   *  =================          Java 1.2,  август 2000 г.
   *
   * Имитирует работу двух автопарков, обслуживающих панораму.
   * Считывает и выводит на экран вид местности.
   * Демонстрирует использование подпроцессов и
   * синхронизированного метода. Может быть выполнен
   * в программе просмотра апплетов.
   */

  Image im;
  ViewPoint view;
  Random delay = new Random();

  public void init () {
```

```
    // Считывает изображение.
    im = getImage(getCodeBase(),"reserve.jpg");
    // Гарантирует, что изображение будет получено до того,
    // как конструктор возобновит свою работу.
    MediaTracker tracker = new MediaTracker(this);
    tracker.addImage(im, 0);
    try { tracker.waitForID(0);}
    catch (InterruptedException e) {}
    Font f = new Font ("SanSerif",Font.PLAIN,24);
    setFont(f);

    view = new ViewPoint("Вид",140);
    new CarThread("Запад",0).start();
    new CarThread("Восток ",280).start();
  }

  class CarThread extends Thread {
  //-----------------------------

    int cars;
    int x;
    String pos;

    CarThread (String s, int n) {
      pos = s;
      x = n;
    }

    public void run () {
      while (true) {
        cars++;
        show(x,cars,pos);
        view.enter();
        try {sleep (factor(x));}
        catch (InterruptedException e) {}
      }
    }
  }

  class ViewPoint {
  //-------------

    int x;
    int cars;
    String pos;

    ViewPoint (String s, int n) {
      x = n;
      pos = s;
    }

    synchronized void enter () {
      cars++;
      show (x,cars,pos);
    }
  }
```

```
// Математические и графические функции
// - - - - - - - - - - - - - - - - - - - -
public void paint (Graphics g) {
  g.drawImage(im,0,0,this);
}

void show(int x, int cars, String s) {
  Graphics g = getGraphics();
  g.setColor(Color.orange);
  g.fillRect(x,300,120,50);
  g.setColor(Color.black);
  g.drawString(s+" "+cars,x+5,330);
}

int factor (int x) {
  return Math.abs(delay.nextInt()%5000+x);
}

}
```

Должен ли метод `show` также быть синхронизированным? Нет, `show` — это пассивный метод, в основном принимающий параметры и выводящий их на экран. Прямоугольники, соответствующие счетчикам, находятся в различных областях экрана, поэтому возникновение конфликтов невозможно. Предположим, что метод `show` также имеет операторы, позволяющие вывести на печать значения (использовать метод `print`). Не возникнет ли путаница между выводимыми строками? К счастью нет, потому что метод `print` и другие аналогичные методы определены в классе `PrintWriter` как `synchronized`. Следовательно, только один метод может их использовать, а значит, работать с объектом `System.out` в любой момент времени.

Тестирование. На рис. 13.8. показан вывод программы, выполненной в программе просмотра апплетов. HTML-файл содержит только тег апплета, поскольку изображение чертится апплетом, а не HTML.



Рис. 13.8. Вывод апплета с видом местности и счетчиками, представляющими автопарки и панораму

Сигнализация между подпроцессами

Защита совместно используемых ресурсов является одним из основных аспектов процесса синхронизации. Другим важным аспектом считается передача подпроцессу сообщения, указывающего, что на данный момент уже выполнено условие, исполнения которого он "ждет". Используемые методы приведены в описании далее.

Если подпроцесс обнаружит, что не может продолжать работу, поскольку объект, с которым он имеет дело, не находится в надлежащем состоянии, он вызовет метод `wait` из синхронизированного метода, принадлежащего данному объекту. После этого подпроцесс будет помещен в очередь, ассоциированную с объектом.

Подпроцесс, изменивший некоторые данные, имеющие отношение к другим подпроцессам, или условия в объекте, должен вызвать метод `notify` или `notifyAll`. Это даст ожидающим подпроцессам возможность проверить выполнение их условий, из-за которых они находятся в очереди. После этого некоторые из них смогут возобновить свою работу при следующей возможности, предоставленной операционной системой.

Методы синхронизации

`wait ();`
`notify ();`
`notifyAll ();`

Метод `wait` приостанавливает текущий подпроцесс, добавляя его в очередь, состоящую из подпроцессов, ожидающих своего шанса поработать с объектом. Метод `notify` "пробуждает" первый подпроцесс очереди (`notify` — переводится, как *уведомлять*), ожидающий возможности поработать с объектом. Метод `notifyAll` "пробуждает" все подпроцессы очереди.

Теоретически все ожидающие подпроцессы в конце концов должны быть уведомлены, что могут выполнить свою работу. На практике это зависит от внимательности программиста и от того, учел ли он все возможные комбинации. Вполне возможно создание такой программы, в которой не будут создаваться определенные условия и возникнет ситуация, когда подпроцесс 1 будет ждать подпроцесс 2, который ждет подпроцесс 1. Такое состояние называется **зависанием** или **тупиковой ситуацией (deadlock)** Java может обнаружить некоторые логические ошибки, особенно если методы `wait` и `notify` используются с несинхронизированными методами. В этом случае система создаст объект-исключение `IllegalMonitorStateException`.

В конце данной главы, в разделе "Практикум 9", показано, как подпроцессы различных видов могут стартовать во время выполнения и синхронизироваться для достижения конечной цели.

13.4. Класс Display

Класс `Display` неоднократно использовался в наших программах. Теперь мы наконец узнаем, из чего он состоит. Не имея никаких дополнительных подпроцессов, он использует методы класса `Thread` для осуществления мониторинга кнопки **Готово**. Вначале мы рассмотрим структуру класса `Display`, а затем используемые внутри него технологии.

Структура класса Display

Класс `Display`, текст которого занимает 220 строк, состоит из восьми открытых методов (некоторые из них являются перегружаемыми), трех закрытых методов и двух поддерживающих классов. Закрытые методы перечислены ниже.

- Метод `initializeDisplay` — осуществляет GUI-инициализацию окна, создает панели (возможно, с полосами прокрутки), текстовые поля для ввода, текстовую область для вывода, а также кнопки **Готово** и **Закрыть**.
- Метод `getEntry` — выполняет работу, общую для трех открытых методов `get`.
- Метод `insertPrompt` — также выполняет работу, общую для трех перегружаемых методов `prompt`.

Таблица вводимых значений

Значения, которые вводятся через GUI класса `Display`, содержатся в хеш-таблице, как показано на рис. 6.16. Таблица имеет интересную структуру. Как уже было сказано, ключ — это строка, которая может быть довольно длинной (например, "Имя на бирке"). Значения — это объекты класса

```
private class Data {
  TextField field;
  String value;
}
```

Как только посредством метода `prompt` запрашивается новое поле ввода, оно создается и добавляется в область ввода. Ссылка на это поле формирует часть вводимой записи в таблице. Вторая часть представляет собой значение, которое передается методу `prompt`. Заметьте, что мы сохраняем значение как строку. Это делает операцию ввода в текстовом поле более наглядной для пользователя. Если мы должны сохранять числа, то столкнемся с необходимостью назначить для них формат, который должен быть удобным для всех пользователей.

Довольно интересен метод `getEntry`. Он используется методами `getInt`, `getDouble` и `getString` для того, чтобы проверить, является ли передаваемая метка действительной, и получить копию текущих данных из таблицы. Вот как выглядит метод с одним из таких "get":

```
private Data getEntry (String s) {
  if (table.containsKey(s))
    return (Data) table.get(s);
  else {
    outDisplay.append("\nОшибка: такая метка не была введена: "+s+"\n");
    return null;
  }
}

public int getInt (String s) {
  Data d = getEntry(s);
  return Integer.valueOf(d.value).intValue();
}
```

Заметьте, что сообщение об ошибке посылается в область вывода, что удобно для пользователя.

Когда происходит обновление таблицы? Заметим, что нажатие клавиши [Enter] после произведенных изменений в поле данных вовсе не является необходимым. Однако нажатие на кнопку **Готово** срабатывает как главное инициирующее событие (такое же по важности, как и нажатие клавиши [Enter]), и именно на этом событии "лежит ответственность" за сканирование области ввода. Метод `ready` (инициируется при нажатии кнопки **Готово**) включает в себя цикл

```
// Копирует все значения из всех текстовых полей в таблицу.
for (Enumeration e = table.keys(); e.hasMoreElements();) {
  String name = (String) e.nextElement();
  Data d = (Data) table.get(name);
  d.value = d.field.getText();
  table.put(name, d);
}
```

Имея в таблице сохраненные ссылки на каждое текстовое поле, мы можем легко считать значения из GUI.

Наблюдатель за кнопкой "Готово"

Одним из наиболее интересных элементов класса `Display` является его класс `Watcher`. Логика применения этого *наблюдателя* (*watcher*) следующая. Когда программа произведет вызов метода `ready`, она передаст контроль классу `Display`, который после этого будет *готов* (так переводится с английского языка слово *ready*) считать значения из области ввода. Однако класс `Display` должен будет ждать, пока пользователь физически не нажмет кнопку **Готово**. Другими словами, мы должны синхронизировать работу метода `actionPerformed` и метода, который реагирует на нажатие кнопки.

Синхронизация выполняется в мониторе, код которого имеет вид:

```
class Watcher {

  private boolean ok;

  Watcher () {
  ok = false;
}

  synchronized void watch () { // Вызывается методом ready.
    while (!ok) {
    try {wait(500); }
    catch(InterruptedException e) {}
    }
    ok = false;
  }

  synchronized void ready () { // Вызывается методом actionPerformed.
    ok = true;
    notify();
  }
}
```

Метод watch использует подпроцесс wait для того, чтобы периодически, дождавшись заданного времени, проверять, назначено переменной ok значение true или нет. После того как будет нажата кнопка **Готово** и событие будет принято и обработано методом actionPerformed, будет вызван метод ready, который присвоит переменной ok значение true и уведомит всех ждущих в этом мониторе (в данном случае только метод watch). После этого метод watch активизируется, изменит значение переменной ok и будет готов к выполнению своих операций.

Далее следует полное определение класса Display.

```
package javagently;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

public class Display extends Frame implements ActionListener {

   /* Класс Display      Автор Дж.М. Бишоп, июль 1999 г.
    * ************
    * Этот простой класс предоставляет средства для
    * ввода и вывода данных в окно.
    * Значения данных вводятся в текстовые поля в области ввода.
    * Можно использовать различные типы данных, если
    * исполняющая программа будет в них нуждаться.
    * Возможна дополнительная интеграция с классом Graph.

    * Интерфейс
    * =========
    * new Display (заголовок) - создает новый объект Display с заголовком
    * println (строка) - выводит строку в области вывода
    * prompt (метка, значение) - выводит текстовое поле в область ввода
    *                            с данной меткой
    * ready (сообщение) - печатает сообщение, затем дает возможность
    *                     выполнять считывание из текстовых полей
    * getDouble (метка) - считывает значение типа double, которое
    *                     ассоциируется с этой меткой
    * getInt (метка) - получает значение типа int, которое соответствует
    *                  этой метке
    * getString (метка) - считывает значение типа string, которое соответствует
    *                     этой метке
    * reposition (изображение) - "берет" изображение и помещает его
    *                            в нижнюю половину области ввода
    */

  private String title;

  public Display (String t) {
    /* Альтернативный конструктор - имеет заголовок (title).*/
    title = t;
    initializeDisplay();
  }
```

```
private Hashtable table = new Hashtable(10);
private int xwidth, yheight;
private Button okButton, closeButton;
private TextArea outDisplay;
private Panel inDisplay;
private ScrollPane inPane, outPane;
private Watcher okWatcher = new Watcher();
private boolean graphInFront;
private Graph graph;

private void initializeDisplay () {
  xwidth = 640;
  yheight = 480;
  setSize(xwidth,yheight);
  setTitle(title);
  setLayout(new BorderLayout());
  Panel p = new Panel ();
    p.add(new Label("ВВОД"));
    p.add(new Label("ВЫВОД"));
  add(p,"North");
  p = new Panel (new FlowLayout(FlowLayout.CENTER,15,0));
    inPane = new ScrollPane(ScrollPane.SCROLLBARS_ALWAYS);
      inPane.setSize(xwidth/2 - 40, yheight - 100);
      inDisplay = new Panel(new GridLayout (0,2,10,10));
      inPane.add(inDisplay);
    p.add(inPane);
    outDisplay = new TextArea(24, 40);
    p.add(outDisplay);
  add(p,"Center");
  p = new Panel(new BorderLayout());
    okButton = new Button("Готово");
    okButton.addActionListener(this);
    okButton.setEnabled(false);
    p.add("Center",okButton);
    closeButton = new Button("Закрыть");
    closeButton.addActionListener(this);
    closeButton.setEnabled(true);
    p.add("East",closeButton);
  add("South",p);
    addWindowListener (new WindowAdapter () {
      public void windowClosing(WindowEvent e) {
        System.exit (0);
      }
    });
  setVisible(true);
  graphInFront = false;
}

public void reposition (Graph g) {
  // Делает изображение меньше и помещает его в нижнюю
  // половину области ввода.
  g.setLocation (30, yheight/2-30);
  g.setSize(xwidth/2 - 40, yheight/2-15);
  graphInFront = true;
```

```
    graph = g;
  }

  public void actionPerformed (ActionEvent e) {
    if (e.getSource() == okButton) {
    okWatcher.ready();
    } else
    if (e.getSource() == closeButton) {
      System.exit(0);
    }
  }

  private class Data {
    TextField field;
    String value;
  }

  private Data getEntry (String s) {
    if (table.containsKey(s))
      return (Data) table.get(s);
    else {
      outDisplay.append("\n:ОШИБКА: Такая метка не была введена "+s+"\n");
      return null;
    }
  }

  public int getInt (String s) {
    Data d = getEntry(s);
    return Integer.valueOf(d.value).intValue();
  }

  public double getDouble (String s) {
    Data d = getEntry(s);
    return Double.valueOf(d.value).doubleValue();
  }

  public String getString (String s) {
    Data d = getEntry(s);
    return d.value;
  }

  private void insertPrompt(Data d, String s, TextField t) {
    Panel p;
    p = new Panel(new FlowLayout(FlowLayout.RIGHT));
      p.add(new Label(s));
      inDisplay.add(p);
    t.addActionListener(this);
      t.setEditable(true);
      p = new Panel(new FlowLayout(FlowLayout.LEFT));
      p.add(t);
      inDisplay.add(p);
    d.field = t;
  }
```

```
public void prompt (String s, int n) {
  Data d = new Data();
  TextField t = new TextField(10);
  insertPrompt(d, s, t);
  d.value = Text.writeInt(n,0);
  t.setText(d.value);
  table.put(s, d);
}

public void prompt (String s, double n) {
   Data d = new Data();
  TextField t = new TextField(10);
  insertPrompt(d, s, t);
  d.value = Double.toString(n);
  t.setText(d.value);
  table.put(s, d);
}

public void prompt (String s, String n) {
  Data d = new Data();
  TextField t = new TextField(n.length()+2);
  insertPrompt(d, s, t);
  d.value = n;
  t.setText(d.value);
  table.put(s, d);
}

public void ready (String s) {
  outDisplay.append(s+"\n");
  okButton.setEnabled(true);
  setVisible(true);
  if (graphInFront) graph.toFront();
  okWatcher.watch();
  // Копирует все значения из текстовых полей в таблицу.
  for (Enumeration e = table.keys(); e.hasMoreElements();) {
    String name = (String) e.nextElement();
    Data d = (Data) table.get(name);
    d.value = d.field.getText();
    table.put(name, d);
  }
}

public void println (String s) {
  outDisplay.append(s+"\n");
}

public void print (String s) {
  outDisplay.append(s);
}

class Watcher {

  private boolean ok;
```

```
    Watcher () {
    ok = false;
    }

    synchronized void watch () {
      while (!ok) {
      try {wait(500); }
      catch(InterruptedException e) {}
      }
      ok = false;
    }

    synchronized void ready () {
      ok = true;
      notify();
    }
  }
}
```

13.5. Практикум 9. Использование в музее аудиоплееров

Администрация музея Саванны намеревается внедрить современную систему работы с посетителями, отказавшись от услуг экскурсоводов. Вместо этого каждому человеку будет выдан аудиоплеер, позволяющий прослушивать информацию в индивидуальном темпе. При входе в музей расположены *столы учета* (*counter*), где служащие за соответствующую плату (или залог) выдают (сдают в аренду) посетителям аудиоплееры. Плата равна 1 грацу. Посетители приходят группами от одного до десяти человек и ждут, пока не наберется достаточное количество аудиоплееров. Мы хотим создать программу, которая бы имитировала работу музея, для того чтобы выяснить, какое количество аудиоплееров следует иметь музею, чтобы обслужить определенный поток посетителей.

Алгоритм задачи

С чего же начать создание программы? Прежде всего следует нарисовать диаграмму, на которой были бы изображены все задействованные в программе компоненты, что позволит нагляднее идентифицировать объекты и их взаимодействие. Наш вариант такой диаграммы приведен на рис. 13.9.

В музее имеется определенное количество аудиоплееров. Для каждого служащего создается по одному подпроцессу `Counter`. *Музей* (*Museum*) — это класс-монитор с двумя синхронизированными методами, `hire` и `replace`. Подпроцессы `counter`, вызывают метод `hire` (дословный перевод — "нанимать, сдавать напрокат") и, когда их запрос на заданное количество аудиоплееров будет удовлетворен, создают новый подпроцесс `Visitors`, который соответствует группе *посетителей* (*visitors*).

Посетители оставляют свои аудиоплееры непосредственно на выходе из музея. Таким образом пополняется количество аудиоплееров, доступных для аренды. Если количества как раз хватает для удовлетворения имеющегося запроса на аудиоплееры, служащие музея экипируют новую группу экскурсантов.

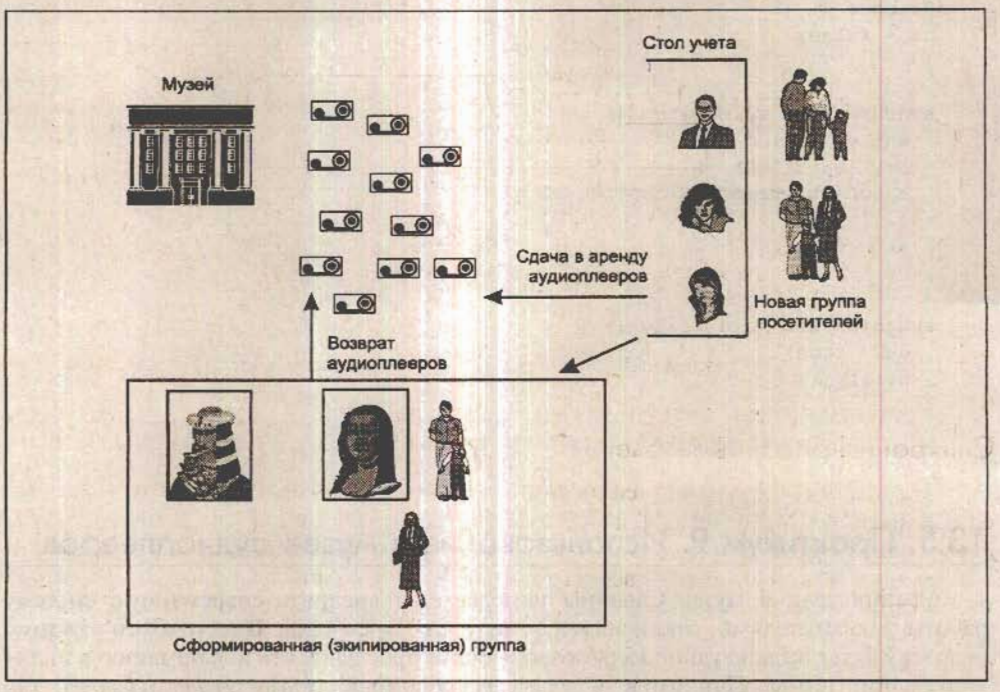


Рис. 13.9. Схема взаимоотношений между элементами программы

Предположим, что в наличии имеется только 2 аудиоплеера. Как видно из рисунка, ни одна из групп, ожидающих у стола учета, не может начать экскурсию с таким количеством "электронных экскурсоводов". Теперь предположим, что одна из групп, находящихся в зале с Моной Лизой, закончила осмотр экспозиции. Экскурсанты возвращают аудиоплееры, после чего те передаются в пользование другой группе, если, конечно, их набирается достаточное количество. После ухода очередной экипированной группы в распоряжении стола учета останется один (или ни одного) аудиоплеер.

Конструкция класса

На этот раз мы не будем чертить детальную схему программы, а лишь укажем направления взаимодействий между объектами четырех классов на высоком уровне (рис. 13.10). Класс `WalkmanHire` создает экземпляр класса `Museum`. Он также инициирует объекты `Counter` как подпроцессы. Объекты `Counter`, при необходимости, инициируют подпроцессы `Visitors`. Подпроцессы `Counter` распределяют аудиоплееры из объекта `Museum`, а посетители (`Visitors`) их возвращают в общий ресурс.

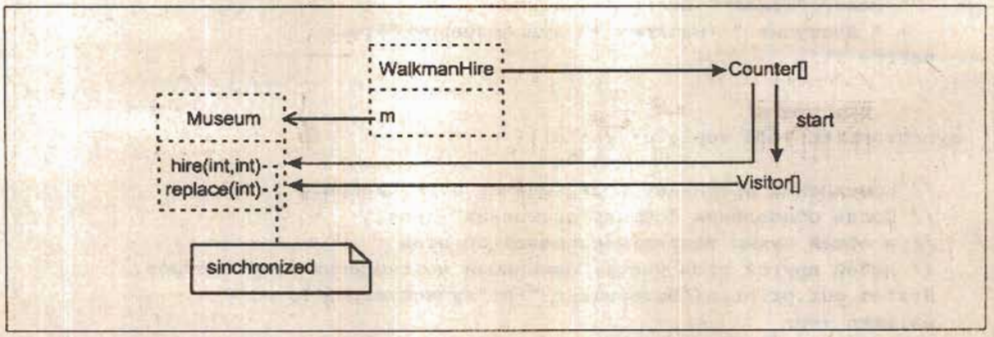


Рис. 13.10. Направления взаимодействий между объектами четырех классов на высоком уровне

Синхронизированные объекты

Теперь более детально рассмотрим ту часть программы, которая отвечает за синхронизацию. Методы `hire` и `replace` контролируют количество аудиоплееров и сумму наличности, принятой в качестве залога. Очевидно, что количество плееров, свободных на данный момент, всегда должно быть равным их исходному количеству. Рассмотрим текст класса `Museum`. Из комментариев вы сможете понять, как взаимодействуют методы `hire` и `replace`, хотя объяснение процесса аренды здесь дано очень поверхностно.

```
static class Museum {

  Museum (int w) {
    walkmen = w;
    cash = 0;
  }

  synchronized void hire (int c, int n) {
    // Если аудиоплееров недостаточно, ждет,
    // пока группа из другого подпроцесса не вернет
    // некоторое их количество и не уведомит об этом.
    // Если возвращенных аудиоплееров недостаточно,
    // будет продолжать ожидание.
    System.out.println("Столу учета "+c+" требуется "+n+"аудиоплееров");
    while (walkmen < n) {
      try { wait(); }
      catch (InterruptedException e) {}
    }

    // Сдает напрокат аудиоплееры и принимает залог.
    // После этого посетители могут уйти, а объект Counter
    // прекращает свой контроль над объектом-монитором
    // посредством вызова метода notify.
    walkmen -= n;
    cash += n;
    count++;
    System.out.println("Стол учета "+c+" получил "+n+"аудиоплееров");
    System.out.println("Общее состояние:"+
```

```
      " Залог "+cash+" Всего "+(walkmen+cash)
      + " Доступно " +walkmen +" аудиоплееров ");
    notify ();
  }

  synchronized void replace (int n) {

    // Немедленно принимает возвращаемые аудиоплееры.
    // После обновления "общего состояния" (pool)
    // и общей суммы залога уведомляет об этом
    // любой другой стол учета, ожидающий поступления аудиоплееров.
    System.out.println("Возвращено "+n+"аудиоплееров");
    walkmen +=n;
    cash -= n;
    notify ();
  }


}
```

Вызов той части класса `Museum`, которая соответствует методу `hire`, означает, что мы уже имеем подпроцессы `Counter`, из которых и вызывается этот метод. После того как потребность подпроцессов `Counter` в аудиоплеерах будет удовлетворена, они, подпроцессы, создают новый подпроцесс `Visitors`, а затем переходят на некоторое время в "спящее" состояние (вызывается метод `sleep`), прежде чем приступить к обслуживанию другой группы посетителей. И количество посетителей в группе, и время, которое должно предшествовать приходу новой группы, задаются путем генерирования случайных чисел.

```
static class Counter extends Thread {

  Counter (Museum m, int q) {
    museum = m;
    queue = q;
  }

  public void run () {

    // Решает, сколько нужно аудиоплееров для группы посетителей
    // и ждет, пока не наберется достаточное их количество.
    // После этого посетители отправляются на экскурсию
    // (стартует новый подпроцесс Visitors, который
    // выполняется независимо).
    while (count<30) {
      int w = (int)(Math.random()*1000%7);
      museum.hire(queue, w);
      new Visitors (museum, w).start();

      // Ожидает некоторое время, прежде чем придет новая группа.
      try {sleep((int) (Math.random()*1000%100));} catch(InterruptedException e) {}
    }
  }

  Museum museum;
  int queue;
}
```

Подпроцесс `Visitors` имитирует группу экскурсантов, переходящих из зала в зал в течение произвольного времени и по окончании осмотра возвращающих аудиоплееры непосредственно объекту `Museum`. Метод `replace` синхронизирован, поэтому может произойти небольшая пауза, если именно в этот момент подпроцесс `Counter` будет занят, завершая комплектацию аудиоплеерами другой группы. Но в любой другой момент времение посетители могут покинуть музей без задержек.

```
static class Visitors extends Thread {

  Visitors (Museum m, int w) {
    museum = m;
    groupSize = w;
  }

  public void run () {

    // Группа находится в музее в течение 50 "условных единиц времени".
    // (Вы можете изменить эту цифру, с тем чтобы она больше соответствовала
    // процессорной мощности вашего компьютера).
    // После экскурсии группа сдает свои аудиоплееры и уходит.
    // С ее уходом подпроцесс уничтожается.
    try {sleep((int) (Math.random()*1000)+1);}
          catch(InterruptedException e) {}
    museum.replace(groupSize);
  }

  Museum museum;
  int groupSize;
}
```

Далее идет текст самой программы*, отвечающей за создание экземпляра класса `Museum`, а также подпроцессов `Counter` и `Visitors`. Количество используемых аудиоплееров и число служащих стола учета можно задать в командной строке. По умолчанию эти параметры равны соответственно 20 и 3.

```
import java.io.*;
class WalkmanHire {

  /*            Программа
     "Использование аудиоплееров в музее"  Автор Дж.М. Бишоп, январь 1997 г.

     Имитирует аренду аудиоплееров (за 1 грац) при их фиксированном
     начальном количестве. Имеется несколько служащих музея,
     которые сдают напрокат и принимают аудиоплееры обратно.

     Первоначальное количество аудиоплееров равно 20, а количество
     служащих — 3, но эти цифры могут быть
     переопределены путем указания в качестве параметров новых значений.
     Например: java WalkmanHire 100 8.
     Начальное значение переменной cash равно 0.
     Демонстрирует использование мониторов, синхронизированных
     методов, а также методов wait и notify. Показывает, как одновременно
```

* Непосредственно для выполнения программы все классы — `Museum`, `Visitors` и `Counter` — нужно скопировать в класс `WalkmanHire`, расположив их перед строкой `public static void main`. — *Прим. ред.*

```
    выполняются главная программа и два вида подпроцессов.
  */
  public static void main (String [] args) {

    // Вводит в программу начальное количество (pool) аудиоплееров
    // и "открывает музей для посетителей".
    count = 0;
    if (args.length >= 1)
      pool = Integer.parseInt(args[0]);
    else pool = 20;
    Museum m = new Museum (pool);

    // Вводит в программу количество помощников
    // (количество подпроцессов) и инициирует их (стартует).
    if (args.length >= 2)
      helpers = Integer.parseInt(args[1]);
    else helpers = 3;
    for (int i=0; i<helpers; i++)
      new Counter (m,i).start();
  }
  static int pool;
  static int helpers;
  static int count;
  private static int walkmen;
  private static int cash;
}
```

Тестирование. Далее представлен вывод этой программы. Мы рассмотрим только его небольшую часть, где количество доступных аудиоплееров становится малым. Для удобства мы пронумеровали строки.

```
1.  Общее состояние: Залог 15 Всего 20 Доступно 5 аудиоплееров
2.  Столу учета 1 требуется 5 аудиоплееров
3.  Стол учета 1 получил 5 аудиоплееров
4.  Общее состояние: Залог 20 Всего 20 Доступно 0 аудиоплееров
5.  Столу учета 2 требуется 7 аудиоплееров
6.  Столу учета 0 требуется 2 аудиоплеера
7.  Столу учета 1 требуется 4 аудиоплеера
8.  Возвращено 1 аудиоплеер
9.  Возвращено 2 аудиоплеера
10. Стол учета 0 получил 2 аудиоплеера
11. Общее состояние: Залог 19 Всего 20 Доступно 1 аудиоплееров
12. Возвращено 6 аудиоплееров
13. Стол учета 2 получил 7 аудиоплееров
14. Общее состояние: Залог 20 Всего 20 Доступно 0 аудиоплееров
15. Столу учета 0 требуется 5 аудиоплееров
16. Столу учета 2 требуется 1 аудиоплеер
17. Возвращено 1 аудиоплеер
18. Возвращено 5 аудиоплееров
19. Стол учета 0 получил 5 аудиоплееров
20. Общее состояние: Залог 19 Всего 20 Доступно 1 аудиоплееров
21. Стол учета 2 получил 1 аудиоплеер
22. Общее состояние: Залог 20 Всего 20 Доступно 0 аудиоплееров
```

```
23. Столу учета 2 требуется 2 аудиоплеера
24. Столу учета 0 требуется 7 аудиоплееров
25. Возвращено 7 аудиоплееров
26. Стол учета 1 получил 4 аудиоплеера
27. Общее состояние: Залог 17 Всего 20 Доступно 3 аудиоплеера
28. Стол учета 2 получил 2 аудиоплеера
29. Общее состояние: Залог 19 Всего 20 Доступно 1 аудиоплеер
```

Для выполнения программы были заданы такие парметры: начальное, или общее, количество (`pool`) аудиоплееров, равное 20, и три стола учета (служащих) — 0, 1 и 2. Приведенный выше вывод программы начинается с момента, когда уже прошло несколько циклов и в наличии имеется 5 аудиоплееров. Столу учета 1 требуется 5 аудиоплееров, и он их получает. После этого каждый из столов учета посылает свои требования, но все они блокируются до тех пор, пока какая-нибудь группа не сдаст плееры. Из строки 8 видно, что в общий запас был возвращен 1 плеер, но этого не достаточно ни для одного из ожидающих подпроцессов, поэтому они продолжают ждать. В строке 9 возвращаются еще 2 плеера, и к моменту, зафиксированному в строке 10, в общем запасе имеется уже 3 аудиоплеера, и подпроцесс, который представляет стол учета 0, может продолжить свое выполнение. Заметьте, что он был не первым в очереди: подпроцесс 2 запросил 7 аудиоплееров раньше (строка 5), но по условию цикла `while` он продолжает ждать. К счастью, в 12-й строке возвращаются еще 6 аудиоплееров, в результате чего доступными становятся 7 таковых, а значит, подпроцесс 2 может продолжить работу и отправить свою группу осматривать экспозицию. Однако общий запас вновь уменьшился до нуля. Чтобы программа не входила в бесконечный цикл, количество выполнений метода `hire` ограничено значением 30.

Изучите остальные строки вывода и, задав другие параметры, выполните эту программу самостоятельно, что позволит вам ощутить ее динамичность.

Резюме

Для того чтобы обеспечить лучшее взаимодействие с пользователями и работу в сети, Java предоставляет возможность организовать несколько подпроцессов. Определяемый класс может расширить класс `Thread`, благодаря чему его объекты будут выполняться независимо и параллельно с помощью своих собственных методов `run`. Как правило, метод `run` содержит цикл и должен завершаться нормальным образом. Коммуникации между подпроцессами осуществляются посредством вызова методов и обновления незакрытых переменных. Если не исключено, что подпроцессы будут вмешиваться в работу друг друга, то методы, которые они вызывают, объявляются как синхронизированные. Таким образом Java гарантирует, что подпроцесс в синхронизированном методе закончится раньше, чем другой подпроцесс получит доступ к любому синхронизированному методу этого объекта. Для управления подпроцессами существует два метода синхронизации — `wait` и `notify`.

В этой главе мы представили одно новое ключевое слово `synchronized` и девять API для класса `Thread`:

```
Thread      wait
  start     notify
  run       notifyAll
  sleep  Display
  yield
```

Приведено пять новых описаний:

- создание подпроцесса;
- методы класса `Thread`;
- определение для подпроцесса метода `run`;
- синхронизированный метод;
- методы синхронизации.

и два новых объекта-исключения:

- `InterruptedException`;
- `IllegalMonitorStateException`.

Вопросы



13.1. Подпроцесс может быть неактивным, потому что он:

а) невезучий

б) ожидает своей очереди на использование процессора

в) находится в ожидании получения информации

г) ответ б или в

13.2. Чтобы реализовать подпроцесс, программист должен:

а) ничего не делать, потому что подпроцессы прозрачны

б) просто инициировать объекты-подпроцессы и дать им возможность выполняться

в) проверить, имеется ли достаточно памяти и других ресурсов для работы подпроцессов

г) создать экземпляры подпроцессов, синхронизировать их и определить, когда они должны быть уничтожены

13.3. Ядром подпроцесса является:

а) метод `run`

б) метод `start`

в) класс `Thread`

г) интерфейс `Runnable`

13.4. Какое воздействие на программу окажут следующие строки кода, взятые из подпроцесса `Spots` (из примера 13.1):

```
try {
  sleep (500);
}
catch (InterruptedException e) {
}
```

а) программа замедлит свою работу после 200 повторений цикла

б) компьютер войдет в сонный режим на 500 секунд

в) подпроцесс остановится на 500 миллисекунд, после чего метод `sleep` сгенерирует объект-исключение `InterruptedException` и выполнение подпроцесса продолжится

г) подпроцесс передаст выполнение задачи другому подпроцессу на полсекунды

13.5. Зачем некоторые операции в программе с несколькими подпроцессами должны быть синхронизированы?

а) чтобы предотвратить зависание

б) чтобы изменить системное время

в) чтобы дать возможность синхронизированному методу, который содержит код, модифицирующий глобальные переменные или состояние программы, закончить выполнение своей задачи, до того, как будет вызван другой метод

г) чтобы им не пришлось простаивать при зажженном свете светофора на каждом перекрестке

13.6. Класс `Display` не имеет ни внутреннего класса, расширяющего класс `Thread`, ни метода `run`. Какие в таком случае у него имеются атрибуты подпроцесса?

а) он имеет метод `ready`

б) он имеет класс `Watcher`, который вызывает методы `wait` и `notify`

в) все главные методы являются подпроцессами

г) ответы а и б

13.7. В каком пакете объявлен класс `Thread`?

а) `java.util`

б) `java.lang`

в) `java.io`

г) `java.applet`

13.8. Подпроцесс уничтожается при условии, что:

а) заканчивается программа, которая его создала

б) он вызывает метод `stop()`

в) он достигает своего естественного завершения, например тогда, когда заканчивается определенный в нем цикл `while`

г) завершаются все остальные подпроцессы этого класса

13.9. Чтобы создать новый автопарк (подпроцесс) из примера 13.4, мы должны использовать оператор:

а) `new CarThread ("North", 450);`

б) `new CarThread ("North", 450).start ();`

в) `new ViewPoint ("View2"), 450);`

г) `new ViewPoint ("View2"), 450).start();`

13.10. Какую структуру данных использует класс `Display` для контроля за вводимыми данными?

а) связанный список

б) массив

в) хеш-таблицу

г) вектор

Задачи

Примечание. Первые четыре задачи не связаны с синхронизацией.

13.1. **Британские светофоры.** В Великобритании каждый светофор имеет четыре фазы. После выключения красного света и перед включением желтого эти же два света загораются одновременно. Измените программу из примера 13.2 таким образом, чтобы она подходила под английские стандарты.

13.2. **Поймать "бегающего" гнома.** Программа, в которой предлагается поймать гнома (пример 12.4), использует псевдоанимацию. Изображение перемещается в результате генерирования случайных чисел, но только тогда, когда двигается курсор мыши. Измените программу, введя в нее подпроцесс, отвечающий за передвижение гнома.

13.3. **Оживающие фигуры.** Программа из задачи 11.1 позволяет выбрать геометрическую фигуру из всех выведенных на экран. Используя решения, предложенные в программе, которая выводит на экран разноцветные точки (Пример 13.1), и в программе "Поймать гнома" (Пример 12.4), расширьте выполняемую задачу, чтобы на любой стадии фигуры могли перемещаться по экрану. Сделайте это, используя пакет `Swing`, и добавьте бегунок, позволяющий регулировать скорость перемещения.

13.4. **Работающие часы.** В задаче 10.1. мы разработали внешний вид электронных часов. Теперь, используя подпроцесс, сделайте так, чтобы они работали, т.е. чтобы применялась цифровая индикация и одновременно могли функционировать кнопки настройки времени и вывода на экран изображения самих часов.

13.5. **Видимые аудиоплееры.** Программу по использованию аудиоплееров в музее (Практикум 9) можно расширить, добавив в нее пользовательский интерфейс. Это позволит показать, как аудиоплееры возвращаются и вновь сдаются напрокат, а также внедрить контрольную панель, с помощью которой пользователь мог бы контролировать "прибытие" новых групп посетителей, не прибегая к генерированию случайных чисел. Создайте пользовательский интерфейс, который будет работать в качестве дополнительного подпроцесса в программе.

13.6. **Поломка аудиоплеера.** Вполне возможна ситуация, что кто-то из экскурсантов обнаружит, что его плеер не работает. Вместо того чтобы возвратить плеер непосредственно на выходе, посетитель вынужден будет вернуться к столу учета и заменить его другим. Как должна измениться схема при таком изменении программы и какие понадобятся методы? Скорректируйте код программы, а заодно добавьте класс `Curator`, который периодически "забирает" сломанные аудиоплееры, а через некоторое время возвращает их.

13.7. **VIP-посетители.** Некоторые посетители могут оказаться "очень важными персонами" (Very Important Person), у которых наблюдается "аллергия" на очереди, поэтому они не могут стоять в таковых (в нашем случае — за плеерами). Добавьте в класс `Curator` дополнительную функцию, которая время от времени "организовывала" бы появление в музее таких VIP-групп. Эти группы при получении аудиоплееров должны иметь наивысший приоритет. Выполните программу и по выводимым результатам определите, какой эффект дадут произведенные изменения.

13.8. **Ограничение доступа к панораме.** Руководство проекта по охране природы озабочено ростом количества автомобилей, пребывающих на территории панорамы одновременно (Пример 13.4). Посмотрите, можно ли расширить программу, чтобы в точке, откуда открывается самый лучший вид, было ограничено число мест парковки и чтобы туда могли въезжать только машины из автопарков и только тогда, когда появляются свободные места. Если мест нет, с помощью метода синхронизации `wait` организуйте очередь. После того как машина покидает территорию панорамы (по прошествии некоторого времени, установленного путем генерирования случайных чисел), автопарк уведомляется о том, что можно прислать еще один автомобиль.

ГЛАВА 14



Работа в сети

14.1. Соединение через Internet

Язык Java был создан с целью расширения средств доступа к сети. Поэтому он предоставляет широкий спектр возможностей в плане сетевого программирования — от создания соединений через URL и сокеты до методов удаленного доступа (*RMI — Remote Methods Invocation*) и функций подключения к базам данных (*JDBC — Java DataBase Connecting*). В Java соединения такого рода реализуются очень просто, поэтому вы не найдете в этой главе пространных пояснений — как и ранее, изложение построено на примерах.

Все четыре метода, которые предназначены для создания соединений, являются частью базовых API языка Java и поддерживаются стандартным JDK и любым броузером, работающим с Java версии 1.2 и выше. В этой главе мы будем использовать следующие пакеты: `java.net`, `java.rmi`, `java.sql`.

URL-соединения

В Java имеется класс `URL`, позволяющий загрузить из Internet данные, на которые указывает URL. Получив строку, Java создает соответствующий URL, который затем можно использовать для установления соединения `URLConnection`. Соединение дает возможность осуществлять взаимодействие с ресурсом в соответствии с его протоколом (возможно, через потоки ввода/вывода).

Методы создания URL приведены в следующей таблице.

Создание URL
`URL urlid = new URL (urlstring);` `URL urlid = new URL (urlbase, urlstring);`
Первый оператор пытается создать URL из данной ему строки. Если это невозможно, система генерирует объект-исключение `MalformedURLException`. Второй оператор позволяет создать относительный URL, т.е. сформировать его на основе URL существующего объекта.

Например:

```
URL info = new info
("http://u/java/Book/chap13/Spots.java");
```

Вы можете использовать URL-соединение с локальной машиной и таким образом обойти ограничение, согласно которому апплет не может читать локальные файлы. Из приведенной ниже команды видно, что мы обращаемся к жесткому диску C:, т.е. фактически указываем на локальный файл, но с помощью Internet-технологии.

```
URL data = new data
("file:///C|/books/jges/chap8/rates.data");
```

Для того чтобы обеспечить независимость апплета и используемых им файлов от машины, на которой выполняется апплет, необходимо пользоваться относительными URL (что мы и будем делать в наших примерах). В этом случае сначала при помощи метода `getCodeBase` апплета извлекается URL, отвечающий адресу апплета, а затем задается адрес файла относительно адреса апплета. Например:

```
URL info = new URL (getCodeBase(),"rates.data");
```

URL также может быть указан относительно URL страницы, с которой загружен апплет. Чтобы узнать URL страницы, используем метод `getDocumentBase()` апплета. Конструктор выполняет синтаксическую проверку заданного URL и в случае ошибки выдает объект-исключение `MalformedURLException`. Поэтому объявление URL обычно помещают в оператор `try`, чтобы перехватить исключение.

Для получения доступа к ресурсу посредством URL мы должны установить соединение. Если ресурсом является текстовый файл, можно создать к нему поток ввода. Как это сделать, описано ниже.

Создание URL-соединения

```
URLConnection connection = URLid.openConnection();

BufferedReader readerid = new InputStreamReader
        (connection.getInputStream());
Stream streamid = new Stream
        (connection.getInputStream());
```

`connection` – это объект, созданный на основе данного объекта `URLid`.
Он используется для получения потока ввода, который передается как параметр объекту `InputStreamReader` для создания объекта `BufferedReader` или объекту `Stream` для создания экземпляра объекта класса `Stream`.

Объекты, названные в описании `readerid` и `streamid`, работают как обычные источники ввода. Таким образом, мы можем объявить соответствующий метод и применять его для установления всех соединений, предназначенных для доступа к файлам. Метод возвращает объект `BufferedReader`, т.е. мы работаем с удаленными файлами так же, как с локальными. Код этого метода может быть таким:

```
BufferedReader connectTo (String filename) {
/* Метод для соединения файла (filename) с объектом BufferedReader
   через Internet. Использует метод класса Display
   для вывода сообщений об ошибках.
*/
  BufferedReader fin = null;
  try {
    URL dataFileURL = new URL(getCodeBase()+filename);
    display.println("Установление соединения с "+dataFileURL);
    URLConnection con = dataFileURL.openConnection();
    try {
      fin = new BufferedReader(new InputStreamReader
            (con.getInputStream()));
    }
    catch (IOException e) {
        display.println("Файл "+filename+" невозможно открыть");
    }
```

```
    }
    catch (Exception e) {
      display.println("Непредвиденная ошибка при соединении");
    }
    return fin;
  }
```

В данном случае метод `connectTo` отправляет сообщения с помощью метода класса `Display`. Однако, произведя небольшие изменения, можно сделать так, чтобы он посылал сообщения посредством оператора `System.out.println();`.

Пример 14.1. Расчет операции конвертирования валюты в апплете

Задача. Директора туристических агентств Саванны хотят иметь программу конвертирования валют, которую можно через Internet предлагать потенциальным путешественникам.

Решение. Следуя инструкции из раздела 12.1, преобразуйте программу, разработанную в примере 11.3, в апплет. Создайте HTML-файл, содержащий вызов апплета. Это позволит вызвать апплет при помощи броузера.

Конструкция программы. Сначала мы должны выполнить в программе из примера 11.3 изменения, описанные в разделе 12.1 и примере 12.1. Однако это еще не все. Поскольку главная программа удалена, мы можем объединить все части программы в один класс.

Осталось учесть еще один момент. Поскольку апплет будет запущен на компьютере клиента, а не на компьютере, находящемся в бюро путешествий, может возникнуть проблема с доступом к файлу `rates.data`. Как ее решить, описано выше. Отметим, что для апплет-версии не нужно изменять класс `Rates`. Код апплета представлен ниже (коды методов, оставшихся в неизменном виде, мы решили не повторять).

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import java.applet.*;
import java.net.*;
import javagently.*;

public class ConverterGUIApplet extends Applet
    implements ActionListener, ItemListener {
   /* Апплет "Конвертер валюты"        Автор Дж.М. Бишоп, декабрь 1998 г.
    *  - - - - - - - Версия с использованием класса Display, июль 1999 г.
    *                                          GUI-версия, июль 1999 г.
    *                                        Java 1.2., август 2000 г.
    * Содержит курсы обмена одной валюты на ряд других
    * и подсчитывает результат конвертирования введенной суммы.
    *
    * Демонстрирует использование простого GUI
    * с раскрывающимися списками, текстовым полем,
    * кнопками, текстовой областью и метками.
    */

  ... Объявления, как и в предыдущей программе.

  public void init () {
  ... как и раньше
  }
```

```
  public void actionPerformed (ActionEvent e) {
  ... как и раньше
  }

  public void itemStateChanged(ItemEvent e) {
  ... как и раньше
  }

  void transaction () {
  ... как и раньше
  }

  void readIn () {
    try {
      // Установление доступа к файлу rates.data через Internet.
      URL ratesFileURL = new URL(getCodeBase()+"rates.data");
      URLConnection con = ratesFileURL.openConnection();
      Stream fin = null;
      try {
        fin = new Stream(con.getInputStream());
      }
      catch (IOException e) {
        resultField.append("Файл rates.data не найден\n");
      }
    }
  ... как и раньше
  }
}
```

Тестирование. Мы создали Web-страницу и кроме вызова апплета поместили не нее немного текста. Файл страницы сохранен под именем `Converter.html`. Именно этот файл мы открываем для вызова апплета. На рис. 14.1 показаны результаты работы данного апплета в броузере Microsoft Internet Explorer.

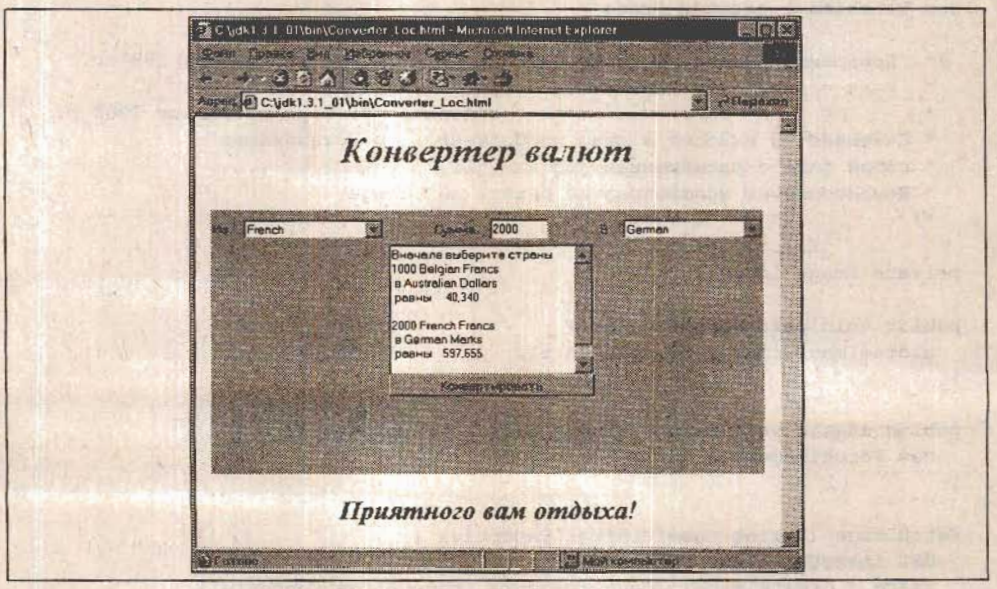


Рис. 14.1. Апплет "Конвертер валют" в броузере Internet Explorer

Отображение рисунков

Существует очень простой способ вывода рисунков на экран. Он заключается в использовании служб, предоставляемых пакетом `awt.image`. Как организовать этот процесс, описано ниже.

Отображение рисунков
`Image imageid = createImage((ImageProducer) imageURL.getContent());`
Метод `getContent` возвращает содержимое файла. Содержимое приводится к типу класса, который соответствует требованиям интерфейса `ImageProducer`. Для того чтобы преобразовать объект-интерфейс `ImageProducer` в объект `Image`, в объекте `Frame` вызывается метод `createImage`.

После того как все это сделано и фрейм объявлен видимым (т.е. выполнен оператор `setVisible(true);`), для вывода изображения на экран необходимо вызвать метод `paint`.

Пример 14.2. Считывание изображения и вывод его на экран

Программа. В следующей программе продемонстрировано, как считать и вывести на экран изображение, путь к которому указан при помощи URL.

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.net.*;

class FetchImage extends Frame {

   /*  Программа "Вывод изображения"  Автор Дж.М. Бишоп, январь 1997 г.
    *  ============================  Java 1.1
    *                                обновлена Б. Воролл, август 2000 г.
    * Считывает и выводит в окно изображение, представляющее
    * собой файл с расширением jpg или gif.
    * Демонстрирует использование пакета awt.image.
    */

  private Image image;

  public void paint(Graphics g) {
    g.drawImage(image, 50, 50, this);
  }

  public static void main(String[] args) throws Exception {
    new FetchImage(args[0]);
  }

  FetchImage (String name) throws Exception {
    URL imageURL = new URL(name);
    image = createImage((ImageProducer) imageURL.getContent());
    setSize(300, 300);
```

```
    setVisible(true);
    addWindowListener(new WindowAdapter () {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });
  }
}
```

Тестирование. Запустим программу из командной строки, указав имя файла, хранящего изображение:

```
java FetchImage http://адрес/Elephant.jpg
```

В результате откроется окно с изображением — фотографией слона. Поскольку мы включили в программу метод `WindowClosing`, ее окно можно закрыть, щелкнув мышью на кнопке закрытия окна.

14.2. Порты и сокеты

В предыдущем разделе мы рассмотрели, как устанавливается соединение на высоком уровне, т.е. через URL. Java предоставляет возможность работать с соединениями и на более низком уровне — через порты и сокеты. **Порт** — это абстрактный физический объект, через который осуществляется коммуникация между клиентом и сервером. При этом сервер предоставляет порт, а клиент с ним соединяется.

В операционных системах имеются процессы, назначенные конкретным портам. Программное обеспечение сервера непрерывно прослушивает эти порты на предмет обнаружения сообщений определенного вида. Как правило, порты идентифицируются при помощи номеров. Например, в системе Unix, соединившись с портом 13, вы можете получить системное время и дату, установленную на данном компьютере. Другие порты обеспечивают получение и отправку электронной почты, проверку статуса компьютера, выявление пользователей, зарегистрированных в системе, и т.д. Из соображений безопасности такие порты большую часть времени не доступны для обычных пользователей. Однако есть много незадействованных портов, которые мы можем применять для создаваемых нами служб.

Примером такой службы является многопотоковая Java-программа, обеспечивающая сокеты для данного порта. **Сокет** (гнездо) — это абстракция сетевого программного обеспечения, которая дает возможность осуществлять входные и выходные соединения к данной программе. Java-сокет может быть создан, если мы имеем действительный Internet-адрес компьютера и действительный номер порта. Для одного порта можно создать несколько сокетов, что позволяет нескольким пользователям получать доступ к данной службе, как это показано на рис. 14.2.

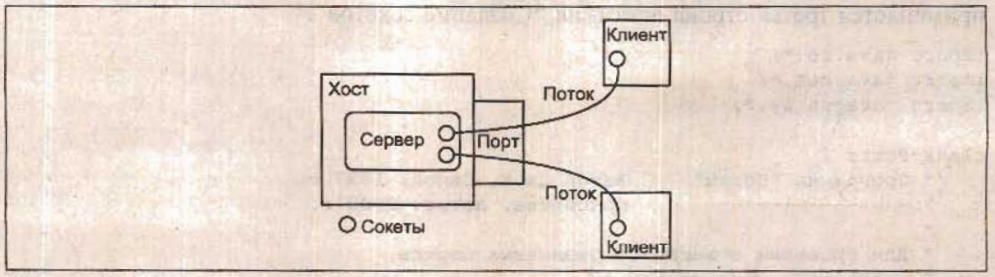


Рис. 14.2. Порты и сокеты

После создания сокета клиент и сервер могут обмениваться информацией определенным для данного соединения способом. Простейшим методом организации обмена является установление обычного потока от сервера к клиенту. После этого сервер может применять для получения и отправки данных методы `read` и `print`.

Если клиентом является Java-программа, то она также должна иметь сокет и потоки, соответствующие аналогичным потокам сервера. Клиентом может быть и служебная программа, такая как `telnet`. В качестве сервера может выступать системная служба, например, служба времени. Как создается сокет, описано ниже.

Создание сокетов

```
ServerSocket listener = new ServerSocket (port);
Socket socketid = listener.accept();
Socket socketid = new Socket (host, port);
```

В следующей таблице описано, как присоединить потоки ввода и вывода к созданному сокету.

Потоки для сокетов

```
BufferedReader instreamname =
    new InputStreamReader (name.getInputStream());
 PrintWriter outstreamname = new
    PrintWriter(name.getOutPutStream(),true);
```

Как видно из описания, работать с сокетами можно при помощи класса `Stream`. Чтобы создать сокет на стороне сервера, сначала необходимо назначить (создать) для данного порта постоянный прослушивающий сокет, базирующийся на классе `ServerSocket`. Затем для каждого клиента, соединяющегося с портом, посредством метода `accept` отдельно создается сокет, представляющий собой объект класса `Socket`. Со стороны клиента для создания такого сокета создается экземпляр класса `Socket` с указанием хоста и порта, с которыми нужно соединиться.

Выходной поток создается по той же схеме, что и другие потоковые соединения, с тем отличием, что он относится к классу `PrintWriter`. Данный класс сразу осуществляет вывод строки (параметр `true`).

Пример 14.3. Открытие порта

Программа. Это простая клиентская программа, организующая доступ к существующему серверу. В данном случае мы используем порт 13, который возвращает значение системного времени, установленного на компьютере. Для создания сокета применяется третья строка описания "Создание сокетов".

```
import java.io.*;
import java.net.*;
import javagently.*;

class Ports {
   /* Программа "Порты"     Автор Дж.М. Бишоп, 1997 г.
    * ----------------      обновлена, август 2000 г.
    *
    * Для проверки ответов от различных портов
    * удаленного компьютера.
```

```
   * Демонстрирует применение сокетов.
   */

  public static void main(String[] args) {
    String computer;
    int port;

    if (args.length > 0) computer = args[0];
    else computer = "syd.cs.up.ac.za";
    if (args.length > 1) port = Integer.parseInt(args[1]);
    else port = 13;
    // Порт, позволяющий узнать системное время.

    new Ports (computer, port);
  }

  Ports (String computer, int port) {
    System.out.println("Осуществление доступа к порту "
           +port+" компьютера "+ computer+"\n");
    try {

      // Создает сокет с данным номером порта.
      // Создает поток ввода, чтобы считать то, что может предложить
      // сокет, находящийся на противоположной стороне.

      Socket t = new Socket(computer, port);
      Stream in = new Stream(t.getInputStream());

      // Считывает из сервера информацию, доступную через этот порт
      // до тех пор, пока метод readLine не считает все данные.
      while (true) {
        String s = in.readLine();
        if (s == null) break;
        else System.out.println(s);
      }
    }
    catch(IOException e) { System.out.println("Ошибка" + e); }
  }
}
```

Тестирование. Если запустить эту программу без указания параметров, то она по умолчанию обратится к компьютеру `syd.cs.up.ac.za` и порту с номером 13. Результатом работы программы будут выведенные на печать дата и время, представленные в таком виде:

```
Осуществление доступа к порту 13 компьютера syd.cs.up.ac.za

Sun Aug 26 15:57:14 2000
```

Теперь рассмотрим, как создать программу-сервер, которая обслуживает свободный порт. Порты с номерами больше 8000 обычно свободны, поэтому можно использовать один из них.

Пример 14.4. Создание ATM-сервера

Задача. Требуется сымитировать работу и обслуживание банкоматов (ATM —automated teller machines), т.е. соединение их с банком и проверку достоверности

вводимого клиентом личного идентификационного номера (PIN — personal identification number).

Решение. Установим программу-сервер на свободный порт. Служба должна обрабатывать несколько одновременных соединений с клиентами, представляющими собой банкоматы. Между клиентом и сервером будет происходить приблизительно следующий обмен сообщениями:

Сервер	Клиент
Добро пожаловать в банк "Саванна"	
Введите, пожалуйста, Ваш номер PIN или слово ОТМЕНИТЬ	1234
Неверный номер PIN. Введите его еще раз	5678
Можете начать транзакции	

Поскольку мы создаем имитацию и в данной службе нет базы данных клиентов, то в качестве PINa для всех клиентов будет использовано число 5678.

Конструкция программы. Поскольку нам надо работать с большим количеством клиентов, то, очевидно, что транзакции будут осуществляться с различными скоростями и, следовательно, должны в рамках сервера иметь свои собственные подпроцессы. Как их реализовать, вы уже знаете, поскольку порядок создания подпроцессов мы рассмотрели в главе 13. Вопрос, скорее, состоит в том, как протестировать работу службы. Конечно, можно создать программу-клиент на Java, аналогичную приведенной в примере 14.3. Однако имеется и более простой способ: использовать стандартную программу-клиент `telnet`. `Telnet` — это программное обеспечение с простым протоколом, которое позволяет осуществлять построчный ввод и вывод информации.

Программа. Программа использует класс `InetAddress`, который преобразует IP-адреса в имена. Код программы-сервера с комментариями, объясняющими ее работу, приведен ниже.

```
import java.io.*;
import java.net.*;
import javagently.*;

class ATMServer {

  /* Простая программа-сервер   Автор Дж.М. Бишоп, декабрь 1996 г.
   * ========================
   *                            Java 1.1, исправлена январь 1998 г.
   *                            Java 1.2 Б.Воролл, август 2000 г.
   * Позволяет работать с несколькими соединениями одновременно.
   * Демонстрирует использование сокетов и работу в сети.
   */

  static final String magicPIN = "5678";
  static int          port = 8190;
  InetAddress         serverAddress = null;
  private boolean     done = false;

  public static void main(String[] args) {
   // Назначает номер порта из командной строки
   // или применяет установленный по умолчанию порт 8190.
    if (args.length > 0)
       port =Integer.parseInt(args[0]);
```

```
      new ATMServer (port);
  }

  ATMServer (int port) {

    try {
      serverAddress = InetAddress.getLocalHost();
    } catch (UnknownHostException e) {}

   // Первоначальный вывод на экран можно осуществить только на стороне
   // сервера, используя оператор System.out.println();.
    System.out.println("******  БАНК \"САВАННА\" ********");
    System.out.println("Имитирует сеанс ATM, "
          + "обслуживая запросы клиентов к серверу ");
    System.out.println(serverAddress.getHostName() +
          ", номер порта "+port);
    System.out.println("от любого количества компьютеров или");
    System.out.println("из различных активных окон.");

   // Создает сокет сервера.
     try {
       ServerSocket listener = new ServerSocket(port);

       int c = 0;
       while (!done) {

         // Теперь программа ждет новых клиентов.
         Socket client = listener.accept( );
         c ++;
         System.out.println("Кредитная карточка вставлена в " +
               client.getInetAddress().getHostName());
         System.out.println("Начало обслуживания нового клиента под номером "
               +c);
         new handler(client, c).start();
       }
       listener.close();
     }
     catch (IOException e) {
       System.out.println("Порт "+port+
         ", возможно, занят. Попытайтесь через другой.");
     }
   }

  void closeDown () {
    done = true;
  }

  class handler extends Thread {

    private Socket toClient;
    private int id;

    handler(Socket s, int i) {
    // Запоминает номер сокета и идентификационный номер клиента.
```

```
        toClient = s;
        id = i;
      }

      public void run() {

        try {
          Stream conin = new Stream (toClient.getInputStream());
          PrintWriter conout = new
            PrintWriter(toClient.getOutputStream(),true);

          conout.println( "Добро пожаловать в банк \"Саванна\"");
          for (int tries = 0; tries < 3; tries++) {
            conout.println("Введите, пожалуйста, Ваш номер PIN"+
                " или слово ОТМЕНИТЬ");
            String s = conin.readLine();
            System.out.println("Клиент "+id+":"+s);

            if (s.equals("ВЫКЛЮЧИТЬ")) closeDown();
            else if (s.equals("ОТМЕНИТЬ")) {
              conout.println("Транзакция завершена. До свидания.");
              break;
            }

            else if (s.equals(magicPIN)) {
              conout.println("Можете начать транзакции");
              break;
            }
            else
              conout.println("Неверный номер PIN. Попытайтесь еще раз");
          }
          System.out.println("Имитация завершена. Спасибо.");
        }
        catch (IOException e) {System.out.println("Ошибка ввода-вывода");}
      }
    }
}
```

При создании каждый подпроцесс запоминает номер своего сокета. Затем он запрашивает потоки ввода и вывода, ассоциированные с этим сокетом. Они дадут ему доступ к клиенту, находящемуся на противоположной стороне. Подпроцесс-обработчик (`handler`) отправляет вступительное сообщение на банкомат (клиенту) и затем максимум три раза запрашивает у клиента номер PIN. Клиент может ввести слово "ОТМЕНИТЬ" и даже имеет шанс остановить сервер, введя слово "ВЫКЛЮЧИТЬ" (хотя, естественно, эта информация ему не сообщается). По завершении диалога с клиентом сокет, созданный для работы с ним, закрывается, а соответствующий ему подпроцесс уничтожается.

Тестирование. Для прогона программы-сервера запустим ее как обычную Java-программу. Если для обращения к серверу в качестве клиента мы захотим использовать программу `telnet` на той же машине (что вполне вероятно), то должны запустить программу-сервер `java ATMserver` в одном окне, а в других окнах вызвать `telnet` (рис. 14.3).

Программа-сервер

Далее приведен полный текст (код) программы-сервера Chat.

```
import java.io.*;
import java.net.*;
import java.util.*;
import javagently.*;

public class ChatServer {

   /* Программа "Chatter"    Автор Дж.М. Бишоп,  январь 1997 г.
    * ==================     Java 1.1, январь 1998 г.
    *                        обновлена, август 2000 г.
    *
    * Создает программу-сервер для обслуживания
    * обмена сообщениями между несколькими клиентами.
    *
    * Для присоединения введите
    * telnet x y
    * где x и y - это имя компьютера и порт,
    * номер которого отображается после запуска программы.
    *
    * Демонстрирует использование сокетов, потоков через эти сокеты,
    * подпроцессов, синхронизации и связных списков.
    */

  private static LinkedList clientList = new LinkedList();
  private static int id = 0;

  public static void main(String[] args) throws IOException {
    // Получает номер порта и создает на нем чат-сервер.
    int port = 8190;
    if (args.length > 0)
      port = Integer.parseInt(args[0]);
    new ChatServer (port);
  }

  ChatServer (int port) throws IOException {
    ServerSocket listener = new ServerSocket(port);
    System.out.println("Порт, на котором выполняется программа
                        ChatServer - "+port);

    // Прослушивает клиентов. Создает новый подпроцесс-
    // обработчик для каждого из них.
    // Добавляет клиента в связный список.
    while (true) {
      Socket client = listener.accept();
      new ChatHandler(client).start();
      System.out.println("Новый клиент № "+id+
          " из "+ listener.getInetAddress()+
          " порт клиента "+client.getPort());
      clientList.add(client);
      id++;
    }
  }

  synchronized static void broadcast(String message, String name)
```

```
      throws IOException {
    // Посылает сообщение каждому клиенту, включая отправителя.
    Socket s;
    PrintWriter p;
    for (ListIterator list = clientList.listIterator(); list.hasNext();) {
      s = (Socket) list.next();
      p = new PrintWriter(s.getOutputStream(), true);
      p.println(name+": "+message);
    }
  }

  synchronized static void remove(Socket s) {
  /* Используя метод remove класса LinkedList, удаляет
   * первый экземпляр этого объекта-сокета.
   */
    clientList.remove(s);
    id--;
  }
}
```

Для каждого клиента существует один объект `ChatHandler`. Это подпроцесс с методом `run`, который создает потоки и выполняется в цикле, пока клиент не введет строку "ДО_СВИДАНИЯ".

```
import java.io.*;
import java.net.*;

class ChatHandler extends Thread {
  /* Класс ChatHandler вызывается из ChatServer:
   * по одному подпроцессу для каждого клиента, "зашедшего поболтать".
   */

  private BufferedReader in;
  private PrintWriter out;
  private Socket toClient;
  private String name;

  ChatHandler(Socket s) {
    toClient = s;
  }

  public void run() {
    try {
      /* Создает потоки ввода-вывода через сокет, который
       * мы получаем при создании экземпляра подпроцесса,
       * и приветствует нового клиента.
       */

      in = new BufferedReader(new InputStreamReader(
        toClient.getInputStream()));
      out = new PrintWriter(toClient.getOutputStream(), true);
      out.println("*** Добро пожаловать в Chatter ***");
      out.println("Для выхода введите ДО_СВИДАНИЯ");
      out.print("Как Вас зовут? ");
      out.flush();
      String name = in.readLine();
      ChatServer.broadcast(name+" присоединился к дискуссии.",
        "Chatter");
```

```
      // Считывает строки и отправляет их для
      // широковещательной рассылки.
      while (true) {
        String s = in.readLine().trim();

        if (s.equals("ДО_СВИДАНИЯ")) {
          ChatServer.broadcast(name+" вышел из разговора.",
          "Chatter");
          break;
        }
        ChatServer.broadcast(s, name);
      }
      ChatServer.remove(toClient);
      toClient.close();
    }
    catch (Exception e) {
    System.out.println(" Ошибка в программе Chatter : "+e);
    }
  }
}
```

Пример сессии chat

Мы вновь в качестве клиентов используем `telnet`. Далее приводится результат работы одного из сеансов *chat*.

```
Chat/Chatter>java ChatServer 8191 &
[2] 5802
Chat/Chatter>Порт, на котором выполняется программа ChatServer —  8191

Chat/Chatter>telnet syd 8191
Trying 137.215.18.16 ...
Connected to syd.cs.up.ac.za.
Новый клиент № 0 порт клиента 57889
Escape character is "^]".
*** Добро пожаловать в Chatter ***
Для выхода введите ДО_СВИДАНИЯ
Как Вас зовут?
Нельсон
Chatter: Нельсон присоединился к дискуссии.
Есть кто живой?
Нельсон: Есть кто живой?
Новый клиент № 1 порт клиента 1026
Chatter: Сигал присоединился к дискуссии.
Привет, Сигал
Нельсон: Привет, Сигал
Сигал: Привет, Нельсон, как дела в кофейном магазине?
Отлично, бизнес процветает.
Нельсон: Отлично, бизнес процветает.
Сигал: Ну что ж, успехов, пока!
Сигал — чтобы выйти из сеанса, ты должен ввести ДО_СВИДАНИЯ
Нельсон: Сигал, чтобы выйти из сеанса, ты должен ввести ДО_СВИДАНИЯ
Сигал: ДО_СВИДАНИЯ
Попытайся еще раз
Нельсон: Попытайся еще раз
Сигал: ДО_СВИДАНИЯ
```

```
Chatter: Сигал вышел из разговора.
ДО_СВИДАНИЯ
Нельсон: ДО_СВИДАНИЯ
Chatter: Нельсон вышел из разговора
Connection closed by foreign host.
```

Заметьте, что мы указали `telnet` соединиться с хостом `syd`. `Telnet` преобразовала это имя в IP-адрес — 137.215.18.16. После установки соединения с клиентом сервер берет свой IP-адрес и преобразует его в имя компьютера, в данном случае `syd.cs.up.ac.za`. Вы можете использовать специальный Internet-адрес 127.0.0.1 и подключиться с помощью `telnet` к своей машине.

При работе программы `ChatServer` с клиентом `telnet` могут наблюдаться некоторые "странности". В зависимости от установок `telnet` строки могут выводиться с эхо-повтором или не выводиться вообще. Случаются также неувязки при попытке выйти из сеанса.

14.4. Подключение к базам данных

Java обеспечивает подключение к базам данных через API-интерфейс JDBC (Java Database Connectivity). JDBC позволяет осуществить соединение с базой данных так же легко, как соединение с ресурсом или изображением. Java обеспечивает пересылку инструкций структурированного языка запросов SQL (Structured Query Language) и получение результатов запросов.

Для доступа к базе данных нужен драйвер. Если разработчик базы данных не предоставляет JDBC-драйвер, то наверняка для нее существует ODBC-драйвер (Open Database Connectivity). Фирма Sun обеспечивает мост ODBC-JDBC. Следовательно, чтобы загрузить драйвер, нужно использовать следующее описание:

Загрузка драйвера базы данных

```
try {
  Class.forName ("driver name");
} catch (ClassNotFoundException c) {
  System.out.println("Could not load database driver.");
}
```

Имя драйвера может выглядеть так — `sun.jdbc.odbc.Jdbc OdbcDriver`. Драйвер будет загружаться как часть класса `DriverManager`, используемого в следующем описании.

Теперь мы можем подключиться к базе данных, применяя драйвер.

Соединение с базой данных

```
Connection con = DriverManager.getConnection (url, user, password);
Connection con = DriverManager.getConnection (database);
```

Первый оператор описания используется для соединения с удаленной базой данных. Второй осуществляет доступ к локальной базе данных (см. пример ниже). Оба применяют драйвер, предварительно загруженный в объект `DriverManager`.

После установления соединения можно посылать предложения SQL и получить результат. Предложения формируются в виде строк (это означает, что корректность их SQL-синтаксиса вы должны обеспечить сами).

Посылка SQL-предложения базе данных

```
Statement stat = con.createStatement ();
int result = stat.executeUpdate("SQL insert");
ResultSet rs = stat.executeQuery("SQL query");
```

С использованием установленного соединения создается экземпляр объекта `Statement`. Метод `executeUpdate` выполняет SQL-предложение типа `INSERT` и возвращает код результата, из которого видно, успешным или нет было выполнение.
Метод `executeQuery` выполняет SQL-предложение типа `SELECT` и возвращает объект `ResultSet`. Результат может быть обработан в цикле строка за строкой.

Пример 14.5. База данных конвертера валют

Задача. Бюро путешествий Саванны решило, что курсы обмена валют для их программы по конвертированию лучше хранить в базе данных. Клиенты должны иметь доступ к базе данных, чтобы получать свежие котировки курса.

Решение. У нас уже есть программа конвертирования валют (см. примеры 6.8, 11.3 и 14.1), поэтому все, что нужно сделать, — это изменить ее так, чтобы осуществлять доступ к базе данных. На рис. 14.5 видно, что наша программа состоит из двух частей: одна часть предназначена для создания базы данных из существующего текстового файла, а вторая — собственно конвертер валют.

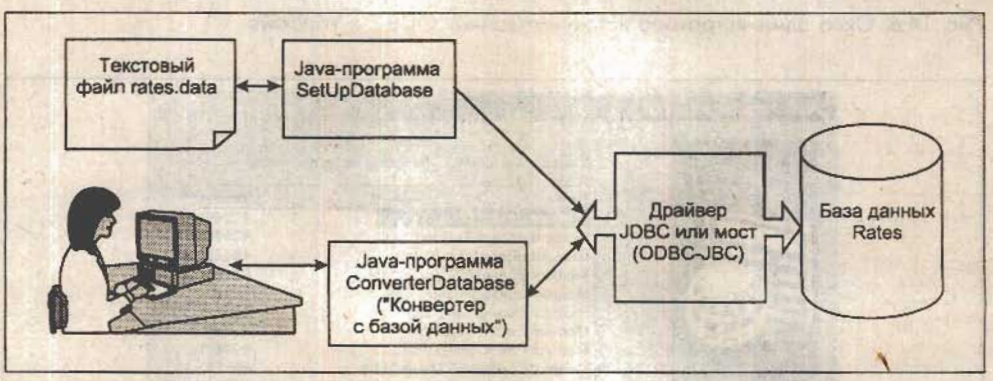


Рис. 14.5. Создание базы данных и доступ к ней

Предварительная установка необходимых компонентов. Воспользуемся одной из самый распространенных для Windows баз данных. Чтобы Java смогла загрузить соответствующий драйвер JDBC-ODBC, он должен быть активизирован. Делается это таким образом. Откройте панель управления Windows и дважды щелкните мышью на пиктограмме **ODBC Data Sources (32 bit)**. О открывшемся окне содержится список источников пользовательских данных (рис. 14.6). В нем должны быть записи MS Access Database и dBase Files. Вам нужно добавить в этот список вашу базу данных. Чтобы это сделать, щелкните мышью на кнопке **Add (Добавить)**, расположенной в правой части окна. Затем выберите драйвер, который будете использовать для доступа к базе данных. В нашем примере используется база данных MS Access, поэтому выбираем запись Microsoft Access Driver (*.mdb) (рис. 14.7).

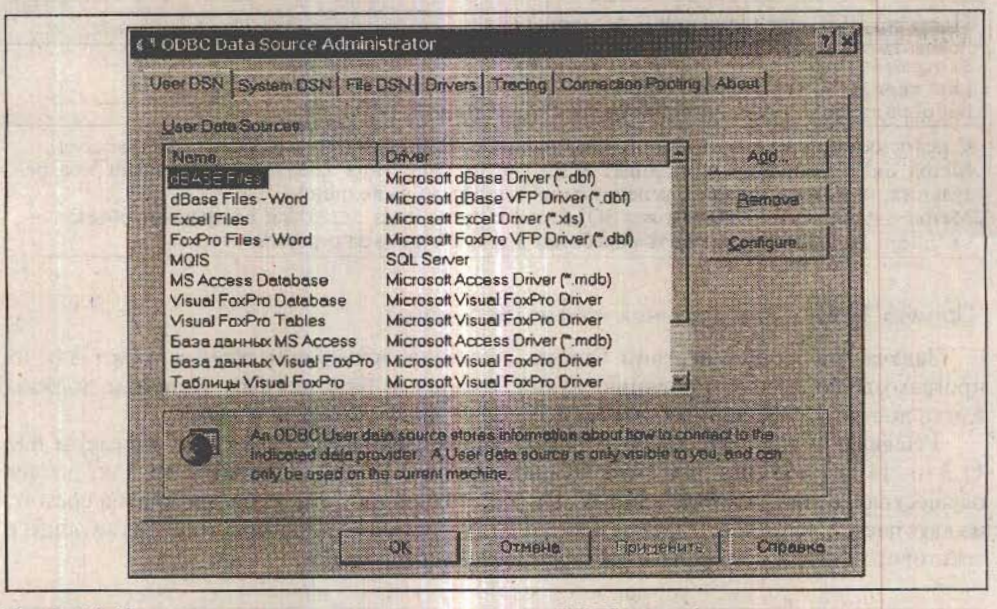


Рис. 14.6. Окно администратора источника данных ODBC в Windows

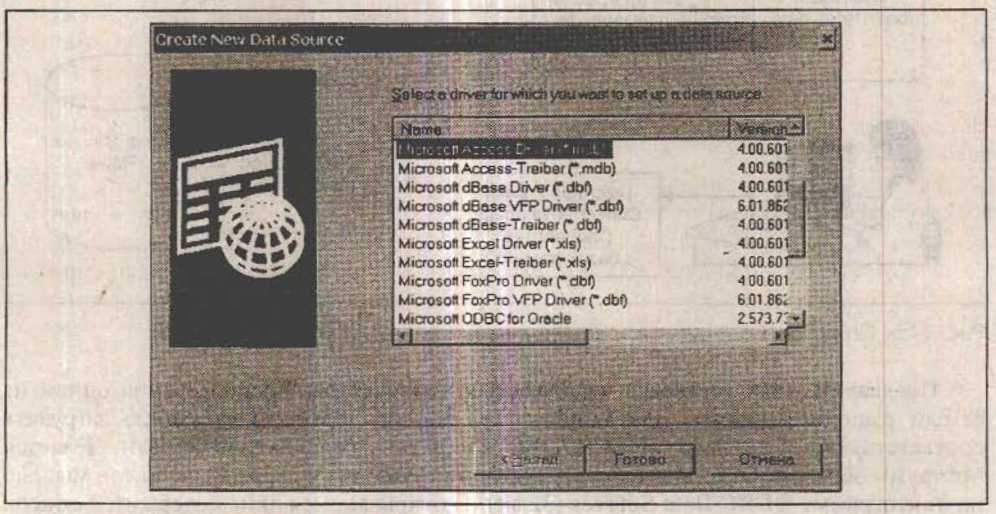


Рис. 14.7. Выбор драйвера для базы данных

После этого появляется еще одно диалоговое окно, в котором следует указать для вашего источника данных имя, описание и файл базы данных (рис. 14.8).

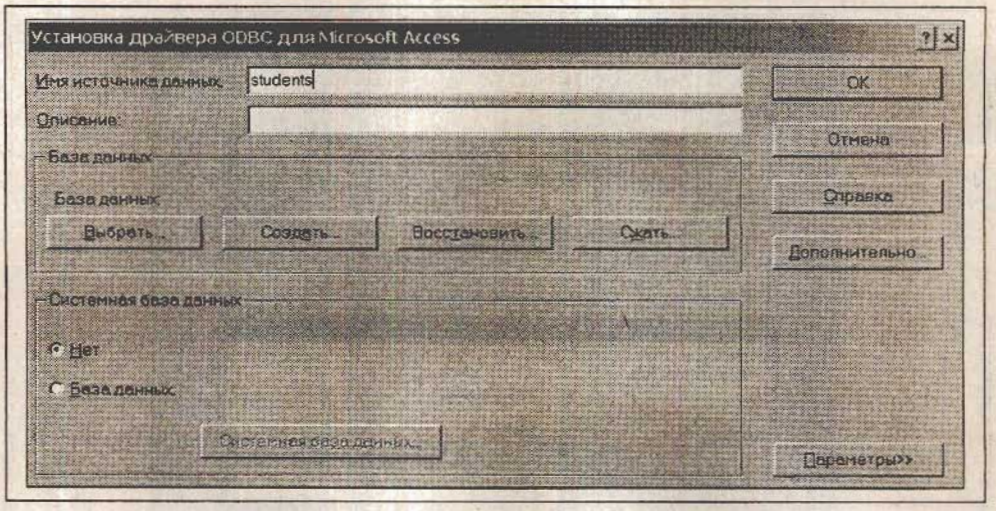


Рис. 14.8. Назначение имени для нового источника данных

Введите имя в поле **Data Source Name (Имя источника данных)** (на рис. 14.8 задано имя students, но вы укажите Rates, для примера `ConverterGUI`). Затем, если вы уже создали базу данных, то щелкните кнопке **Select (Выбрать)**, в противном случае щелкните на кнопке **Create (Создать)**. В обоих случаях это приведет к появлению диалогового окна открытия файла, в котором необходимо выбрать каталог и имя для базы данных. Если база данных уже создана, вы должны будете только выбрать файл базы данных и нажать кнопку **OK**. Если же это новая база данных, то перейдите в каталог, где она должна располагаться, и введите имя в поле **Database Name (Имя базы данных)**.

Если вы хотите ограничить доступ к базе данных, щелкните на кнопке **Advanced (Расширенная настройка)** в правой части диалогового окна и введите имя пользователя и пароль для доступа к базе данных (рис. 14.9). Эти имя и пароль нужно будет указывать в программе, однако для этого вам придется применить первый вариант конструктора `DriverManager.getConnection` (см. описание "Соединение с базой данных"). Затем вы можете сохранить все установки, щелкнув мышью на кнопках **OK** во всех диалоговых окнах.

Конструкция программы создания базы данных. Программа должна послать базе данных следующие SQL-предложения:

`DROP TABLE rates`	Удаляет таблицу `rates` со всеми ее данными из базы данных
`CREATE TABLE rates` (метаданные)	Создает таблицу `rates`, определяет имена столбцов и тип данных
`INSERT INTO rates VALUES` (значения)	Добавляет одну запись в таблицу

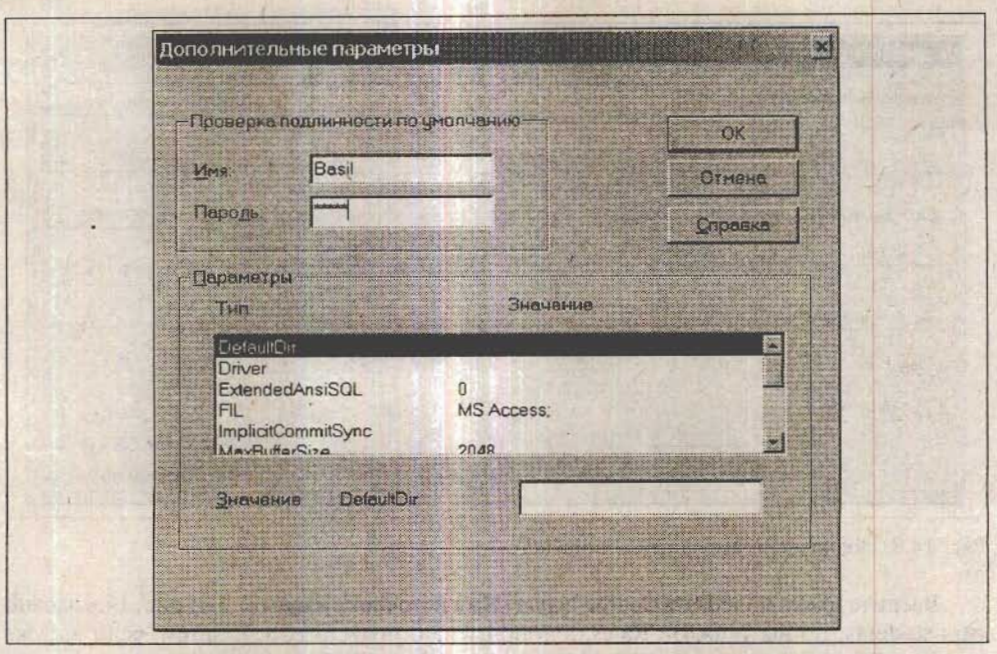


Рис. 14.9. Изменение или установка санкционированного доступа к базе данных

Ниже приведен текст программы, которая предназначена для удаления старой таблицы, создания новой таблицы и заполнения ее данными из текстового файла.

```
import java.sql.*;
import javagently.*;
import java.io.*;

class SetUpDatabase {

   /* Программа создания базы данных     Автор Б.Воролл, август 2000 г.
    * ==============================     откорректирована Дж.М. Бишоп
    *
    * Создает базу данных котировок курсов валют.
    * База данных использует интерфейс jdbc:odbc,
    * предназначенный для соединения с базами данных типа MS Access.
    */

  private Connection conn;

  SetUpDatabase() {
    // Загружает мост JDBC-ODBC.
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch (ClassNotFoundException c) {
      System.out.println("Система не может загрузить драйвер базы данных.");
    }

    // Считывает данные из текстового файла и загружает их
    // в базу данных.
```

```
    try {
      OpenConnection();
      WriteData();
    } catch (SQLException s) {
      System.out.println(s);
    } catch (IOException i) {
      System.out.println(i);
    }
  }

  void OpenConnection() throws SQLException {
    conn = DriverManager.getConnection("jdbc:odbc:Rates");
    Statement s = conn.createStatement();
    String state = "DROP TABLE rates";
    try {
      // Удаляет старые данные (если таких
      // данных нет, система выдаст объект-исключение
      // и выполнение программы продолжается).
      int temp = s.executeUpdate(state);
    } catch (SQLException q) {}

    // Создает таблицу в базе данных.
    state = "CREATE TABLE rates (Code VARCHAR(3), Country VARCHAR(80), " +
      "Curr VARCHAR(30), Conv DOUBLE)";
    int temp = s.executeUpdate(state);
  }

  void WriteData() throws IOException, SQLException {
    Stream fin = new Stream("rates.data", Stream.READ);
    try {
      for (; ;) {
        Rates r = new Rates();
        r.setRate(fin);
        // Добавляет данные в таблицу rates.
        Statement s = conn.createStatement();
        String state = "INSERT INTO rates VALUES ('" +
                       r.code + "', '" +
                       r.country + "', '" +
                       r.currency + "', " +
                       r.conversion + ")";
      int rs = s.executeUpdate(state);
      }
    } catch (EOFException e) {}
  }

  public static void main (String [] args) {
    new SetUpDatabase();
  }
}
```

Программа последовательно читает строки из файла `rates.data` и добавляет записи в базу данных. При этом применяется формат с разделителями-запятыми.

Изменение программы конвертирования. Главное отличие программы `ConverterDatabase` от написанной ранее состоит в том, что в новой версии нет хеш-таблицы. Однако в ней по-прежнему присутствует часть, отвечающая за ввод данных, поскольку программа должна выбрать из базы данных все названия стран, которые нужно поместить в два раскрывающихся списка. Вот код, позволяющий это сделать.

```
conn = DriverManager.getConnection("jdbc:odbc:Rates");

// Выполняет SQL-запрос для получения всех данных из таблицы.
Statement s = conn.createStatement();
String command = "SELECT Country FROM rates";
ResultSet rs = s.executeQuery(command);

while (rs.next()) {
  String temp = rs.getString(1).trim();
  toChoice.addItem(temp);
  fromChoice.addItem(temp);
}
```

Вы видите, что переход на следующий объект `ResultSet` осуществляется с помощью вызова метода `rs.next()`. Метод `transaction` в целом похож на своего предшественника, с тем отличием, что вместо операторов доступа к хеш-таблице в нем используются следующие операторы:

```
Statement s1 = conn.createStatement();
String command = "SELECT Country,Curr,Conv FROM rates " +
  "WHERE Country = '" + fromCountry +"'";
```

Полный текст программы конвертера приведен ниже.

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import java.sql.*;
import javagently.*;

class ConverterDatabase extends Frame
  implements ActionListener, ItemListener {

   /*          Программа
    * "Конвертер с базой данных"        Автор Дж.М. Бишоп, декабрь 1998 г.
    * --------------------------  версия с классом Display, июль 1999 г.
    *                                        GUI-версия, июль 1999 г.
    *                          версия с базой данных Б.Воролл, август 2000 г.
    * Содержит курсы обмена одной валюты на ряд других
    * и производит "конвертацию" вводимой суммы.
    *
    * Демонстрирует использование драйвера jdbc
    * для соединения с базой данных.
    */

  private Connection conn;

  Choice    fromChoice, toChoice;
  TextField amountField;
  TextArea  resultField;
  int       amount = 1000;
  String    toCountry, fromCountry;
  Button    goButton;

  public ConverterDatabase () throws SQLException {
    // Вначале создает GUI-интерфейс, но не делает его
```

```
    // видимым (не назначает setVisible(true)).
    Panel p = new Panel (new BorderLayout());

      // Левая панель.
      Panel q = new Panel();
        q.add ("North",new Label ("Из"));
        fromChoice = new Choice();
          fromChoice.addItemListener (this);
          q.add("Center",fromChoice);
        p.add ("West", q);
      // Правая панель.
      q = new Panel();
        q.add ("North",new Label ("В"));
        toChoice = new Choice();
          toChoice.addItemListener (this);
          q.add("Center",toChoice);
        p.add ("East",q);
      // Центральная панель.
      q = new Panel(new BorderLayout());
        Panel r = new Panel();
          r.add(new Label("Сумма"));
          amountField = new TextField("1000  ");
            amountField.addActionListener(this);
            r.add(amountField);
            q.add("North",r);
          resultField = new TextArea(8,20);
            q.add ("Center",resultField);
          goButton = new Button ("Конвертировать");
            goButton.addActionListener(this);
            q.add("South",goButton);
        p.add("Center",q);
      add(p);
      setTitle("Конвертер валют");
      setSize(610,300);
      addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
          System.exit(0);
        }
      });

      // Вызывает функцию readIn для открытия (установления) соединения
      // с базой данных и для заполнения раскрывающихся списков
      // названиями стран.
      readIn();
    }

    public void actionPerformed (ActionEvent e) {
      if (e.getSource() == amountField)
        amount = (int) Integer.parseInt(amountField.getText().trim());
      else
      if (e.getSource() == goButton)
        transaction();
    }

    public void itemStateChanged(ItemEvent e) {
      String s = (String) e.getItem();
      if (e.getItemSelectable() == fromChoice)
```

```
      fromCountry = s;
    else toCountry = s;
  }

  void transaction () {
    // Вызывается при нажатии кнопки Конвертировать.
    try {
      // Создает оператор для получения всех данных,
      // относящихся к стране, валюту которой нужно конвертировать
      // (fromCountry). Затем выполняет выборку данных, сохраняя результаты в rs.
      Statement s1 = conn.createStatement();
      String command = "SELECT Country,Curr,Conv FROM rates " +
        "WHERE Country = '" + fromCountry +"'";
      ResultSet rs = s1.executeQuery(command);
      rs.next();

      // То же самое, что и выше, но для страны, в валюту которой
      // производится конвертирование (toCountry); результаты
      // сохраняются в rs2.
      Statement s2 = conn.createStatement();
      command = "SELECT Country, Curr, Conv FROM rates " +
        "WHERE Country = '" + toCountry + "'";
      ResultSet rs2 = s2.executeQuery(command);
      rs2.next();

      // Добавляет текст в область вывода resultField.
      // Заметьте, как используются методы доступа к базе данных
      // в сравнении с применяемым для хеш-таблиц методом get.
      resultField.append(amount+" "+rs.getString(1)+" "+
        rs.getString(2)+
        "\n в "+rs2.getString(1)+" "+rs2.getString(2)+"\n равны "+
        Stream.format(amount/rs.getDouble(3)*
          rs2.getDouble(3),10,3)+"\n\n");
    } catch (SQLException q) {
      q.printStackTrace();
      System.exit(0);
    }
  }

  void readIn() throws SQLException {
    // Вызывается в конце конструктора.
    // Открывает соединение с базой данных.
    conn = DriverManager.getConnection("jdbc:odbc:Rates");

    // Выполняет SQL-запрос для получения из таблицы всех данных.
    Statement s = conn.createStatement();
    String command = "SELECT Country FROM rates";
    ResultSet rs = s.executeQuery(command);
     // В цикле извлекаем строки таблицы и добавляем
    // все названия стран в раскрывающиеся списки.
    while (rs.next()) {
    String temp = rs.getString(1).trim();
      toChoice.addItem(temp);
      fromChoice.addItem(temp);
    }

    // Теперь мы можем сделать окно видимым.
```

```
    setVisible(true);
  }

  public static void main(String[] args) {

    // Загружает мост JDBC-ODBC.
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch (ClassNotFoundException c) {
      System.out.println("Ошибка при загрузке драйвера JDBC-ODBC.");
       System.exit(0);
    }
    try {
       new ConverterDatabase();
    } catch (SQLException s) {
       s.printStackTrace();
    }
  }
}
```

Тестирование. При тестировании программы вы обнаружите, что исходная программа и ее новая версия с базой данных выводят на экран одинаковые GUI-компоненты и результаты. Различие состоит в том, что в новой версии администратор может обновить файл `rates.data` и запустить программу `SetUpDatabase` в то время, когда программа `ConverterGUI` не работает.

14.5. Доступ к удаленным объектам

Сокеты обеспечивают передачу произвольных данных от одной машины к другой, а при подключении к базам данных мы имеем возможность передавать структурированные данные. Между этими возможностями находится еще один часто используемый метод передачи данных. Он состоит в том, что данные передаются в соответствии с требованиями программиста, а не базы данных. Очевидно, что в Java такие данные будут представлены в виде объектов.

Иногда нежелательно пересылать данные по сети (например, из-за большого объема или соображений безопасности). В этом случае можно создать объект на удаленном компьютере, получить к нему доступ и, вызывав его методы, выполнить нужные операции.

Такая модель в точности соответствует модели клиент-сервер. Сервер хранит объект, к которому клиент хочет получить доступ. Согласно требованиям Java, на сервере объект нужно зарегистрировать, или привязать (от англ. rebind — привязывать). Для этого в реестре указывают имя объекта и название службы (см. ниже описание "RMI-привязка"). Кроме того, клиенту нужно знать, к какому компьютеру он должен подсоединиться, и имя, под которым был зарегистрирован объект. Только в этом случае клиент сможет отыскать объект и получить на него ссылку. Далее объект может использоваться так же, как локальный. Заботу обо всех протоколах, передаче и проверке данных в обоих направлениях берет на себя *интерфейс вызова удаленных методов* RMI (remote method invocation). Пакет `java.rmi` является стандартным пакетом JDK и поэтому доступен во всех системах Java. В следующих описаниях показано, как на сервере объявляется удаленный объект и как выполняются его привязка и поиск.

Объявление удаленного объекта

```
// общее для сервера и его клиентов
interface remoteInterfaceid {
... методы
}
// на сервере
class remoteClassid extends UnicastRemoteObject
        implements remoteInterfaceid {
... реализация методов
}
```

Класс, имеющий имя *remoteClassid*, является частью системы RMI и создает класс объектов, образующих интерфейс.

RMI-привязка

```
RemoteClassid remoteObjectid;
try {
  Naming.rebind ("Название службы", remoteObjectid);
catch (Exception e) {
  System.out.println("Нет в реестре.\n"+e);
}
```

Название службы вводится в реестр сервера наряду со ссылкой на удаленный объект, который в описании указан как remoteObjectid. Удаленный объект должен быть классом, который реализует интерфейс Remote.
Любые уже существующие службы с таким же именем будут переопределены. Чтобы этого избежать, используйте метод bind вместо rebind.
Реестр должен быть доступен.

RMI-поиск

```
RemoteInterfaceid localObjectid;
try {
  localObjectid = (RemoteInterfaceid)
  Naming.lookup (url + "Название службы");
}
catch (Exception e) {
  System.out.println("Нет такой службы или реестр недоступен\n"+e);
}
```

Клиент "видит" название удаленной службы только через общий интерфейс с сервером — *RemoteInterfaceid*.
Клиент ищет Название службы в реестре, чтобы в переменной *localObjectid* получить ссылку на удаленный объект.
Если реестр или конкретная служба не доступны, система генерирует объект-исключение.

После успешного завершения поиска переменная *localObjectid* содержит прямую ссылку на объект *remoteObjectid*, который теперь можно использовать, как любой локальный объект.

Пример 14.6. Сервер, генерирующий "магические числа"

Задача. Продемонстрировать применение RMI на простой программе.

Решение. Имеется удаленный объект, генерирующий случайные числа. Клиенты могут обратиться к нему и получить свои "магические числа".

Конструкция. Таким образом, мы имеем сервер на одном компьютере и произвольное количество клиентов на других компьютерах (или на том же). Обе стороны используют пакет `java.rmi`. Сервер осуществляет привязку объекта в реестре, а клиент выполняет его поиск. При разработке системы нужно пройти следующие этапы.

1. Определить класс `MagicNumber`, из которого будут создаваться экземпляры удаленных объектов.
2. Определить соответствующий интерфейс `Transferable`, который использовал бы клиент, поскольку клиент не может непосредственно видеть класс `MagicNumber`.
3. Определить класс `MagicNumberServer` на сервере.
4. Определить класс `MagicNumberClient` на клиентском компьютере.
5. Скомпилировать серверные компоненты, после чего применить к ним программу `rmic`, чтобы создать соответствующие модули и заготовки (stubs and skeletons). Удаленный доступ к объекту `MagicNumberServer` будет осуществляться посредством этих модулей и заготовок.
6. Запустить программу `rmiregistry`. Она создает реестр, к которому обращаются клиенты.
7. Запустить программу-сервер, а затем программы-клиенты.

Программа. Прежде всего определим класс объекта.

```
import java.rmi.*;
import java.rmi.server.*;

/*  Удаленный объект   Автор Дж.М. Бишоп, октябрь 1997 г.
 *  ================             Java 1.2, август 2000 г.
 *
 * Эти объекты могут быть зарегистрированы на одном компьютере,
 * а с помощью rmi-протокола к ним можно осуществить доступ через
 * Internet с других машин. Компиляция состоит из двух этапов:
 *    javac MagicNumber.java
 *    rmic MagicNumber
 * Последняя команда создает для удаленного объекта заготовки и
 * фиктивные * модули. Чтобы сделать их доступными через сеть,
 * введите в окне № 1:
 *    rmiregistry
 * Эта команда запускает реестр.
 * Затем активизируйте обе части системы — программу-сервер
 * и программу(ы)-клиент(ы).
 */
class MagicNumber extends UnicastRemoteObject implements Transferable {

  MagicNumber () throws RemoteException {
  }

  public String getNumber () throws RemoteException {
    return "Ваше \"магическое число\": "+ (int) (Math.random()*100);;
  }

}
```

Теперь определим интерфейс `Transferable`, который расширяет интерфейс `Remote`:

```
import java.rmi.*;
```

```
interface Transferable extends Remote {
  String getNumber () throws RemoteException;
}
```

Код программы-сервер имеет следующий вид:

```
import java.rmi.*;

public class MagicNumberServer {

  /* Система клиент-сервер
   *  "Магическое число"              Автор Дж.М. Бишоп, август 2000 г.
   * ===================
   * адаптирована с применением системы ThingServer, октябрь 1997 г.
   *
   * Демонстрирует работу системы клиент-сервер, используя RMI.
   * Выполняется при вводе в окне сервера команды:
   *    java MagicNumberServer
   */

  public static void main (String args []) throws RemoteException {

    MagicNumber number = new MagicNumber ();

    try {
      Naming.rebind ("Служба \"Магическое число\"", number);
    }
    catch (Exception x) {
      System.out.println("Невозможно выполнить привязку, или нет такого хоста"+x);
    }
    System.out.println("Работает служба \"Магическое число\"");
  }
}
```

И в завершение приводим код программы-клиента:

```
import java.rmi.*;
import java.rmi.server.*;

public class MagicNumberClient {
  /* Программа-клиент          Автор Дж.М. Бишоп, октябрь 1997 г.
   * ================          Java 1.2 обновлена август 2000 г.
   *
   * Чтобы активизировать программу-клиент, введите в окне клиента:
   * java MagicNumberClient
   * что по умолчанию относится к текущей машине.
   * Если программа-клиент выполняется на другой машине
   * (не на той, где выполняется программа-сервер), введите:
   * java MagicNumberClient serverid
   */

  public static void main (String args []) {

  String computer;
  if (args.length == 0) computer = "127.0.0.1";
  else computer = args[0];
```

```
    try {
      Transferable t = (Transferable) Naming.lookup
          ("rmi://"+computer + "/Служба \"Магическое число\"");
      System.out.println(t.getNumber());
    }
    catch (Exception e) {
      System.out.println("Ошибка "+e);
    }
  }
}
```

Поскольку работа с магическими числами выполняется через сеть, они нуждаются в особом контроле. Поэтому класс `MagicNumber` импортирует пакет `rmi` и подпакет `rmi.server` и указывает в своих методах и конструкторе, что система может генерировать объект-исключение `RemoteException`, например в том случае, если сервер прекратит работу.

Тестирование. Система можно протестировать на одном компьютере, запустив сервер и клиентов в разных окнах, или на разных компьютерах. Результат выполнения будет одним и тем же. Необходимо учесть, что процесс запуска RMI-программ имеет некоторые особенности. Сначала на сервере требуется запустить реестр, выполнив следующую команду:

```
rmiregistry
```

Затем вы должны вызвать программу-сервер:

```
java MagicNumberServer
```

Теперь наступает очередь клиентов. Если мы введем в новом окне команду

```
java MagicNumberClient
```

система вернет строку, подобную этой:

```
Ваше "магическое число": 69
```

Более полезный с практической точки зрения пример использования RMI мы рассмотрим в следующем разделе "Практикум 11".

14.6. Практикум 11. Информационная система аэропорта

Компания "Аэропорт Саванны" решила автоматизировать работу, ранее выполнявшуюся диспетчером, отвечающим за объявления информационного характера. Эти объявления поступают из двух источников: от авиакомпаний и компании аэропорта. Авиакомпании подают объявления, касающиеся проверки билетов и багажа, задержек вылета и прибытия и т.д. Аэропорт отвечает за сообщения, относящиеся к безопасности, которые обычно автоматически посылаются с определенными интервалами. Следовательно, мы будем разрабатывать систему, приблизительная схема которой показана на рис. 14.10. В функции монитора входит распространение сообщений при помощи громкоговорителей и информационных табло, расположенных по всему аэропорту. Чтобы не усложнять программу, будем использовать простой монитор, который выводит текстовые сообщения.



Рис. 14.10. Схема работы информационной системы аэропорта

Апплеты для авиакомпаний

Система предоставляет авиакомпаниям возможность доступа к набору апплетов. Каждый апплет создает экран и запрашивает информацию, которая затем используется при составлении сообщения. Например, апплет `Checkin` запрашивает номер рейса и пункт назначения, а затем формирует следующее сообщение:

"Пассажиры, которые еще не прошли регистрацию на рейс SA507 на Киев, пожалуйста, покиньте свои очереди и пройдите непосредственно на пункты регистрации"

Другие апплеты выводят сообщения о посадке (апплет `Board`), задержке вылета (апплет `Delay`) и т.д. В каждом случае объявляется только необходимая информация. На рис. 14.11 показан апплет `Checkin` со своим сообщением.



Рис 14.11. Апплет `Checkin` для информационной системы аэропорта с выведенным внизу монитором

Окно апплета размещается в правой части страницы, в левой ее части находится перечень ссылок для вызова имеющихся апплетов. При вызове любой страницы в правую ее часть выводится фоновое изображение, как показано на рис. 14.12.



Рис. 14.12. Первый экран информационной системы аэропорта с объявлением, касающимся безопасности пассажиров

HTML-файлы для авиакомпаний

Апплеты для авиакомпаний выполняются в броузерах, которые осуществляют вывод рамок, меню и фоновых рисунков. Файл `index.html` для авиакомпаний выглядит так:

```
<HTML>
  <HEAD>
    <TITLE>Аэропорт</TITLE>
  </HEAD>

  <FRAMESET COLS="215,*">
  <FRAME SCR="Contents.html" NORESIZE SCROLLING="NO" NAME="Contents">
  <FRAME SCR="Applets.html" NAME="Applets">
  </FRAMESET>

</HTML>
```

В левый фрейм выводится файл `Contents.html`, который содержит следующий HTML-код:

```
<HTML>
  <HEAD>
    <META NAME="Author" CONTENT="JM Bishop">
    <TITLE>Содержание</TITLE>
  </HEAD>
  <BODY TEXT="#CE5831" BGCOLOR="#F1D7BE" LINK="#EF5310"
       VLINK="#974D0F" ALINK="#EF5310" SCROLLING="NO">
```

```
  <CENTER>
  <H1>
  <I><FONT COLOR="#00CC00"><FONT SIZE=+3>
     Авиалинии Саванны</FONT></FONT></I></H1>

  <H2>
  <FONT FACE="Arial, Helvetica"><FONT COLOR="#00CC00"><FONT SIZE=+1>
     Объявления</FONT></FONT
  </H2>

  <FONT COLOR="#E17100"
  <A HREF="Checkin.html" target="Applets">Регистрация</A>
  <A HREF="Board.html" target="Applets">Посадка</A>
  <A HREF="Last.html" target="Applets">Завершение посадки</A>
  <A HREF="Page.html" target="Applets">Страница пассажира</A>
  <A HREF="Delay.html" target="Applets">Задержка вылета</A>
  </FONT><BR>
  <BR>
  <H2>
  <FONT COLOR="#00CC00"><FONT SIZE=+1>
  <A HREF="Applets.html" target="Applets">Продолжить</A>
  </FONT>

   <I><FONT SIZE=+0>"Авиалиния "Gentle""
  </FONT></FONT></FONT></I></H2>
  </CENTER>
      
  </BODY>
</HTML>
```

Тег `target` в HTML-ссылке указывает, в какой фрейм должна быть загружена страница. Мы объявили два фрейма, `Contents` и `Applets`. Поэтому в приведенном выше коде указывается, что страницы, вызывающие апплеты, загружается в правую часть, т.е. во фрейм `Applets`.

Заметьте, что при активизации ссылки **Продолжить** в правой рамке отображается первоначальный фоновый рисунок. Каждая HTML-ссылка указывает на очень простой HTML-файл, который выводит изображение фона и вызывает соответствующий апплет. Так, содержимое файла `Checkin.html` выглядит следующим образом:

```
<HTML>
  <HEAD>
    <TITLE>Регистрация</TITLE>
  <HEAD>

  <BODY TEXT="#CE5831" BGCOLOR="#F1D7BE" LINK="#A76330"
      VLINK="#974D0F" ALINK="#EF5310" BACKGROUND="Genishot.jpg"
      SCROLLING="NO">

  <APPLET CODE="Checkin.class" WIDTH=260 HEIGHT=120></APPLET>
  </BODY>
</HTML>
```

Теперь можно добавить в систему апплеты и серверы.

Апплеты

В апплетах предусмотрен ввод данных в текстовые поля. Чтобы сделать интерфейс более наглядным, вместо менеджера размещения `GridLayout` используем менеджер размещения `GridBaglayout` (рис. 14.11). Применять его не сложно. Размещение компонентов пользовательского интерфейса менеджером `GridBaglayout` осуществляется так, что левый компонент занимает требуемое ему пространство, а правый — получает его оставшуюся часть (в нашем случае большую). Такой метод очень удобен для размещения пар "метка-текстовое поле". Полный текст апплета `Checkin` выглядит следующим образом:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Checkin extends Applet implements ActionListener {

  /*  Апплет для компоновки и вывода сообщения о регистрации
   *                              Авторы Л.Бота и Дж.М. Бишоп
   *  ====================        Java 1.1, январь 1998 г.
   * Апплет берет введенные в текстовые поля данные,
   * формирует сообщение и передает его
   * rmi-серверу, который должен быть уже запущен.
   */

  private Button sendButton;
  private TextField flight, to;
  private String message;

  public void init() {
    // В качестве цвета выбран особый оранжевый оттенок,
    // имеющийся в Java Gently, он указан как
    // шестнадцатеричное число.
    setBackground(new Color(0xF1D7BE));
    setLayout(new BorderLayout());

    add("North",new Label("ОБЪЯВЛЕНИЕ О РЕГИСТРАЦИИ",Label.CENTER));

    Panel p = new Panel();
      // Мы используем GridBagLayout, чтобы обеспечить
      // удобное размещение меток и текстовых полей.

      GridBagLayout layout = new GridBagLayout();
      p.setLayout(layout);
      GridBagConstraints c = new GridBagConstraints();
      c.gridwidth = 1; c.gridheight = 1;
      Label flightLabel = new Label("Номер рейса");
        c.gridx = 0; c.gridy = 0;
        layout.setConstraints(flightLabel, c);
        p.add(flightLabel);
      Label toLabel = new Label("Пункт назначения");
        c.gridx = 0; c.gridy = 1;
        layout.setConstraints(toLabel, c);
        p.add(toLabel);
```

```
        flight = new TextField("", 10);
          flight.addActionListener(this);
          c.gridx = 1; c.gridy = 0;
          layout.setConstraints(flight, c);
          p.add(flight);
        to = new TextField("", 10);
          to.addActionListener(this);
          c.gridx = 1; c.gridy = 1;
          layout.setConstraints(to, c);
          p.add(to);

        add("Center",p);

      Panel q = new Panel();
        sendButton = new Button("Отправить");
          sendButton.addActionListener(this);
          q.add(sendButton);
          q.add(new Label("затем щелкните на ссылке \"Продолжить\""));
      add("South",q);
    }

  public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source == sendButton) {
      message =
        " Пассажиры, которые еще не прошли "
      + "регистрацию на рейс " + flight.getText() + " на "
      + to.getText() + " пожалуйста, покиньте свои очереди и "
      +"пройдите непосредственно на пункты регистрации"
      + " под номером 35 или 36.";
      AnnouncerClient.sendMessage(message);
      flight.setText("");
      to.setText("");
    }
  }
}
```

Соединение посредством RMI

Подразумевается, что в работе системы будет задействовано несколько компьютеров, как минимум три — компьютер авиакомпании, сервер объявлений с программой-монитором и компьютер аэропорта — плюс по одному для каждой дополнительной авиакомпании. Как же они будут взаимодействовать? Конечно, можно создать сокеты, однако RMI предоставляет более элегантное решение, к тому же на удивление легкое в реализации и использовании.

Схема взаимоотношений между различными классами, необходимыми нашей системе, представлена на рис. 14.13. Апплеты для авиакомпаний входят в состав программы-клиента, которая передает сообщение локальному клиенту, находящемуся на той же машине. Работа клиента компании аэропорта базируется на работе сервера объявлений (`AnnouncerServer`), потому что клиент активизируется тогда, когда этого "захочет" сервер объявлений. После активизации клиент вызывает сервер компании для получения подходящего сообщения.

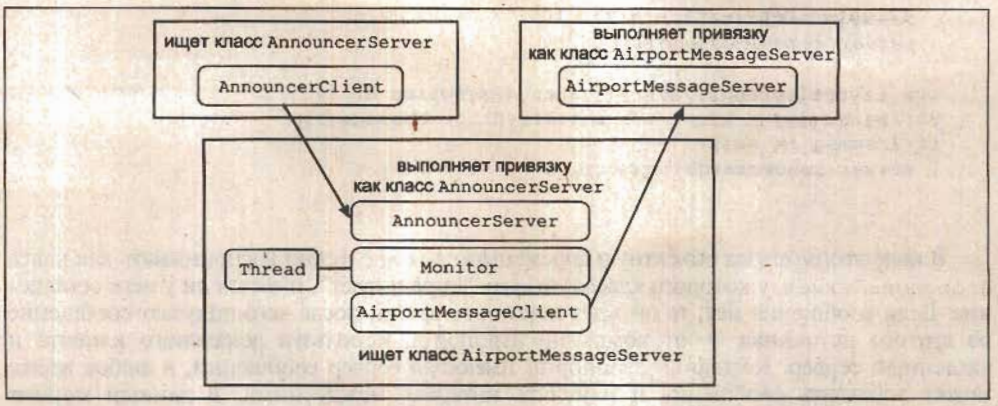


Рис. 14.13. Схема информационной системы аэропорта, раскрывающая характер взаимодействия между различными компьютерами (классами)

Апплет вызывает своего клиента следующим образом:

```
AnnouncerClient.sendMessage (message);
```

Затем клиент выполняет поиск удаленного интерфейса на сервере и передает сообщение. Это представлено следующими операторами:

```
Announcer an = (Announcer)Naming.lookup
    ("rmi://localhost/AnnouncerServer");
an.sendMessage(message);
```

Серверная версия метода `sendMessage` сохраняет сообщения в массиве. Это необходимо делать, поскольку сообщения могут поступать быстрее, чем монитор сможет их отобразить. Сервер играет пассивную роль, он получает сообщения, а затем отдает их по запросу монитора.

Монитор является отдельным подпроцессом и, следовательно, может работать в то время, когда сервер принимает очередные сообщения. Монитор получает сообщения от нескольких источников. Если сервер объявлений имеет сообщения, монитор получает одно из них, в противном случае он ждет несколько секунд, после чего опрашивает клиента аэропорта о наличии сообщения и принимает его. Метод монитора `run` является активным и выглядит следующим образом:

```
public void run() {
  boolean startdelay;
  while (true) {
    startdelay = true;
    if (server.hasMessage()) {
      String msg = server.getMessage();
      while (msg.length() > 0) {
      String displaymsg = formatString(msg);
      message.setText("");
      message.setText(displaymsg);
      if (startdelay == true) {
      try { Tread.sleep(1000); } catch (Exception e) {}
        startdelay = false;
      }
      try { Tread.sleep(200); } catch (Exception e) {}
        msg = msg.substring(1);
      }
```

```
        message.setText("");
        server.removeMessage();
      }
      try { Tread.sleep(5000); } catch (Exception e) {}
      String newmsg = AirportMessageClient.getMessage();
      if (newmsg != null)
        server.sendMessage(newmsg);
    }
}
```

В коде этого метода объектную переменную `server` нужно воспринимать как класс `AnnouncerServer`, у которого класс `Monitor` "спрашивает", имеется ли у него сообщение. Если сообщения нет, то он ждет некоторое время, после чего получает сообщение из другого источника — от компании аэропорта, используя локального клиента и удаленный сервер. Компания аэропорта, имеющая сервер сообщений, в любое время может добавлять сообщения и изменять интервал между ними. В данный момент определены следующие сообщения, которые периодически выводятся на монитор:

Внимание, это объявление, касающееся безопасности. Не оставляйте ни на минуту ваш багаж без присмотра. Вещи, оставленные без присмотра, будут изъяты и уничтожены.

Детям не разрешается играть на эскалаторах или в лифтах.

Компания приветствует вас в нашем аэропорту и гарантирует вам приятное и безопасное путешествие.

Схема информационной системы аэропорта приведена на рис. 14.14. Соответствующие коды программ вы можете найти на Web-узле.

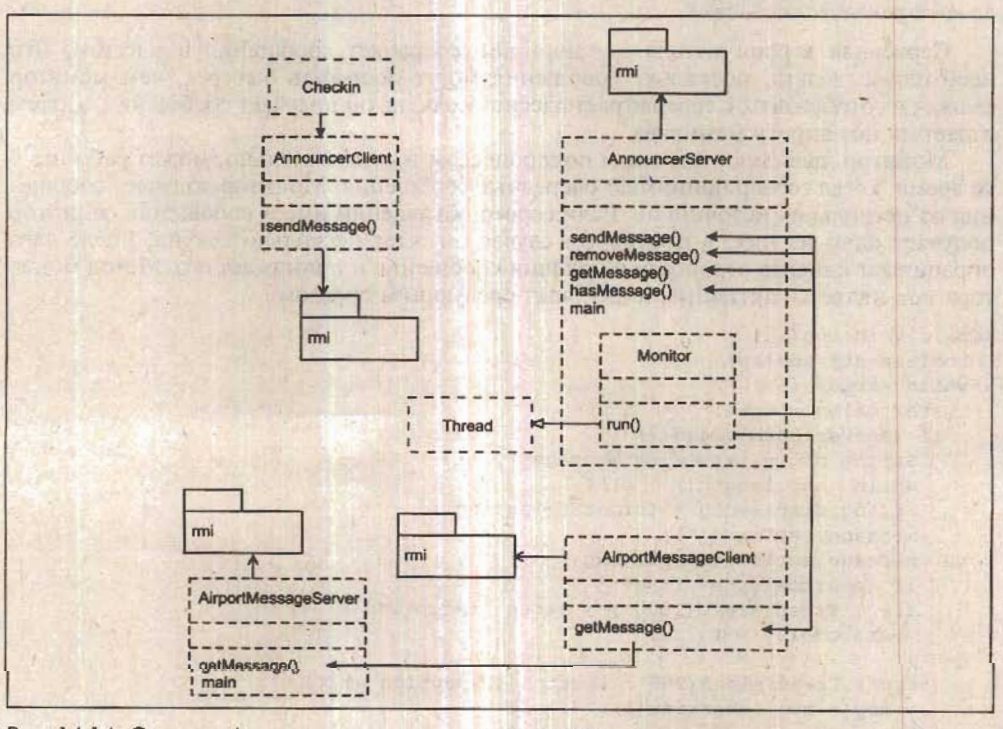


Рис. 14.14. Схема информационной системы аэропорта

Запуск программы

Чтобы система заработала, нужно запустить три процесса и броузер. В первом окне посредством ввода приведенной ниже команды запускается реестр RMI:

```
rmiregistry
```

В следующих двух окнах активизируются два сервера, которые выполнят привязку (самих себя) к реестру:

```
java AnnouncerServer

java AirportMessageServer
```

Затем в броузере необходимо открыть файл `index.html`. Его вид представлен на рис. 14.12. После щелчка мышью на ссылке "Регистрация" вы увидите изображение, представленное на рис. 14.11, и т.д. Класс `AnnouncerServer` создает окно монитора, в котором бегущей строкой выводятся сообщения.

Резюме

Java предоставляет различные способы установления сетевых соединений: при помощи URL, с использованием портов и сокетов, подключение к базам данных и удаленным объектам. Каждый способ поддерживается специальными классами и библиотеками, которые являются частью стандартного языка Java.

В этой главе мы ознакомились с некоторыми новыми пакетами — `java.net`, `java.sql`, `java.rmi` и `java.rmi.server`, тут же были представлены их классы и методы.

```
URL                     Class
  openConnection          forName
  getContent            DriverManager
URLConnection             getConnection
  getInputStream        Connection
Image                     createStatement
  createImage           Statement
ImageProducer             executeUpdate
ServerSocket              executeQuery
  accept                ResultSet
Socket                  Naming
InetAddress               rebind
  getHostName             lookup
  getLocalHost
```

Следующий перечень включает темы, на которые в главе обращено особое внимание:

- создание URL;
- создание URL-соединения;
- отображение рисунков;
- создание сокетов;
- потоки для сокетов;
- загрузка драйвера базы данных;
- подключение к базе данных;
- посылка SQL-предложения базе данных;
- объявление удаленного объекта;

- RMI–привязка;
- RMI-поиск.

В главе также представлено несколько новых объектов-исключений:

```
MalformedURLException
ClassNotFoundException
SQLException
RemoteException
```



Вопросы

14.1. Если `elephant.jpg` — это файл изображения, находящийся в том же каталоге, что и выполняющийся в данный момент апплет, то для доступа к этому файлу используется оператор:

а) `Image e = elephant.getImage();`

б) `Image e = getImage(this, elephant);`

в) `Image e = getImage(getCodeBase(), "elephant.jpg");`

г) `Image e = (MediaTracker) getImage("elephant.jpg");`

14.2. К какому классу принадлежит переменная `imagename`, указанная в следующем операторе:

```
a.b = a.createImage((ImageProducer)imagename.getContent());
```

а) `URL`

б) `ImageProducer`

в) `Image`

г) `Object`

14.3. В примере 14.4 (ATM-сервер) подпроцесс `handler`, получив специальное сообщение, может вызвать метод `server.closedown()`. Когда сервер будет реагировать на данное сообщение?

а) немедленно

б) после завершения цикла `while`

в) после того, как с сервером соединится следующий клиент

г) варианты в и г

14.4. Какое различие между сокетом и сокетом сервера в API Java?

а) сокет находится на клиенте, а сокет сервера на сервере

б) оба находятся на сервере, но сокет сервера может прослушивать новых клиентов

в) `ServerSocket` — это класс, а `socket` — объект

г) сокеты сервера для установления соединения прослушивают новых клиентов и сокеты, находящиеся на обеих сторонах

14.5. Какой тег предназначен для вывода изображения?

а) `<IMAGE = ...>`

б) `<IMG SRC = ...>`

в) `<IMG = ...>`

г) `<DISPLAY = ...>`

14.6. Предположим, что в примере 14.6 имеется еще один класс под названием `Secret`, который также реализует интерфейс `Transferable`. Если опустить обработку исключительных ситуаций, то оператором для привязки объекта `s` класса `Secret` в реестре RMI будет:

а) `Naming.rebind (Secret, s);`

б) `Naming.rebind ("Secret Service", s);`

в) `s.rebind ("Secret Service");`

г) `Naming.rebind (s);`

14.7. Класс `Socket` объявлен в пакете Java, который называется:

а) `java.lang`

б) `java.net`

в) `java.util`

г) `java.rmi`

14.8. Если `con` — соединение с базой данных, то оператором для создания объекта, выполняющего SQL-запрос, будет:

а) `String s = con.readln();`

б) `Statement s = con.createStatement();`

в) `Statement s = con ("SELECT");`

г) `String s = con.create("SELECT");`

14.9. Прежде чем мы создавать на языке Java программы для работы с базами данных, мы должны:

а) назначить подходящий JDBC-драйвер

б) назначить подходящий JDBC- или ODBC-драйвер

в) назначить подходящий JDBC-драйвер, или JDBC-ODBC-мост

г) вызвать `Class.forName()`

14.10. Компилятор `RMI (rmic)` должен применяться:

а) ко всем классам, которые являются частью RMI-системы

б) только к классам клиента

в) только к классам сервера

г) только к классам удаленного объекта

Задачи

Использование соединений и сокетов

14.1. **Получение списка ссылок посредством HTML.** Создайте апплет, предназначенный для получения списка ссылок из HTML-файла, и поместите его на страницу. Апплет должен брать имена файлов из текстового поля, добавленного в интерфейс апплета. Поместите апплет на HTML-страницу, и пусть HTML предоставляет список файлов, ссылки на которые есть на странице.

14.2. **Чат-клиенты.** Программа `Chat` ("Практикум 10") предоставляет только версию сервера. Используя в качестве основы программу "Порты" из примера 14.3,

создайте Java-программу, которую можно инициировать на стороне клиента и которая обеспечивает средства работы, аналогичные `telnet`.

14.3. **Общение через Web.** Даже после решения задачи 14.2 пользовательский интерфейс программы `Chat` остался довольно примитивным. Постарайтесь сделать программу-сервер апплетом и внедрите его в HTML-страницу, которая предоставляет инструкции, описание процедуры выхода и имеет отдельную строку для ввода.

14.4. **Тук-тук!** Используя в качестве основы ATM-сервер (Пример 14.4), напишите программу-сервер для игры "Тук-тук!". Игра протекает следующим образом.

Клиент	Сервер
Тук-тук!	
	Кто там?
Петр	
	Какой Петр?
Петр Иванов	

Используйте `telnet` в качестве клиента.

14.5. **Текстовые сообщения по сотовому телефону.** Создайте систему обработки текстовых сообщений, передаваемых по сотовому телефону. Имеется центральный сервер, который обслуживает зарегистрированные сотовые телефоны. Каждый сотовый телефон — это клиент, который получает сообщения и отправляет сообщения серверу, ретранслирующему их на нужный номер телефона. Система очень похожа на систему `Chat` ("Практикум 10"), за исключением того, что сообщение идет конкретному получателю. Вы можете имитировать сотовые телефоны, используя командную строку или GUI-интерфейс, подобный разработанному в задаче 10.3 или 12.10.

14.6. **Голосовая почта.** Если телефон отключен, сообщение должно быть сохранено на сервере. Как только телефон вновь начнет работать, система пошлет сообщение, уведомляющее о наличии сохраненных сообщений. После этого пользователь сможет по одному извлечь их. Реализуйте такую систему для сотовых телефонов из задачи 14.5.

Подключения к базам данных

14.7. **Ведение базы данных о домашних животных.** Используя методы, описанные в разделе 14.4, преобразуйте программу "Ветеринарные бирки" ("Практикум 4" в главе 9 и задача 12.9) так, чтобы она работала с базой данных. Апплет клиента должен предусматривать возможность выполнения запросов и подчиняться отдельному подпроцессу-менеджеру, который отвечает за обновление данных. Неплохо было бы также проверить правильность данных перед вводом их в базу.

14.8. **Поиск дома.** Маклер хранит в базе данных записи о домах, выставленных на продажу. Когда приходит клиент, нужно вывести на печать список подходящей недвижимости, отсортированный по цене и/или по площади. Начните с определения объекта "дом" и следуйте этапам, описанным в разделе 14.4.

14.9. **База данных для кофейного магазина.** Нельсон хочет проверить, будет ли база данных лучшей альтернативой связным спискам (см. задачу 9.6). Из этой базы данных клиенты смогут делать только выборку данных, все обновления базы будет делать сам Нельсон. Создайте такую программу и представьте результаты сравнения обоих методов.

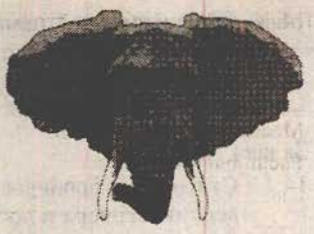
14.10. **"Бронирование" номеров в отелях.** Обратитесь к системе бронирования номеров, разрабатываемой в задачах из глав 4 и 5, и изучите возможность ее работы с базой данных. Все запросы и обновления должны быть доступны для каждого потенциального постояльца.

Задачи по RMI

14.11. **Статистика бронирования номеров.** Рассмотрите программу из задачи 14.10 как версию сервера и добавьте в нее класс `RoomStats`, который имеет доступ к базе данных для получения статистики об использовании номеров. Затем создайте отдельный апплет, взаимодействующий с сервером посредством RMI-вызовов методов класса `RoomStats`. Апплетом будут пользоваться менеджеры, поэтому он должен быть простым в работе.

14.12. **Апплеты для компании аэропорта.** Напишите остальные апплеты, упомянутые в программе "Информационная система аэропорта"

ГЛАВА 15



Структуры данных и алгоритмы

15.1. О структурах данных

В завершение изучения языка Java мы заглянем вперед и создадим программу для курсов по компьютерным наукам, а также рассмотрим структуры данных.

Структура данных — коллекция **узлов** (**nodes**) одного и того же класса или типа, организация которых и доступ к которым осуществляется определенным образом.

Узел обычно является объектом, но может быть и таким простым типом, как целое число (`integer`). Поскольку согласно определению узлы должны принадлежать к одному классу или типу, мы можем обращаться к структуре этого типа, например, связному списку объектов `coffees` (см. главу 8) или хеш-таблице строк.

Структура данных — связный список, который мы строили в разделе 8.2, относится к разряду *ad hoc*, т.е. разработана для конкретной цели. Другая задача может потребовать совершенно другой структуры данных. Компьютерной наукой определено несколько структур данных, которые постоянно используются при решении задач. Они имеют имена и определенные характеристики и могут восприниматься как расширение базисных типов данных языка.

Создателями Java осознана важность структур данных, поэтому в разработанных стандартных пакетах есть немало классов, которые их представляют. В таблице 15.1 приведен канонический список структур данных, указывающий те из них, которые есть в языке Java, и те, которые рассматриваются в этом издании книги.

В этой главе мы вновь вернемся к связным спискам, чтобы добавить некоторые новые функции и привести еще один пример их использования. Кроме того, мы рассмотрим две новые структуры данных, классы `Stack` и `BitSet`, и исследуем, как, базируясь на массивах, можно реализовать очереди.

Характеристики

Каждая структура данных имеет свои характеристики, учитывающие следующее:

- отношения с другими узлами;
- структуру начального узла;
- точку добавления узлов — начало, конец или где угодно;
- точку удаления узлов — начало, конец или где угодно;
- направление сканирования — нет направления, вперед, назад или другое.

По мере рассмотрения структур данных мы более подробно поговорим об этих характеристиках.

Таблица 15.1. Таблица структур данных

Структура данных	Представляющий ее класс Java (если такой есть)	Раздел в этой книге
Массив	Встроенная	6.1, 6.2
Расширяемый массив	`java.util.Vector`	6.5
Связный список	`java.util.LinkedList`	8.1, 8.3, 15.5
Стек	`java.util.Stack`	15.4
Очередь		15.4
Древовидная структура	`java.util.TreeMap`	
Хеш-таблица	`java.util.Hashtable`	6.5
Битовое множество	`java.util.Set,BitSet`	15.6
Словарь	`java.util.Dictionary`	
Последовательный файл	`java.io.File…Stream`	4.5
Файл произвольного доступа	`java.io.RandomAccessFile`	

Представление

Структуры данных могут быть представлены или как **массивы** объектов, или как узлы, **связанные** ссылками, как это показано в разделе 8.2. Выбор метода для дальнейшего использования зависит от двух факторов. Связанное представление имеет то преимущество, что количество узлов в структуре может быть любым, а в представлении с использованием массива должно быть зафиксировано максимальное количество элементов. С другой стороны, структуры, основанные на ссылках, требуют для них места, а это увеличивает общий размер структуры.

Однако в обоих случаях не возникает каких-либо сложностей при программировании, и мы рассмотрим оба варианта представления, пытаясь выявить лежащие в основе алгоритмы и показать их независимость от представления. Выполняя это, мы создаем **абстрактные типы данных**, которые имеют различимые формы и свойства.

Алгоритмы

Чаще всего к структурам данных применяются две операции: сортировка и поиск. Прежде всего мы рассмотрим внутренние детали структур данных, а завершим наше изучение рассмотрением базисных алгоритмов, которые должны быть частью "инструментария" специалиста. Список этих алгоритмов включает в себя:

- линейный поиск и ускоренный;
- прямая сортировка и ускоренная.

Линейный поиск применительно к массивам рассмотрен в разделе 6.4, а в разделе 8.3 показано выполнение линейного поиска при работе со связными списками посредством итераторов. Прямая сортировка также рассматривалась в разделе 6.4. Теперь мы рассмотрим два алгоритма, ускоряющих эти процессы, для чего будем применять базисную технологию вычислений — рекурсию, без описания которой не может считаться полным ни один учебник по программированию.

Алгоритмы, предоставляемые Java

Вместе со структурами данных (табл. 15.1) Java предоставляет фундаментальные алгоритмы, которые мы изучим далее. Есть класс, называемый `Arrays`, его методы

`binarySearch`, `sort`, `equals` и `fill` позволяют работать с массивами, элементы которых могут принадлежать к каждому из восьми примитивных типов, а также к суперклассу `Object` (например, метод `get()` возвращает значение заданного элемента указанного массива в виде экземпляра суперкласса `Object`). Выбрав из перечисленных типов только целые числа, в приводимом ниже описании мы даем методы для работы с ними.

Методы класса `Arrays` для `int` (целых чисел)

```
int binarySearch (int [] a, int key)
boolean equals (int [] a1, int [] a2)
void fill (int [] a, int val)
void sort (int [] a)
```

Некоторые программисты используют эти методы не задумываясь, но мы рекомендуем вам изучить, как они работают. В следующих двух разделах описаны внутренние детали двоичного поиска и быстрой сортировки и сравниваются их эксплуатационные показатели с линейными версиями.

15.2. Линейный и двоичный поиск

Мы можем определить **последовательность** как структуру данных, доступ к каждому элементу которой предоставляется после предыдущего элемента. Другими словами, доступ к структуре данных выполняется с соблюдением последовательности. Примерами таких структур могут быть массивы, связные списки или потоки файлов. Хотя в Java доступ к хеш-таблице может выполняться последовательно посредством элементов перечисления (*enumerators*), она не является последовательностью.

Если у нас есть последовательность значений, то можно найти конкретное значение, начав с начала последовательности и сравнив по очереди каждый ее элемент с заданным. Это называется **линейным (linear)** поиском, поскольку мы линейно (в одном направлении) сканируем последовательность.

Алгоритм линейного поиска является примером цикла с двумя выходами (*double-exit loop*), т.е. цикл нормально завершится в обоих случаях: или значение будет найдено в последовательности, или конец последовательности будет достигнут раньше, чем будет найдено значение. Эти виды циклов подробно рассматривались в разделе 5.2, где последовательностью был поток вводимых данных, и повторно в разделе 8.3 при рассмотрении связных списков. Теперь воспользуемся этой логикой для выполнения поиска в массиве.

Преимущество поиска в массиве заключается в том, что мы знаем его длину. Следовательно, можно использовать цикл `for` наряду с возможностью контроля, которую этот цикл предоставляет (возможность завершения цикла, если значение не найдено). Вот первая попытка линейного поиска:

```
int i;
for ( i = 0; i< a.length; i++)
  if (a[i] == x) break;
```

Хотя этот цикл прост и аккуратен, он, к сожалению, имеет недостаток. При его завершении мы не знаем, почему он закончился: потому что мы нашли искомое значение или потому что была просмотрена вся последовательность значений. Также заметьте, что мы преднамеренно объявили `i` вне цикла, чтобы впоследствии можно было получить `a[i]`.

Это означает, что алгоритм возвращает два значения: индикатор да/нет (или булеву переменную) и индекс, что делает проблематичной возможность оформления поиска в метод, поскольку метод возвращает только одно значение (если не будет указан для возвращения объект, имеющий много значений).

Можно применить следующую технологию (хотя мы не рекомендуем этого делать) — в случае, если поиск не увенчается успехом, назначить возвращаемому индексу значение, выходящее за диапазон массива. Такими значениями могут быть или −1, или длина массива. Лучший способ решения этой проблемы — использование определенных пользователем исключений. Рассмотрим следующий вариант:

```
class ItemNotFoundException extends Exception { }

item a [] = new a[n];

static int LinearSearch (item a [], item x)
       throws ItemNotFoundException {
for (int i = 0; i< a.length; i++)
  if (x.equals(a[i])) return i;
  //  Искомое значение не найдено: нет нормального возврата.
  throw ItemNotFoundException();
}

// Проверка метода.
try {
  int found = LinearSearch (a, x);
  // Все необходимые операторы для работы с объектом a[found].
  ...
}
catch (ItemNotFoundException e) {
  // реагирование на отсутствие "x" в массиве
  ...
}
```

Методу передается массив и элемент (объект), который нужно найти. Метод просматривает массив и сравнивает каждый элемент с `x`, используя длину массива как условие для остановки цикла (что делает метод применимым для массивов любой длины). Сравнение производится посредством вызова метода `equals`, поскольку здесь мы принимаем, что элементы являются объектами. Если элемент найден, то цикл возвращает объект немедленно, если же цикл безрезультатно проходит весь массив, то мы "выбрасываем" объект-исключение.

Вызывающий метод имеет пару `try-catch`. Если поиск увенчался успехом, выполняется следующий оператор, каким бы он ни был. Если нет, управление передается обработчику, определенному в блоке `catch`, и выполняются альтернативные действия. На Web-узле есть тестовая программа, которая демонстрирует этот метод.

Двоичный поиск

Если данные, просматриваемые в процессе поиска, упорядочены (отсортированы), то можно применить более эффективные методы, чем линейный поиск. Одним из таких методов является **двоичный поиск (binary search)**. Он предусматривает разделение последовательности пополам и выполнение поиска только в той части, в которой должно находиться искомое значение. Поскольку принимается, что последовательность отсортирована, то можно уверенно определить, в какой части находится искомое значение.

Например, у нас есть отсортированная последовательность

```
23 45 61 65 67 70 82 89 90 99
```

и мы ищем значение 90. Можно разделить последовательность пополам (между 67 и 70) и увидеть, что 90 должно находиться в правой части:

```
70 82 89 90 99
```

Эту подпоследовательность делим еще на две части (между 89 и 90) и вновь переходим в правую часть:

```
90 99
```

В этой последовательности находится только два элемента, но мы вновь ее делим и перемещаемся влево, где находим 90.

Этот процесс включает в себя три разделения и три сравнения, что гораздо удобнее, чем девять сравнений при линейном поиске.

Очевидно, что двоичный поиск работает быстрее, чем средний алгоритм поиска, но так бывает не всегда. Если бы числом, которое нужно найти, было 23 (в нашем примере оно стоит в последовательности первым), двоичный поиск все равно стартовал бы с середины последовательности и постепенно продвигался бы влево к этому числу. Лучший способ сформулировать такой двоичный поиск — использовать технологию, которая называется **рекурсией**.

Рекурсия позволяет нам описать операцию, исходя из ее же принципов работы, базируясь на модифицированных данных и включая условия для завершения.

Используя такой подход, алгоритм двоичного поиска можно описать следующим образом.

Двоичный поиск элемента x в **последовательности**
Если последовательность имеет один элемент, то сравнивает с ним x и возвращает булеву переменную или объект-исключение, указывающие, найден или не найден элемент.
В противном случае рассматривается элемент, находящийся в середине последовательности:
если x меньше среднего элемента
(в дальнейшем "средняя точка"), то двоичный поиск
продолжает искать x в левой подпоследовательности,
если больше, то двоичный поиск продолжает работу с
правой подпоследовательностью.

Во многих случаях последовательность или подпоследовательность, которую нужно разделить пополам, не будет иметь четного числа элементов, поэтому мы применяем соглашение, согласно которому дополнительный элемент переходит в левую подпоследовательность. Рассматривая алгоритм, заметьте, что его можно усовершенствовать, добавив дополнительное условие в его вторую часть и проверив, не является ли искомое значение средним элементом.

Условия для двоичного поиска

Двоичный поиск полагается на значения, которые уже были упорядочены, а также на возможность проиндексировать последовательность. Поэтому невозможно применить двоичный поиск к файлам или связным спискам. Он может работать только с массивами.

Пример 15.1. Интерактивный двоичный поиск

Задача. Для демонстрации работы механизма двоичного поиска разработаем Java-программу, в которой реализуем описанный алгоритм, и покажем все этапы поиска.

Решение. Начнем работать с последовательностью чисел и каждый раз после разделения последовательности пополам будем выводить на экран только числа, оставшиеся в подпоследовательности. Возникает вопрос, как мы разделим последовательность? Ответ: мы не будем ее делить! Последовательность останется как базис, а мы будем работать с подпоследовательностями, каждый раз указывая для них левые и правые границы.

Алгоритм. Процесс деления последовательности пополам будет несколько иным, чем в предыдущем примере. Если последовательность начинается с 0 и имеет `n` элементов (т.е. `n` — последний индекс), то средней точкой будет элемент `n/2` (целочисленное деление). Например, для массива с индексами от 0 до 9 средней точкой будет 9/2 = 4. Однако если большая последовательность уже была разделена и для подпоследовательности назначены границы `j` и `k`, то средней точкой будет элемент с индексом `(j+k)/2`. Например, между 5-м и 9-м элементами средним будет (5+9)/2 = 7.

Программа. Далее следует программа для интерактивного поиска. Используем символьные данные, сохраняемые как объекты, но предварительно отсортированные. Вспомните, что данные *должны быть отсортированы*, иначе программа не будет правильно работать. (Вставьте в программу процедуру сортировки и для уверенности всегда выполняйте ее перед поиском.)

Важной частью метода двоичного поиска является использование рекурсии, поэтому вначале рассмотрим ее отдельно и пронумеруем все строки кода.

```
1.  static int binarySearch (item a [], item x, int left, int right)
2.          throws ItemNotFoundException {
3.       display(a,left,right);
4.     if (left==right)
5.       if (x.equals(a[left]))
6.         return left;
7.       else
8.         throw new ItemNotFoundException ();
9.     else {
10.       int mid = (int) ((left+right) / 2);
11.       if (x.equals(a[mid]))
12.         return mid;
13.       else
14.         if (x.less(a[mid]))
15.           return binarySearch (a, x, left, mid);
16.         else
17.           return binarySearch (a, x, mid+1, right);
18.     }
19.  }
```

Метод `binarySearch` будет самовызывать для работы с левой или правой подпоследовательностями (строки 15 и 17) до тех пор, пока длина подпоследовательности не станет равна одному элементу (строка 4). Затем он или завершит свою работу, выдав объект-исключение (строка 8), или возвратит индекс найденного элемента (строка 6). Точка, в которой метод будет вызывать себя, находится или в 15-й, или в 17-й строке. В дальнейшем, из какой бы строки он ни стартовал, метод, возможно, еще несколько раз рекурсивно вызовет себя и в конце концов достигнет

естественного завершения (строка 19). После этого он вернется в предыдущую точку вызова (15-ю или 17-ю строку). Так очередь дойдет до самого первого вызова метода `binarySearch`, который был сделан для полной последовательности. Он также достигнет своего естественного завершения, и управление (программой) перейдет обратно к вызывавшему методу. В течение всего этого процесса значение индекса, обнаруженное как содержащее искомый элемент `x`, будет передаваться назад в качестве возвращаемого значения метода.

Метод `binarySearch` объявлен как статический, поскольку он аналогичен методу `mean` или `stdDev`: получает параметры, а возвращает значение. Однако он использует вспомогательный метод `display` и переменную `counter`. Следовательно, они также должны быть статическими. Переменная `counter` не объявляется внутри метода `binarySearch`, потому что переменная должна сохранять значение, которое будет независимым от рекурсии. Если бы она была внутри рекурсивного метода, то при каждом повторении цикла получала бы значение 0 и в ней невозможно было бы сохранить учет количества выполненных поисков.

Далее следует код программы с ассоциированными классами.

```
import javagently.*;
import java.io.*;

class Animated {

  /*       Программа                  Автор Дж.М. Бишоп, январь 1997 г.
   * "Интерактивный двоичный поиск"                          Java 1.1
   *                                          улучшена, август 2000 г.
   * Ищет заданное значение в отсортированной последовательности
   * и показывает ход работы двоичного поиска.
   *
   * Демонстрирует использование рекурсии и определенных
   * пользователем объектов-исключений.
   */

  static item a [] = new item [10];
  static int counter;
  static int countRuns;

  static int binarySearch (item a [], item x, int left, int right)
  //----------------------
                              throws ItemNotFoundException {
    display(a,left,right);
    if (left==right)
      if (x.equals(a[left]))
        return left;
      else throw new ItemNotFoundException ();
    else {
      int mid = (int) ((left+right) / 2);
      if (x.equals(a[mid]))
        return mid;
      else
        if (x.less(a[mid]))
          return binarySearch (a, x, left, mid);
        else
          return binarySearch (a, x, mid+1, right);
    }
  }
```

```
public static void main (String args []) throws IOException {

  Stream in = new Stream(System.in);
  System.out.println("**** Тестирование двоичного поиска ****");
  System.out.println("Введите 10 отсортированных символов, "+
                     "разделенных пробелами");
  countRuns = 0;
  for (int i=0; i<a.length; i++)
    a[i] = new item (in.readChar());

  // Определяем цикл для выполнения нескольких поисков.
  while (countRuns < 4)
    try {
      counter = 0;
      System.out.print("Какой символ найти?");
      item x = new item(in.readChar());
      System.out.println("Индексы массива символов:");
      System.out.println("0  1  2  3  4  5  6  7  8  9");

      try {
        countRuns++;
        int found = binarySearch (a, x, 0, a.length-1);
        // Все необходимые операторы для работы с объектом a[found]
        System.out.println("Символ "+(char) x.data+
              " найден на позиции "+found
              +" за "+counter+" прохода.");
      }

      catch (ItemNotFoundException e ) {
        // Реагирование на отсутствие объекта "x" в массиве.
        System.out.println("Символ "+(char) x.data+" отсутствует в
                           массиве. Потребовалось "
          +counter+" прохода.");
      }
    }
  catch (EOFException e) {break;}
}

static void display (item [] a, int left, int right) {
  for (int j = 0; j < left; j++)
    System.out.print("   ");
  for (int j = left; j <= right; j++)
    System.out.print((char) a[j].data+"  ");
  System.out.println();
  counter++;
}

}

class item {
// ---------
// Отсортированные объекты.

  item (char i) {
    data = i;
  }
```

```
    boolean equals (item x) {
      return data==x.data;
    }

    boolean less (item x) {
      return data < x.data;
    }

    char data;
  }
class ItemNotFoundException extends Exception { }
```

Тестирование. Показано несколько результатов тестирования*, включая варианты, где числа были найдены, и вариант, когда искомое число отсутствовало в массиве. Заметьте, что `t` и `k` были найдены с меньшим количеством проходов, чем при линейном поиске, но для нахождения `e` сделано больше попыток. Реальное преимущество двоичного метода состоит в том, что он позволяет установить отсутствие искомого элемента за меньшее количество проходов, чем линейный метод. Точное количество попыток вычисляется по формуле, которую мы рассмотрим в следующем разделе.

```
****Тестирование двоичного поиска****
Какой символ найти? e
0  1  2  3  4  5  6  7  8  9
e  g  h  j  k  m  r  t  w  y
e  g  h  j  k
e  g
Символ e найден в позиции 0 за 4 прохода
Какой символ найти? f
0  1  2  3  4  5  6  7  8  9
e  g  h  j  k  m  r  t  w  y
e  g  h  j  k
e  g
   g
f отсутствует в массиве. Потребовалось 5 прохода
Какой символ найти? k
0  1  2  3  4  5  6  7  8  9
e  g  h  j  k  m  r  t  w  y
Символ k найден в позиции 4 за 1 прохода
Какой символ найти? t
0  1  2  3  4  5  6  7  8  9
e  g  h  j  k  m  r  t  w  y
                m  r  t  w  y
Символ t найден в позиции 7 за 2 прохода
```

15.3. Ускоренная сортировка Quicksort и эффективность работы

Существует много алгоритмов, предназначенных для сортировки, но один признан классическим благодаря высокой эффективности; во многих языках программирования он называется одинаково — **Quicksort (быстрая сортировка)**. Сортировка

* Чтобы программа выполнялась определенное количество раз, была введена переменная `countRuns`, которой позволено иметь максимальное значение 4, то есть можно найти только 4 буквы.

базируется на двух операциях: сравнения и обмена. Алгоритм `Quicksort` основывается на том принципе, что любой обмен должен происходить на наибольшем возможном расстоянии. Поэтому вместо замены прилегающих элементов мы выбираем элементы, которые фактически находятся на противоположных концах последовательности. Для этого применяем технологию разделения последовательности пополам, использующуюся в двоичном поиске. Мы делим последовательность в определенной точке, далее назовем ее осью, и перемещаем все большие элементы вправо, а меньшие влево, выполняя "длинные обмены". После чего поочередно работаем с каждой подпоследовательностью и повторяем описанный процесс до тех пор, пока в каждой из них не останется по одному элементу. Сортировка `Quicksort` не делит каждую подпоследовательность точно пополам, как при двоичном поиске, а делает это, базируясь на позиции последнего обмена.

Алгоритм `Quicksort` можно описать так:

Quicksort
Если ему дана последовательность, имеющая более чем один элемент:
выбирает элемент в качестве оси (например, среднюю точку);
перемещает все элементы, меньшие элемента оси, влево;
перемещает все элементы, большие элемента оси, вправо;
вызывает `Quicksort` для работы с левой подпоследовательностью;
вызывает `Quicksort` для работы с правой подпоследовательностью.

Алгоритм для перемещения показан на рис. 15.1.

Алгоритм `Quicksort` довольно сложный, и даже детальное рассмотрение в примере не делает его проще. Это один из тех алгоритмов, которые нужно понять теоретически, а затем принять его правильность как постулат, хотя в Java с ним можно вполне успешно работать и создавать программы.

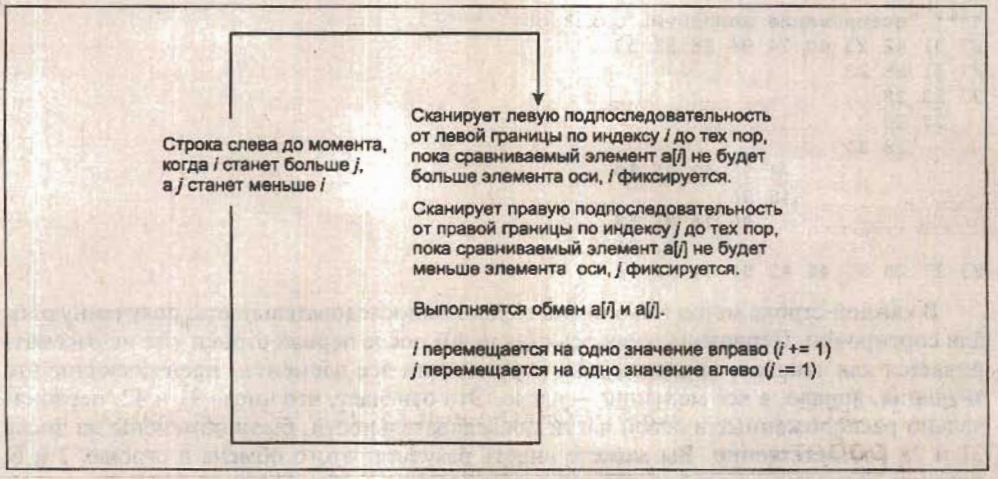


Рис. 15.1. Алгоритм перемещения в сортировке `Quicksort`

В Java рекурсивная версия алгоритма `Quicksort` выглядит так:

```
1.  static Item [] quicksort (Item [] a, int l, int r) {
2.    if (l < r) {
3.      display(a,l,r);
```

```
4.      int i = l;
5.      int j = r;
6.      int k = (int) ((l+r) / 2);
7.      Item pivot = a[k];
8.      do {
9.        while (a[i].less(pivot)) i++;
10.       while (pivot.less(a[j])) j--;
11.         if (i<=j) {
12.           t = (Item) a[i].clone();
13.           a[i] = a[j];
14.           a[j] = t;
15.           i++; j--;
16.       }
17.     } while (i<j);
18.     a = quicksort (a, l, j);
19.     a = quicksort (a, i, r);
20.   }
21.   return a;
22. }
```

Отметим несколько моментов, касающихся программирования. В строке 7, используя присвоение, мы назначили ось как ссылку на средний элемент, а в строке 12 применили метод `clone`, чтобы сделать копию `a[i]`. Разница в том, что в строках 12–14 выполняется обмен двумя элементами, `a[i]` и `a[j]`. Прежде чем в строке 13 будет удален (стерт) элемент `a[i]`, мы должны скопировать его содержимое.

Как и `binarySearch`, `quicksort` является типизированным методом, но на этот раз мы возвращаем не один индекс, а массив. Следовательно, при вызове метод принимает массив в качестве параметра и возвращает его новую версию. На нашем Web-узле вы можете найти весь текст программы `Quicksort`. Ниже приводим один из возможных результатов ее работы:

```
**** Тестирование алгоритма Quicksort ****
27 91 42 23 40 74 96 28 31 59
27 31 28 23
27 23 28
   27 28
      28 31
                74 96 42 91 59
                42 96
                   96 74 91 59
                         91 96
23 27 28 31 40 42 59 74 91 96
```

В каждой строке метод выводит на экран подпоследовательность, полученную им для сортировки. Первоначальная ось (число 40) после первой строки уже не рассматривается как таковая, поскольку мы переместили все элементы, превышающие это значение, вправо, а все меньшие — влево. Это означает, что числа 91 и 42, первоначально расположенные в левой части последовательности, были обменены на числа 31 и 28 соответственно. Вы можете видеть результат этого обмена в строках 2 и 6, которые представляют собой две первые подпоследовательности.

Оценка эффективности

Использование различных алгоритмов для решения одной и той же задачи объясняется стремлением повысить скорость: новые алгоритмы могут быть быстрее, а следовательно, почти всегда, — лучше. Рассматривая алгоритмы поиска и сортировки, мы

видим, что различия могут быть значительными. Кроме того, скорость алгоритма проявляется при работе с большими количествами элементов, и разница становится весьма ощутимой, когда приходится иметь дело с тысячами и миллионами элементов.

Есть два способа сравнения эффективности работы: теоретический и экспериментальный. Вначале рассмотрим теоретическую оценку эффективности. Мы вычисляем скорость алгоритма, основываясь на количестве базисных операций, которые ему необходимо выполнить. Они рационализированы до уровня, когда учитываются только сравнения и обмены, и оценка эффективности сводится к подсчету количества этих внутренних циклов.

Сравнение алгоритмов сортировки

Рассмотрим сортировку методом отбора (она изучалась в разделе 6.4). Внешний цикл доходит до индекса $n-1$ и предусматривает $n-1$ обменов. Внутренний цикл проходит от крайнего левого значения до $n-1$ и после каждого повторения цикла уменьшается на один пункт. Следовательно, количество сравнений равно:

$$(n-1) + (n-2) + (n-3)+...+ 1 = n(n-1)/2$$
$$= (n2 - n)/2$$

Для простоты мы игнорируем коэффициенты в формуле, вычисляющей количество обменов, и принимаем, что оно приблизительно равно n^2. Если мы добавим количество обменов к количеству сравнений, то весь процесс также можно приблизительно принять, как насчитывающий n^2 выполненных операций.

С другой стороны, сортировка `Quicksort` использует алгоритм разделения, который предусматривает приблизительно $\log_2$ повторений (проходов цикла). При каждом повторении выполняется `n` сравнений и приблизительно $n/6$ обменов. Поэтому считается, что алгоритм `Quicksort` работает со скоростью, пропорциональной $n\log_2 n$. Но как реально сравнить эффективность его работы с эффективностью сортировки, предусматривающей отбор? В табл. 15.2 приведены значения, вычисленные по обеим формулам для различных значений `n`. Различие феноменальное! Предположим, что одно повторение происходит за единицу времени, равную 1 микросекунде (10^{-6}с). Для сортировки миллиона элементов алгоритму `Quicksort` потребуется 20 секунд, тогда как алгоритму, предусматривающему отбор, — 11,5 дней! Интересно, что при такой длительности цикла различие между двумя методами сортировки не будет реально ощутимо до тех пор, пока количество элементов не превысит 1000. К этому моменту алгоритму, предусматривающему отбор, потребуется целая секунда, тогда как алгоритм `Quicksort` выполнит работу за 0,01 секунды. В интерактивной среде различие может даже остаться незамеченным.

Таблица 15.2. Сравнение скорости работы алгоритма `Quicksort` и алгоритма, предусматривающего отбор

n	Сортировка с отбором, время пропорционально n^2	Quicksort, время пропорционально $n \log_2 n$
10	100	30
50	2500	300
100	10 000	700
1000	1 000 000	10 000
10 000	100 000 000	130 000
100 000	10 000 000 000	1 600 000
1 000 000	1 000 000 000 000	20 000 000

Так почему же мы постоянно не используем алгоритм `Quicksort`? Дело в том, что эффективность зависит еще и от объема памяти, ведь Java-методам требуется место в памяти и для инструкций, и для данных. Что касается места в памяти для инструкций, здесь особого выбора при сравнении алгоритмов нет, поскольку оба они имеют примерно одинаковое количество операторов, а вот требуемый для данных объем памяти отличается значительно. Сортировка с отбором объявляет четыре локальные переменные, которые затем использует в своей работе. Сортировка `Quicksort` объявляет восемь локальных переменных и параметров для каждого рекурсивного вызова. Поскольку рекурсивные вызовы помещаются в стек, то в случае сортировки миллиона элементов, `Quicksort` поместит в стек 160 переменных и параметров.

Увы, но даже не это является причиной относительно редкого использования `Quicksort`. Существует некоторая настороженность, с которой средние программисты воспринимают `Quicksort`, очевидно, вследствие недостаточного знания механизма рекурсии. Конечно, `Quicksort` не обязательно должен программироваться рекурсивно, но его нерекурсивная версия выглядит еще менее элегантной.

Сравнение алгоритмов поиска

Можно легко обнаружить, что два рассмотренных нами алгоритма поиска имеют эффективность, пропорциональную n (для линейного) и $\log_2 n$ (для двоичного). Здесь различие в скорости даже более радикальное, чем при сравнении алгоритмов сортировки (см. таблицу 15.3). Если применить эти цифры к реальной задаче, то поиск нужного вам номера в телефонном справочнике, насчитывающем миллион записей, при применении двоичного поиска займет только 20 повторений цикла. Понятно, что это несравнимо с кропотливой работой линейного поиска, который запись за записью будет проверять весь справочник.

Таблица 15.3. Сравнение скорости работы линейного и двоичного алгоритмов поиска

n	Линейный поиск, время пропорционально n	Двоичный поиск, время пропорционально $\log_2 n$
10	10	3
50	50	6
100	100	7
1 000	100	10
10 000	10 000	13
100 000	100 000	16
1 000 000	1 000 000	20

Другие алгоритмы сортировки

Алгоритмы и анализ их эффективности — большой раздел компьютерных наук, изучение которого занимает большую часть учебного процесса второго курса соответствующих колледжей. В дополнение к двум алгоритмам сортировки, упомянутым здесь, вы будете изучать и другие, такие как пузырьковый метод (очень медленный), метод сортировки слиянием и вариант древовидной сортировки (очень быстрый), а также метод сортировки "помещением в определенную нишу", который чрезвычайно быстр, но содержит много операторов и переменных. Вы научитесь выбирать алгоритмы для решения конкретной задачи и учитывать то, насколько быстро и эффективно они могут работать. Это увлекательная тема, к которой мы, объективно говоря, только прикоснулись.

15.4. Стеки и очереди

Теперь приступим к изучению **стека** — нашей первой структуры данных. Неформально стек можно определить как кипу элементов, которые добавляются и удаляются (извлекаются) с его верха. Схему стека часто рисуют вертикально (рис. 15.2 а), хотя можно нарисовать и горизонтально (рис. 15.2 б).

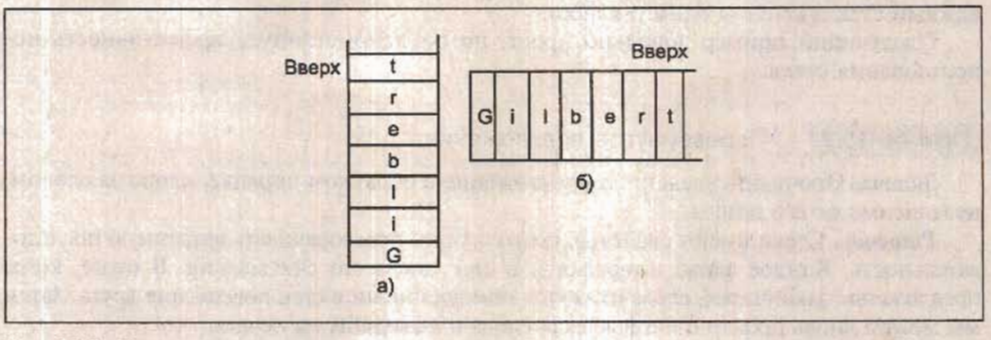


Рис. 15.2. Схема строения стека

Характеристики стека

Формально мы можем детализировать характеристики стека в соответствии с критериями, указанными в разделе 15.1:

- узлы организованы в последовательность;
- есть один конец — **верх**;
- узлы добавляются **сверху**;
- узлы извлекаются **сверху**;
- стек невозможно сканировать: виден только **верхний** узел*.

Стек также может быть определен как структура данных, соответствующая принципу `LIFO` (`last in first out`), когда последний помещенный элемент извлекается первым.

Класс Stack

Мы должны определить стек как класс. Java предоставляет такой класс в пакете `util`, и он имеет следующие методы:

Класс `Stack`

```
public class Stack extends Vector {
  public Object push (Object item);
  public Object pop () throws EmptyStackException;
  public Object peek () throws EmptyStackException;
  public boolean empty ();
```

Метод `push` получает элемент и помещает его на вершину стека. Метод `pop` *возвращает верхний элемент, одновременно "приподнимая" стек, в то время как*

* Здесь есть некоторая неточность: фактически Java предоставляет метод для выполнения поиска в стеке.

метод peek просто возвращает верхний элемент. Метод empty определяет, имеются ли в стеке элементы. Класс Stack базируется на классе Vector, это означает, что стек будет эффективно реализован как массив и не будет ограничен указанным размером.

Методы стека определены для суперкласса Object, поэтому мы можем поместить в стек и извлечь из него объекты любого класса, обеспечивая правильное приведение типа. Хотя в Java это не является обязательным условием, мы должны помещать в каждый стек элементы одного класса.

Следующий пример довольно прост, но он демонстрирует эффективность использования стека.

Пример 15.2. Перевернутое предложение

Задача. Прочитать введенное предложение в обратном порядке, слово за словом, независимо от его длины.

Решение. Стеки имеют свойство, позволяющее переворачивать введенную последовательность. Каждое слово помещается в стек после его считывания. В итоге, когда предложение закончится, слова окажутся помещенными в стек поверх друг друга. Затем мы можем вновь пройти стек, извлекая слова и выводя их на печать.

Программа

```
import java.io.*;
import javagently.*;
import java.util.Stack;

public class Reverser {

  /* Тестирование класса Stack      Автор Дж.М. Бишоп, январь 1997 г.
   * -------------------------      Java 1.1
   *                                обновлена, август 2000 г.
   * Считывает предложение, а затем выводит его на печать
   * в обратном порядке, т.е. "переворачивает".
   * Демонстрирует использование методов pop и empty класса Stack.
   */

  public static void main (String args [])
    throws IOException {
    new Reverser();
  }

  Reverser () throws IOException {

    Stream in = new Stream (System.in);

    Stack S = new Stack();
    System.out.println("**** Тестирование класса Stack ****");
    System.out.println("Введите предложение, а в конце ввода" +
        " нажмите (Ctrl-D или Ctrl-Z)");
    System.out.println("Введенное предложение: ");

    while (true) {
      try {
        String word = in.readString();
        S.push(word);
      }
```

```
      catch (EOFException e) {break;}
    }
    System.out.println(" ");
    System.out.println("Перевернутое предложение: ");
    while (!S.empty())
      System.out.print (S.pop()+" ");
    System.out.println();
  }
}
```

Тестирование. Ввод и вывод будут выглядеть так:

```
**** Тестирование класса Stack ****
Введите предложение, а в конце ввода нажмите (Ctrl - D или Ctrl - Z)
Введенное предложение:
Карл у Клары украл кораллы
Перевернутое предложение:
кораллы украл Клары у Карл
```

Использование стеков

Использование стеков оправданно при записи (фиксировании) состояния вычислений по мере их выполнения. Типичным примером является оценка выражений, которые включают в себя старшинство и вложение. Стек относится к структурам данных, которые реализуют рекурсию. В Java обычно есть возможность применять рекурсию с подразумеваемым стеком, что предпочтительнее, чем с явно заданным. Такая структура данных, возможно, практически более удобна.

Очереди

Хотя стеки являются наиболее простой имеющейся структурой, их свойство реверсного извлечения введенной последовательности полезно не всегда. Более общей структурой данных является такая, в которой элементы могут извлекаться в том же самом порядке, в каком они вводились. Эта структура называется **очередь** (**queue**) и работает по принципу FIFO (`first-in, first-out`), когда элемент, введенный первым, первым же и извлекается.

Для того чтобы можно было реализовать этот принцип, очередь должна иметь начало и конец. Новые элементы добавляются в конец очереди, а удаляются из ее начала. Графически очереди обычно изображаются горизонтально (рис. 15.3 а). Однако они могут "двигаться" (как только спереди удаляется один элемент, все остальные элементы, которые находились за ним, перемещаются вперед на одну позицию). Сразу же заметим, что такое перемещение при реализации очереди для элементов массива будет неэффективным, поэтому рассмотрим альтернативную организацию очереди как круга. Здесь имеется фиксированное количество позиций, и очередь продвигается по кругу, как это показано на рис. 15.3 б.

Характеристики очереди

Формально мы можем детализировать характеристики очереди следующим образом:

- узлы организованы в последовательность;
- имеется два конца: **начало** очереди и ее **конец**;
- узлы добавляются **сзади**;
- узлы удаляются **спереди**;
- очередь можно сканировать по направлению от начала к концу.

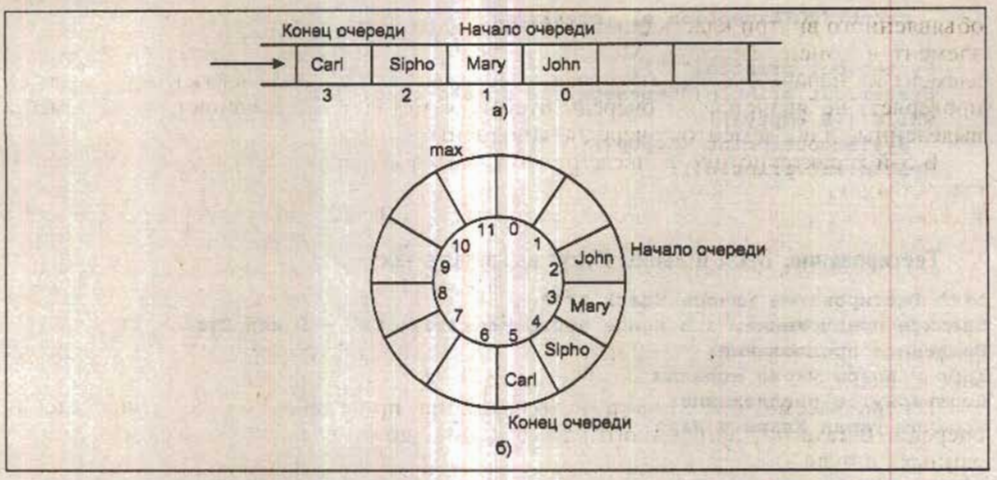


Рис. 15.3. Линейная и круговая организации очереди

Очередь как абстрактный тип данных

Теперь нам нужно определить очередь как тип данных. Так же, как и для стека, мы определим ее на основе суперкласса `Object`. Однако в Java нет класса, определяющего очередь, поэтому мы должны его разработать. Разрабатываемый класс может базироваться на массиве, классе `Vector` или классе `LinkedList`. Последние два позволили бы очереди иметь переменную длину. Класс `LinkedList` является, возможно, более мощным, чем это необходимо, поскольку нам нужно удалять элементы только спереди, а не из любой позиции. Класс `Vector` предоставляет возможность расширения, но ведь мы фактически хотим контролировать очередь с помощью повторного использования "мест", освобожденных после удаления элемента, что очень удобно вписывается в движение по кругу (рис. 15.3 б). Поэтому лучшим выбором будет обыкновенный массив.

Если количество элементов очереди сравняется с выделенным размером массива (очередь станет *полной — full*) или будет предпринята попытка удалить элемент из *пустой* очереди (*empty*), мы сгенерируем объект-исключение. Сделаем это, используя один объект-исключение с параметром для полной и пустой очереди. И так же, как в случае с классом `Stack`, определим очередь на основе суперкласса `Object`.

Ограниченная очередь

```
public class Queue {
  Queue (int n);
  void add (Object x);
  Object remove ();
  boolean empty ();
  boolean full ();
void reset ();
  void succ ();
  boolean eol ();
  Object current ();
 }
```

Одним из свойств круговой очереди является то, что она ограниченна, и в конструкторе мы можем указать границы. В реализации очереди на основе массива любая указываемая граница должна быть меньше, чем максимальный размер массива,

объявленного внутри класса `Queue`. Метод `add` (известный и как `enqueue`) добавляет элемент в конец очереди. Метод `remove` (известный и как `dequeue`) извлекает элемент из начала очереди (в случае, если в очереди есть элементы). Метод `empty` проверяет, не является ли очередь пустой, а метод `full` выясняет, все ли места выделенные для элементов очереди, заполнены.

В 5-й характеристике приведенного выше пронумерованного списка характеристик говорится, что мы можем сканировать очередь, но не указывается, каким образом это сделать. Например, нам может понадобиться вывести элементы очереди на печать или найти конкретный элемент. Без более сложных функций языка будет трудно создать обобщенную операцию сканирования очереди. Поэтому мы оставляем пользователю возможность самостоятельно разработать метод для доступа к элементам очереди. Заметим только, что доступ должен контролироваться, причем все операции сканирования будут использовать четыре метода итерации, предоставляемые классом.

Теперь рассмотрим пример использования приведенного выше определения очереди. В Java нет встроенного класса `Queue`, поэтому мы разработаем структуру данных "с нуля".

Пример 15.3. Приемная доктора

Задача. У доктора есть небольшая приемная, в которой расположено ограниченное количество кресел, скажем, семь. Пациенты могут заходить и ждать, но после того, как все кресла будут заняты, новым больным придется уйти и вернуться позже. Создайте программу, которая позволит доктору решить, строить ли ему новую приемную большего размера.

Решение. Мы можем определить очередь с необходимым максимальным размером и, используя механизм генерирования случайных чисел, организовать "посещения" доктора пациентами. Состояние очереди может постоянно отображаться на экране, и когда приемная заполнится, будет подан сигнал, после которого доступ новых пациентов в приемную станет невозможным.

Алгоритм. Мы можем имитировать происходящее при помощи алгоритма, показанного на рис. 15.4. При каждом повторении существует 50-процентная вероятность, что прибудет пациент (выбор варианта хода событий в программе аналогичен бросанию кубика, у которого есть шесть граней (значений). Принимается, что пациент "приходит", когда выпадает четное число). Вероятность того, что доктор будет готов принять следующего пациента (грани 1 или 2) составляет 30 %. Вероятность того, что ничего не произойдет (грани 3 или 5) также составляет 30 %.



Рис. 15.4. Имитация ситуаций, происходящих в приемной доктора

Программа. Элементами очереди будут простые целые числа. Мы знаем, что очередь работает с экземплярами суперкласса `Object`, поэтому выполняем необходимые приведения типа.

```
import myutilities.*;

class WaitingRoom {
  /*  Имитация приемной     Автор Дж.М. Бишоп, январь 1997 г.
   *  -----------------     Java 1.1
   *  Показывает очередь, увеличивающуюся или уменьшающуюся
   *  в зависимости от случайно генерированных событий.
   *
   * Демонстрирует применение очередей
   * и определенных пользователем итераторов очереди.
   */

  public static void main (String args []) throws QueueException {
      new WaitingRoom ();
  }

  WaitingRoom () throws QueueException {
    Queue chairs = new Queue (7);
    int patient;
    int choice;

    System.out.println("*** Имитация приемной доктора ***");
    System.out.println("Имеется 7 кресел");
    System.out.println("Если выпало 2,4,6, приходит пациент");
    System.out.println("Если выпало 1,2, доктор принимает пациента");
    System.out.println();
    System.out.println("Время\tВариант\tНомера пациентов");
    for (int i = 2; i<30; i++) {
      choice = (int) (Math.random()*6+1);
      System.out.print(i+"\t"+choice+"\t");
      display(chairs);
      try {
        if (choice == 1 || choice == 2)
          if (!chairs.empty())
            patient = ((Integer) chairs.remove()).intValue();
        if (choice == 2 || choice == 4 || choice == 6)
          chairs.add(new Integer(i));
      }
      catch (QueueException e) {
        System.out.println("\t\t\tПриемная "+e.getMessage());
      }
    }
  }

  void display (Queue q) {
    if (q.empty()) System.out.print("Входите"); else
    for (q.reset(); !q.eol(); q.succ())
      System.out.print(((Integer) q.current()).intValue() +"  ");
    System.out.println();
  }
}
```

Тестирование. Результат выполнения программы* выглядит так:

```
*** Имитация приемной доктора ***
Имеется 7 кресел
Если выпало 2,4,6, приходит пациент; patients seen on 1,2
Если выпало 1,2, доктор принимает пациента
Время Вариант Номера пациентов
2     1       Входите
3     1       Входите
4     6       Входите
5     5       4
6     4       4
7     4       4 6
8     5       4 6 7
9     5       4 6 7
10    1       4 6 7
11    4       6 7
12    4       6 7 11
13    6       6 7 11 12
14    4       6 7 11 12 13
15    1       6 7 11 12 13 14
16    4       7 11 12 13 14
17    6       7 11 12 13 14 16
18    6       7 11 12 13 14 16 17
              Приемная заполнена
19    1       7 11 12 13 14 16 17
20    1       11 12 13 14 16 17
21    5       12 13 14 16 17
22    2       12 13 14 16 17
23    1       13 14 16 17 22
24    6       14 16 17 22
25    3       14 16 17 22 24
27    1       16 17 22 24
28    1       17 22 24
29    4       22 24
```

В этом коротком варианте выполнения программы приемная оказалась заполненной только один раз, но поскольку все зависит от случайного генерирования чисел, то при следующем выполнении будут получены совершенно другие результаты. Заметьте, что когда "выпадает" 2, то за один проход цикла одновременно один элемент добавляется в очередь и один удаляется. Это происходит в строках 22–23.

Очереди, использующие массивы

Наилучшая реализация массива при конструировании очереди наблюдается в том случае, когда очередь организована как круговая структура. Очередь начинается с 0 и постепенно перемещается вверх до максимального размера. После того как будет занято последнее свободное место, новый элемент будет добавляться вновь в позицию 0. Разумеется, перед этим из позиции 0 должен быть удален находившийся там элемент. По этой причине по индикаторам положений 0 и максимальный размер

---

* Для выполнения этой программы вначале нужно скомпилировать в пакете `myutilities` класс `QueueException`:

```
public class QueueException extends Exception {
 public QueueException (String s) {super(s);}
}
```

а затем там же скомпилировать класс `Queue`, и уже после этого выполнять вашу программу.

(индекс) нельзя определить, является ли очередь пустой или полной. Поэтому необходим подсчет всех элементов очереди. Далее следует полное определение класса Queue, который можно добавить в пакет myutilities.

```
package myutilities;

public class Queue {
  /* Абстрактный тип данных Queue   Автор Дж.М. Бишоп, январь 1997 г.
   *                                обновлен, август 2000 г
   * Реализует очередь как ограниченный круговой массив.
   * Максимальное заданное количество
   * элементов = 100.
   */

  public Queue (int m) {
    if (m <= maxQueue)
      size = m;
    else size = maxQueue;
    front = 0;
    back = -1;
    live = 0;
    reset();
  }

  public void add (Object x) throws QueueException {
    // Генерирует объект-исключение, если очередь полная.
    if (live < size) {
      back = (back + 1) % maxQueue;
      Q[back] = x;
      live++;
    }
    else
      throw new QueueException("заполнена");
  }

  public Object remove () throws QueueException {
    // Генерирует объект-исключение, если очередь пустая.
    if (live >=1) {
      Object x = Q[front];
      front = (front + 1) % maxQueue;
      live--;
      return x;
    }
    else
      throw new QueueException("пустая");
  }

  public boolean empty () {
    return live == 0;
  }

  public boolean full () {
    return live == size;
  }

  // методы-итераторы
  public void reset () {
    now = front;
```

```
  }
  public void succ () {
    now = (now+1) % maxQueue;
  }

  public boolean eol () {
    if (back==maxQueue)
      return now == 0;
    else
      return now > back;
  }

  public Object current () {
    return Q[now];
  }

  private
  int size,          // Полная емкость вектора.
      front, back,   // Индикаторы.
      live,          // Количество используемых (занятых) мест
                     // в круговой очереди.
      now;           // Позиция для выведения очереди на экран.
  private Object Q [] = new Object [maxQueue];
  static private int maxQueue = 100;

}
```

Заметьте, что все элементы данных в очереди объявляются закрытыми (*private*) и являются переменными экземпляра, чтобы их можно было тиражировать для каждого объекта.

Очереди, использующие связные списки

Так же, как и в случае со стеками, очереди могут быть изменены только с концов. Поэтому единственное преимущество при использовании связного списка как базиса для очереди — возможность изменять ее размер. Кроме того, мы сможем назначить границы и остановить рост размера очереди. Приведенное выше определение списка можно преобразовать, воспользовавшись связными списками, мы предлагаем читателям сделать это самостоятельно. Если один класс заменить другим, то программа `WaitingRoom` будет выполняться без изменения.

15.5. И вновь связные списки

При создании связного списка (это рассмотрено в разделе 8.2) вы заметите такую особенность: при добавлении в начало списка нового элемента все имеющиеся данные перемещаются назад. Это означает, что в действительности мы создаем стек — очень простую структуру данных, имеющую только один индикатор. Если нам понадобится хранить данные в более корректном порядке, то используем два индикатора — один для начала и один для конца, как в очереди.

Фактически связные списки являются более общей структурой, чем стеки или очереди, и позволяют выполненять операции добавления или удаления не только на концах, но и в любом другом месте. Мы обнаружили это сразу же, как только в разделе 8.3 начали применять класс `LinkedList`. Чтобы закончить изучение этой темы, приведем перечень характеристик связных списков, а также пример их использования.

Характеристики связных списков

Формально мы можем определить характеристики связных списков следующим образом:

- узлы связаны друг с другом линейно в обоих направлениях, вперед и назад;
- список имеет два конца — начало и конец;
- узлы могут быть добавлены в список в любом месте;
- узлы могут быть удалены из любого места списка;
- связный список можно сканировать в обоих направлениях — вперед и назад.

Эти характеристики соответствуют классу `LinkedList`, о котором уже говорилось в главе 8, за исключением того, что теперь нам потребуется большее количество методов этого класса, которые соответствуют указанным требованиям. Ниже приведены эти дополнительные методы.

Класс `LinkedList` (дополнительные методы)

```
void addFirst (Object objectid)
void addLast (Object objectid)
void removeFirst()
void removeLast()
void add(int index, Object objectid)
void remove (int index)
int size()
```

Первые четыре метода могут эффективно использоваться для добавлений и удалений на обоих концах списка.
Чтобы делать изменения в списке, мы проходим его с помощью итератора, выполняя учет, а затем применяем метод `add` с указанием индекса.
Целое число `size` необходимо при выполнении процесса итерации.

В следующем примере показано использование этих дополнительных характеристик класса `LinkedList`.

Пример 15.4. Фотография группы студентов

Задача. Фотограф собирается сфотографировать группу студентов и хочет построить их по росту: самые высокие должны находиться в заднем ряду, причем в каждом ряду те, кто выше, находятся в центре, а остальные располагаются слева и справа по убыванию роста. Зная имена студентов и рост каждого из них, мы хотим получить план их размещения для фотографирования.

Пример. Каждый ряд должен выглядеть так, как показано на рис. 15.5.

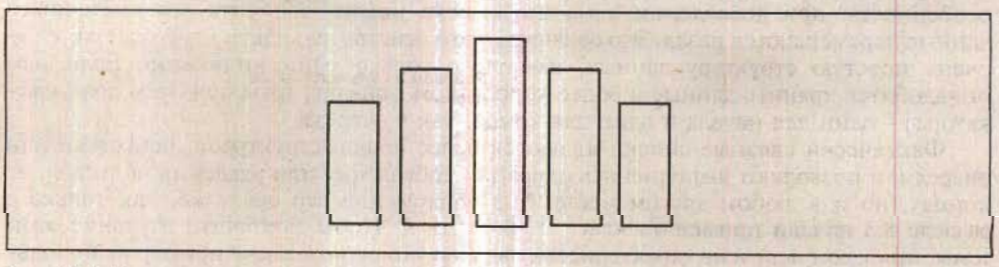


Рис. 15.5. Расположение пяти человек в ряду

Решение. Начнем с создания одного ряда, в котором все студенты будут выстроены по росту. Затем, начиная с самого высокого, начнем отбирать из ряда по нескольку человек, это должно быть постоянное число, определяющее количество людей в ряду. Тем самым мы создадим новый связный список, который будет начинаться с предполагаемой середины ряда, а элементы будут добавляться попеременно слева и справа (т.е. в начало списка и в конец). Предположим, что эти студенты будут названы буквами алфавита — `А`, `Б` и т.д., причем студент `А` будет самым высоким. Тогда ряд из семи человек будет организован следующим образом:

```
Ж Д В А Б Г Е
```

Алгоритм. В алгоритмах очень широко используются описанные выше методы класса `LinkedList`. Сначала создадим список, упорядоченный в соответствии с ростом студентов, начиная с самого высокого (рис. 15.6).

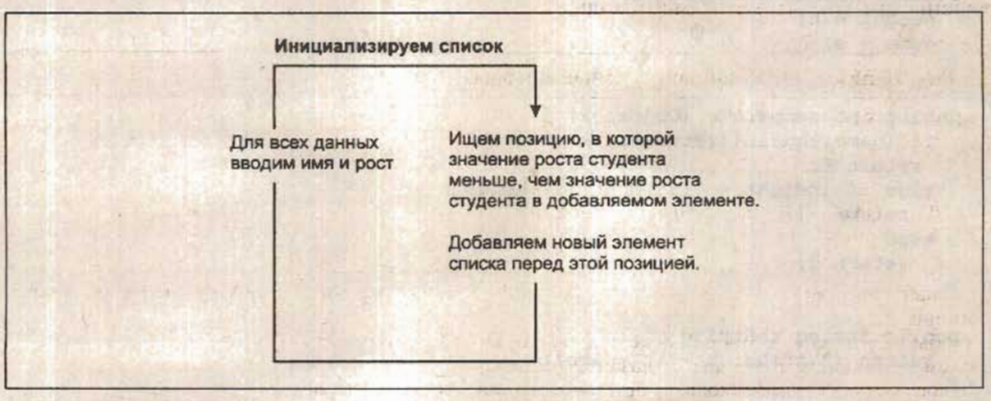


Рис. 15.6. Создание упорядоченного списка

Теперь опишем, как выбрать из списка ряд и расположить студентов слева и справа от самого высокого в этом ряду. Для простоты вычислений примем, что в ряду будет нечетное количество человек. Алгоритм приведен на рис. 15.7.

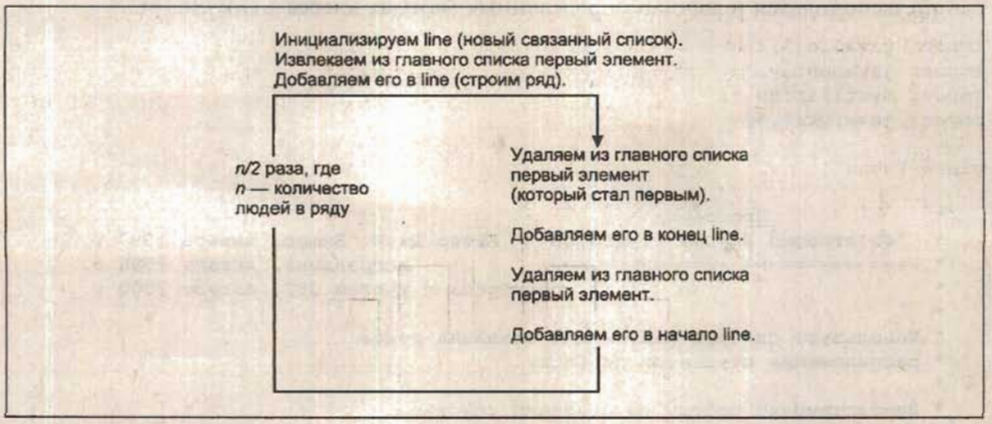


Рис. 15.7. Создание отсортированного связного списка `line`, в котором самый высокий студент находится посредине

Добавление элемента в конец списка равносильно применению метода addLast(back), а в начало списка — метода addFirst(front). Вызвав методы итератора мы можем напечатать или вывести на экран список line, затем можем удалить его и инициализировать новый список. В Java для удаления списка ему просто присваивается значение null, после чего память, выделенная для всех ассоциированных с ним узлов, будет возвращена системе.

Программа. Прежде всего мы рассмотрим тип объектов, которые должны быть отсортированы в списке. Это объекты класса Student.

```
class Student {

  double height;
  String name;

  Student(double h, String s) {
    height = h;
    name = s;
  }

  public int compareTo (Object x) {
    if (name.equals(((Student) x).name))
     return 0;
    else if (height < ((Student) x).height)
      return -1;
    else
      return 1;
  }

  public String toString () {
    return ("\t"+height+"\t"+name);
  }
}
```

Согласно определению метода compareTo объекты Student равны, если равны строковые значения их полей name. В противном случае сравниваются значения роста. Другими словами, определенный здесь класс не позволяет иметь в одном списке объекты с одинаковыми именами.

Как и прежде, программа короткая и простая для понимания, поскольку вся работа выполняется в хорошо определенных методах класса LinkedList.

```
import java.io.*;
import javagently.*;
import myutilities.*;
import java.util.*;

class Photo {

  /*              Программа
   *  "Фотография группы студентов"   Автор Дж.М. Бишоп, январь 1997 г.
   * ==============================           исправлена, январь 1998 г.
   *                              проверена с учетом JFC, август 2000 г.
   *
   * Использует связные списки для создания рядов
   * расположения студентов по росту.
   *
   * Демонстрирует работу со связными списками.
   */

  LinkedList studentList = new LinkedList();
```

```
  public static void main (String args []) throws IOException{
    new Photo();
  }

  Photo () throws IOException {
    System.out.println("**** Фотография группы ****");
    lineUp();
    for (int row = 1; studentList.size() > 0; row++)
      pickOff (row);
  }

  void lineUp () throws IOException {

    Stream in = new Stream ("students.dat", Stream.READ);
    System.out.println("Введенные данные для каждого студента:");
    while (true) {
      try {
        double h = in.readDouble();
        String n = in.readString();
        Student s = new Student (h, n);
        System.out.println(s);

        // Находим место для нового вводимого элемента.
        int before = 0;
        for (ListIterator e = studentList.listIterator(); e.hasNext();) {
          if (s.compareTo(e.next()) > 0)
            break;
          before++;
        }

        // Добавляем новый элемент в связный список.
        studentList.add(before,s);
      }
      catch (EOFException e) {break;}
    }
    System.out.println("Список студентов, упорядоченный по их росту:");

    for (ListIterator e = studentList.listIterator(); e.hasNext();)
      System.out.println(e.next());
  }

  void pickOff (int row) {
    System.out.println("Ряд "+row+"\t");
    LinkedList line = new LinkedList ();
    Student s;
    s = (Student) studentList.removeFirst();
    line.addLast(s);
    for (int i = 0; i<rowSize/2; i++) {
      s = (Student) studentList.removeFirst();
      line.addLast(s);
      s = (Student) studentList.removeFirst();
      line.addFirst(s);
    }
    for (ListIterator e = line.listIterator(); e.hasNext();)
      System.out.println(e.next());
  }

  static final int rowSize = 7;
}
```

Алгоритм. Предположим, что данные формируются следующим образом: за номером, присвоенным каждому из работников, следует список, состоящий из номеров курсов. Типичный файл, хранящий такие данные, будет выглядеть так:

```
50   1 2 3
44   3 8 7
99   1
61   1 7
```

Это означает, что работник под номером 50 собирается посещать курсы 1, 2 и 3; работник под номером 99 собирается посещать только курс 1 и т.д. Для каждого курса нам нужно множество, состоящее из номеров работников, и множество из всех имеющихся курсов. Это выполняется с помощью структур данных:

```
BitSet courses [];
BitSet schedules [];
```

Затем в алгоритм включается введение числа (номера работника), за которым следует список посещаемых им курсов. Массив `courses` проиндексирован номерами курсов, каждый из которых представляет собой битовое множество, и для каждого курса мы добавляем номер работника, посещающего этот курс. Затем мы так же добавляем номера курсов в массив `schedules`, проиндексированный номерами работников, которые, будучи элементами массива, также представляют собой битовые множества. Обратите внимание на следующие важные операторы:

```
// Помещаем работника "n" в курс "c".
courses[c].set(n);
// Помещаем курс "c" в перечень курсов, посещаемых работником "n".
schedules [n].set(c);
```

Прежде чем назначать биты в битовом множестве, мы должны позаботиться о том, чтобы объект был создан. Например, создание нового битового множества `course` будет выполнено следующим образом:

```
// Создаем курс "c", если он еще не был создан.
if (courses[c] == null)
  courses[c] = new BitSet (employeeMax);
```

Программа. Программа разбита на три части, каждая из которых представлена своим методом: `readIn`, `display` и `userCodes`. Есть еще одна особенность этой программы, не связанная с битовыми множествами, — обеспечение считывания данных, которые заканчиваются символом конца строки. Класс `Text` не пригоден для этого случая, поскольку он не различает символ конца строки. Но теперь мы можем написать наши собственные процедуры ввода, считывая числа по мере того, как они нам потребуются, при помощи объекта класса `StringTokenizer`.

Ввод начинается с получения целой строки, которая затем обрабатывается объектом-разделителем, и в ней подсчитывается количество лексем (с использованием предварительно определенного метода). После получения номера работника может быть использован цикл `for` для обработки каждого курса (т.е. битового множества, элементами которого являются номера работников, посещающих этот курс, как было описано выше). Это выполняется в методе `readIn`.

Список курсов, на которых предусматривается использование компьютера, мы получаем аналогичным образом, а студенты, записавшиеся на эти курсы, добавляются в другое битовое множество:

```
codesNeeded.or(courses[c]);
```

Остальная часть программы посвящена выводу на экран множеств.

```
import java.util.*;
import java.io.*;
import javagently.*;

public class Training {

  /*            Программа
   * "Курсы повышения квалификации"  Автор Дж.М. Бишоп, январь 1997 г.
   * ==============================   Java 1.1
   *                                  обновлена, август 2000 г.
   * Создает списки работников, записавшихся на курсы,
   * а также перечни посещаемых курсов для каждого работника,
   * включая список сотрудников, посещающих специальные курсы.
   *
   * Использует битовые множества и свой собственный класс
   * StringTokenizers, а не определенный в javagently класс Text.
  */

  BitSet courses [];
  BitSet schedules [];
  int employeeMax, courseMax;
  Stream in = new Stream (System.in);

  public static void main (String arg []) throws IOException {
    new Training ();
  }

  Training () throws IOException {

    // Главная программа объявляет два массива множеств, а затем
    // вызывает три статических метода для считывания данных и
    // вывода их на печать.
    System.out.println ("*** Курсы повышения квалификации ****");
    System.out.print("Какой наибольший номер курса?");
    courseMax = in.readInt();
    System.out.print("Какой наибольший номер работника?");
    employeeMax = in.readInt();

    // Создаем массив битовых множеств, по одному для каждого курса.
    // Размер каждого множества должен равняться числу employeeMax.
    // Сами множества создаются позже, когда будут вводиться данные.
    courses = new BitSet [courseMax];

    // Создаем массив битовых множеств, по одному для каждого работника.
    // Размер каждого множества должен равняться числу courseMax.
    // Сами множества создаются позже, когда будут вводиться данные.
    schedules = new BitSet [employeeMax];

    System.out.println("Введите перечень курсов, посещаемых работником:");
    System.out.println("Шаблон ввода: номер работника  номера курсов");
    System.out.println("Пример: 100  12  7  4  15");
    System.out.println("После перечня курсов введите пустую строку");

    readIn();
    display();
    userCodes();
  }
```

```
 void readIn () throws IOException {
  // Использует свой собственный класс StringTokenizer,
  // поскольку данные вводятся по одной строке за один проход.
  // В пустой строке нет лексем, и это является признаком завершения
  // процесса считывания данных.

  String s;
  StringTokenizer T;
  int n, c, ntokens;

  while(true) {
    s = in.readLine();
    T = new StringTokenizer (s);
    ntokens = T.countTokens ();
    if (ntokens == 0) break; // больше нет работников
    n = getInt(T);

    // Создаем перечень для работника "n."
    schedules [n] = new BitSet (courseMax);
    for (int i=0; i<ntokens-1; i++) {
      c = getInt(T);

      // Создаем курс "c", если он еще не был создан.
      if (courses[c] == null)
        courses[c] = new BitSet (employeeMax);

      // Помещаем работника "n" в курс "c".
      courses[c].set(n);

      // Помещаем курс "c" в перечень курсов,
      // посещаемых работником "n".
      schedules [n].set(c);
    }
  }
  System.out.println("Данные введены успешно.");
}

void display () {
  // Выводы на печать каждого из массивов множеств
  // немного отличаются
  // друг от друга. Для вывода на печать одного множества вызывается
  // метод printSet.

  System.out.println("Курсы и записавшиеся на них");
  System.out.println("==========================");
  for (int c =0; c<courseMax; c++) {
    System.out.print(c+": ");
    if (courses[c]==null) System.out.println("Нет желающих");
    else
      printSet(courses[c]);
  }
  System.out.println("Курсы, посещаемые работником");
  System.out.println("============================");
  for (int n = 0; n < employeeMax; n++) {
    if (schedules[n] != null) {
      System.out.print(n+": ");
      printSet(schedules[n]);
```

```
    }
  }
}

void userCodes () throws IOException {
  // Метод, демонстрирующий операцию объединения множеств.
  // Получив подмножество номеров курсов,
  // создаем объединение (метод or) всех работников, записавшихся на
  // них, и выводим на печать это множество.

  Text.prompt("На каких курсах будет нужен компьютер?");
  BitSet codesNeeded = new BitSet(employeeMax);

  String s;
  StringTokenizer T;
  int c, ntokens;

  s = in.readLine();
  T = new StringTokenizer (s);
  ntokens = T.countTokens ();
  for (int i=0; i<ntokens; i++) {
    c = getInt(T);
    codesNeeded.or(courses[c]);
  }
  System.out.println("Работники, которым будут нужны компьютеры");
  System.out.println("=========================");
  printSet(codesNeeded);
}

void printSet (BitSet b) {
  // Распечатывает элементы единственного битового множества.

fs24     for (int i = 0; i < b.size(); i++)
    if (b.get(i)) System.out.print(i+"  ");
  System.out.println();
}

int getInt (StringTokenizer T) {
  String item = T.nextToken();
  return Integer.valueOf (item.trim()).intValue();
}
}
```

Тестирование. Выполнение программы даст следующее:

```
*** Курсы повышения квалификации ****
Какой наибольший номер курса? 10
Какой наибольший номер работника? 100
Введите перечень курсов, посещаемых работником:
Шаблон ввода: номер работника  номера курсов
Пример: 100  12  7  4  15
После перечня курсов введите пустую строку
50 1 2 3
44 3 8 7
99 1
61 1 7
Данные введены успешно.
```

```
Курсы и записавшиеся на них
============================
0: Нет желающих
1: 50 61 99
2: 50
3: 44 50
4: Нет желающих
5: Нет желающих
6: Нет желающих
7: 44 61
8: 44
9: Нет желающих
Курсы, посещаемые работником
============================
44: 3 7 8
50: 1 2 3
61: 1 7
99: 1
На каких курсах будет нужен компьютер? 2 3
Работники, которым будут нужны компьютеры:
==========================
44 50
```

Резюме

В этом кратком рассмотрении алгоритмов и структур данных мы хотели подчеркнуть, что всегда существует более одного способа решения поставленной задачи. Например, для сортировки и поиска есть два метода — линейный и двоичный, эффективность которых существенно различается. Также была рассмотрена рекурсия как средство, позволяющее разработать простой и элегантный алгоритм решения.

Мы можем реализовать структуры данных с помощью массивов или связных списков, которые имеют различные возможности. Для каждой структуры мы привели определенный комплект свойств, позволяющий описать эти структуры независимо от принципов, лежащих в основе их реализации.

Были рассмотрены четыре структуры данных — стеки, очереди, связные списки и битовые множества. Три из них, которые реализованы в Java (стеки, связные списки и битовые множества) служат основой и справочным материалом для определения четвертой структуры. В примерах, демонстрирующих работу с этими структурами данных, вновь были задействованы классы и функции, которые реже употребляются в Java. Это — интерфейсы, классы-разделители и определенные пользователем объекты-исключения.

Вопросы



15.1. Если для обработки массива целых чисел потребуется линейный поиск, описанный в разделе 15.2, какую строку вы выберете?

а) `trow new ItemNotFoundException ();`

б) `for (int i = 0; i < a.length; i++)`

в) `if (x.equals(a[i])`

г) ни одну из них

15.2. Сколько сравнений потребуется выполнить двоичному поиску для списка чисел

```
8 17 25 35 41 52 60 75 86
```

чтобы найти число 35?

а) 4

б) 3

в) 5

г) 2

15.3. Двоичный поиск всегда будет работать быстрее, чем линейный, если искомый элемент находится:

а) в конце списка

б) в начале списка

в) отсутствуют в списке

г) во второй половине списка

15.4. Если мы помещаем попеременно числа от 1 до 10 в два стека, `S1` и `S2`, затем извлекаем все элементы стека `S1`, а следом за ними все элементы стека `S2`, какая последовательность будет выведена на экран после выполнения следующего кода:

```
Stack S1 = new Stack(5);
Stack S2 = new Stack(6);
for (int i = 1; i<10; 1+=2) {
  S1.push(i);
  S2.push(i+1);
}
while (!S1.empty())
  System.out.println(S1.pop()+" ");
while (!S2.empty())
  System.out.println(S2.pop()+" ");
```

а) 10 9 8 7 6 5 4 3 2 1

б) 9 10 7 8 5 6 3 4 1 2

в) 9 7 5 3 1 10 8 6 4 2

г) 10 8 7 6 4 2 9 7 5 3 2 1

15.5. В примере 15.3 (программа `WaitingRoom`) примем, что каждый период времени равен двум минутам. На основе приведенного здесь результата работы программы определите, сколько времени первому в списке пациенту пришлось ждать окончания выполнения программы:

а) 2 мин

б) 12 мин

в) 10 мин

г) 20 мин

15.6. Чтобы добавить новый элемент куда-нибудь в середину связного списка, мы должны:

а) с помощью итератора пройти по списку с самого начала, найти подходящее место и использовать метод `add(index, object)`

б) выполнить действия, указанные в варианте а) или использовать текущую позицию для указания индекса в методе `add(index, object)`

в) применить метод `contains` для нахождения нужной позиции и использовать ее как точку отсчета

г) применить метод `binarySearch` для нахождения нужной позиции, а затем использовать `add(index, object)`

15.7. В программе "Приемная доктора" (Пример 15.3) назовите, сколько времени пришлось ждать пациенту, указанному в списке под номером 12, до начала приема?

а) 10 мин

б) 9 мин

в) 11 мин

г) 1 мин

15.8. Технология, по которой метод может вызывать сам себя, называется:

а) итерацией

б) рекурсией

в) интроспекцией

г) инверсией

15.9. Дайте определение новому битовому множеству, хранящему информацию обо всех курсах, на которые записался любой из работников (Пример 15.5):

а) `BitSet allCourses = new BitSet (courseMax);`

б) `BitSet allCourses [] = new BitSet (courseMax);`

в) `BitSet allCourses = new BitSet [courseMax];`

г) `BitSet allcourses.setMax (courseMax);`

15.10. Каким должен быть оператор, который можно добавить в метод `readIn` для обновления данного множества всех курсов (`allCourses`) во время обработки перечня курсов, на которые записался работник?

а) `allCourses.bitOn(c);`

б) `allCourses [c].set();`

в) `allCourses.set(c);`

г) ни один из них

Задачи

15.1. **Ускорение поиска в алфавитном списке.** Как в задаче 7.5 можно ускорить сортировку алфавитного списка и поиск в нем?

15.2. **Шпион, считывающий стеки.** Предположим, что мы хотим вывести на печать содержимое стека, не уничтожая его и не используя ничего, кроме методов `push` и `pop`. Напишите программу, позволяющую выполнить это задание.

15.3. **Статистика ожидания.** Из примера 15.3 видно, что некоторым пациентам приходится долго ожидать приема доктора. Создайте программу, после выполнения которой в правой стороне экрана будет выведено значение времени ожидания пациентов в приемной.

15.4. **Очереди на основе связного списка.** Реализуйте класс `Queue`, базирующийся на связном списке. Проверьте его.

15.5. **Стеки для получения палиндромов.** Решите задачу 7.2, используя стеки вместо массивов.

15.6. **Выигрыш по жребию.** На вечеринке каждому раздается пронумерованный билет. Затем тянется жребий, определяющий номер, обладатель которого выигрывает приз, кроме него призы получают все члены его семьи. Напишите программу, в которой был бы составлен список членов семей всех играющих, организована жеребьевка и выведены на экран имена всех членов семьи победителя.

15.7. **Сортировка списков.** Связные списки можно отсортировать следующим образом. Создайте список с теми же параметрами. Добавляйте в него каждый элемент, используя сравнение в соответствии с поставленными условиями. После этого новый список будет отсортированной версией старого. Примените этот метод в задаче 8.7.

15.8. **Электронное расписание.** Студенты университета Саванны на третьем курсе должны разработать групповой проект. Количество групп и студентов в группах изменяется из года в год. Одной из проблем, возникающих в начале учебного года, становится выбор времени встречи, когда студенты конкретной группы могли бы обсудить свой проект, поскольку каждый из них может посещать различные занятия и иметь различную загруженность. Поэтому они хотели бы использовать компьютер для назначения удобного для всех времени. Создайте такую программу, применяя связные списки и битовые множества.

ПРИЛОЖЕНИЕ А



Обозначения, используемые в схемах

Часть I. Строительные блоки

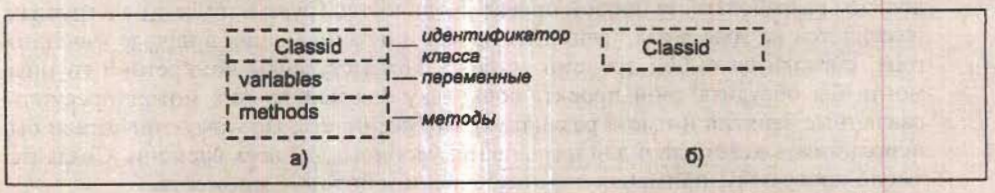


Рис. А.1. Элемент **Класс** (а — длинное обозначение, б — короткое обозначение)

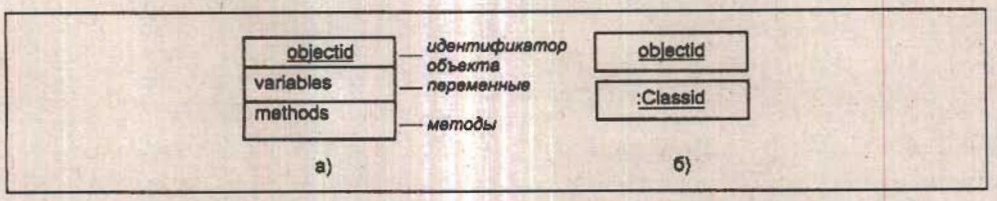


Рис. А.2. Элемент **Объект** (а — длинное обозначение, б — короткое обозначение)

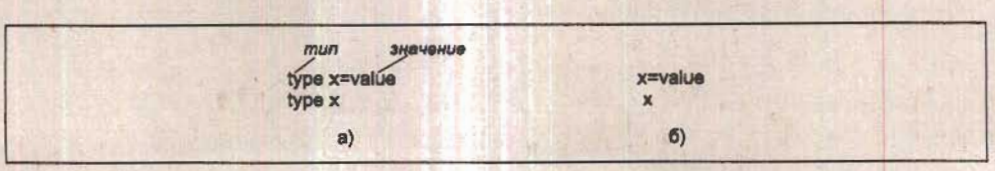


Рис. А.3. Элемент **Простые переменные** (а — длинное обозначение, б — короткое обозначение)

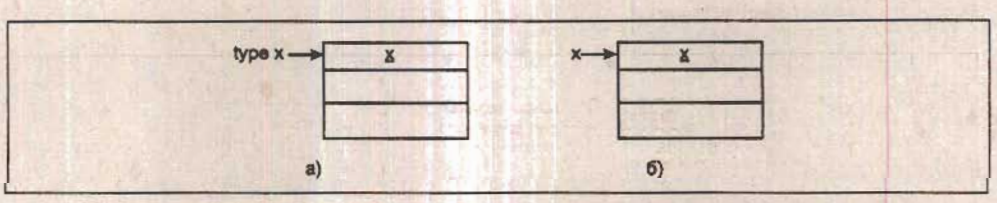


Рис. А.4. Элемент **Объектные-переменные** (а — длинное обозначение, б — короткое обозначение)

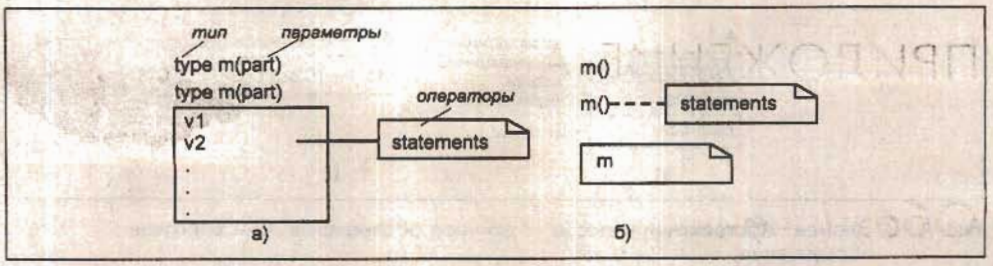


Рис. А.5. Элемент **Метод** (а — длинное обозначение, б — короткое обозначение)

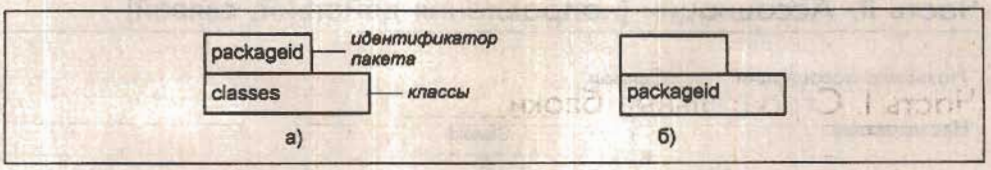


Рис. А.6. Элемент **Пакет** (а — длинное обозначение, б — короткое обозначение)

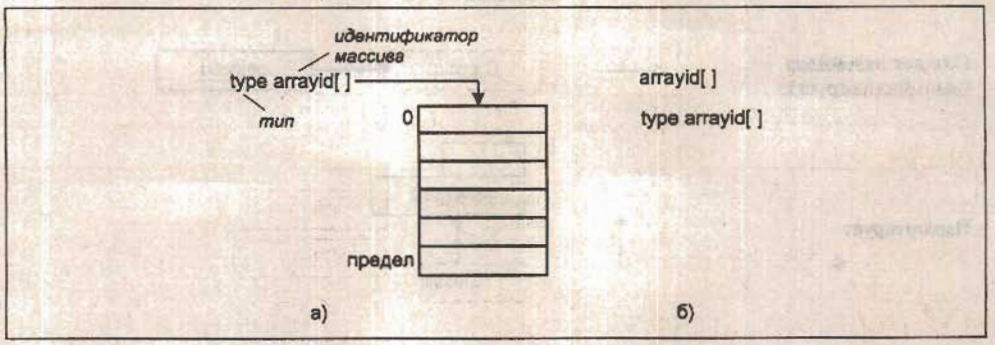


Рис. А.7. Элемент **Массив** (а — длинное обозначение, б — короткое обозначение)

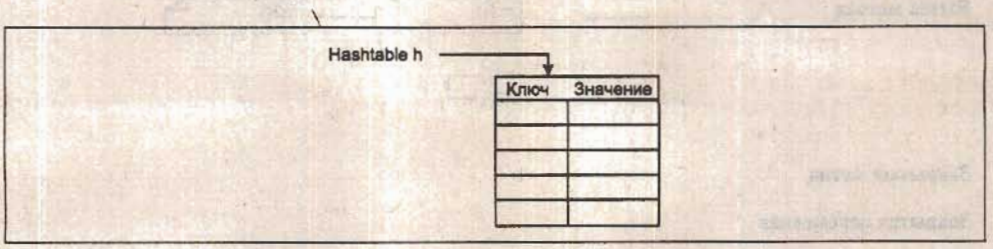


Рис. А.8. Элемент **Хеш-таблица**

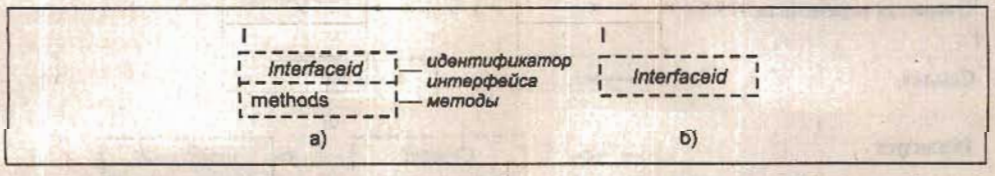


Рис. А.9. Элемент **Интерфейс** (а — длинное обозначение, б — короткое обозначение)

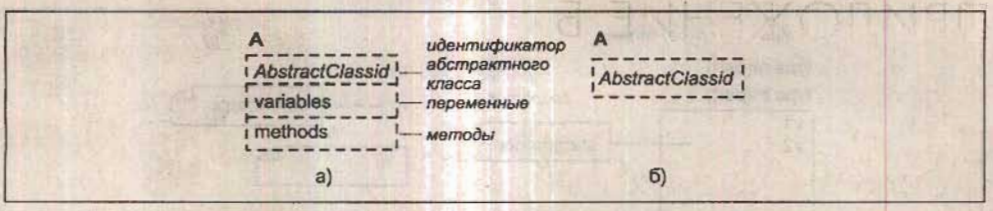
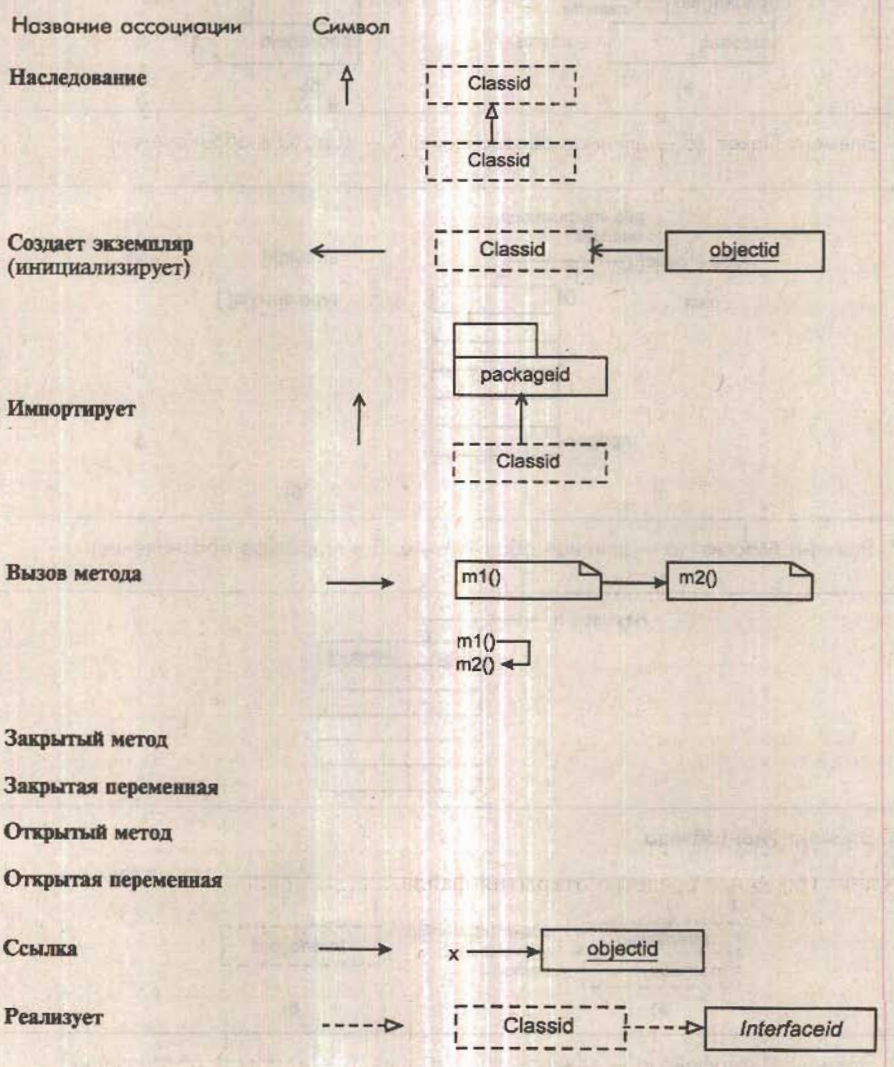


Рис. А.10. Элемент **Абстрактный класс** (а — длинное обозначение, б — короткое обозначение)

Часть II. Ассоциации (направления действий, связей)



ПРИЛОЖЕНИЕ Б



Перечень описаний конструкций языка Java

Синтаксис и семантика конструкций языка Java приведены на следующих страницах:

Пример 2.1. Welcom 26
Пример 2.2. Рисование флага 27
Пример 2.3. Лавка древностей 29
Пример 2.4. Вывод предупреждения 44
Пример 2.5. Прейскурант лавки древностей 45
Пример 2.6. Расписание маршрутов автопарка 51

Пример 3.1. Исследование класса `Math` 62
Пример 3.2. Печать набора этикеток 66
Пример 3.3. Таблица преобразований 68
Пример 3.4. Гистограмма уровня осадков 73
Пример 3.5. Объем продаж в лавке древностей 79
Пример 3.6. Большая таблица преобразований 81

Пример 4.1. Приветствие 109
Пример 4.2. Тестирование класса `Time` 114
Пример 4.3. Форматирование большой таблицы температур 118
Пример 4.4. Еще один вариант программы приветствия 119
Пример 4.5. Поиск наибольшего числа 123
Пример 4.6. Вычисление среднего арифметического 125
Пример 4.7. Нумерация деталей 130
Пример 4.8. Нумерация деталей 135
Пример 4.9. Надежное средство открытия файла 138
Пример 4.10. Олимпийские медали 139

Пример 5.1. Наибольший общий делитель 153
Пример 5.2. Повседневная работа лавки древностей 155

Пример 5.3. Проверка наличия товаров на складе ... 163
Пример 5.4. Вывод на печать набора символов ... 166
Пример 5.5. Курсы обмена валют ... 169

Пример 6.1. Определение частоты употребления значений ... 187
Пример 6.2. Соревнования по прыжкам в воду ... 194
Пример 6.3. Статистический анализ ... 198
Пример 6.4. Время службы лампочек ... 199
Пример 6.5. Поиск месторождений золота ... 203
Пример 6.6. Определение уровня осадков ... 209
Пример 6.7. Сортировка слов в файле ... 216
Пример 6.8. Конвертер валют ... 222

Пример 7.1. Удаление двойных пробелов ... 236
Пример 7.2. Поддельные чеки ... 238
Пример 7.3. Где я? ... 253
Пример 7.4. Таблица конвертирования валют ... 256
Пример 7.5. Дни для покупок перед Рождеством ... 259
Пример 7.6. Форматирование и синтаксический анализ дат ... 262
Пример 7.7. Тестирование часовых поясов ... 264
Пример 7.8. Многоязычные даты ... 265

Пример 8.1. Кофейный магазин Нельсона ... 274
Пример 8.2. Составление списка ... 283

Пример 9.1. Иерархия `Nature` ... 312
Пример 9.2. Сортировка ... 317
Пример 9.3. Применение интерфейса `Serializable` ... 333

Пример 10.1. Вывод предупреждения о возможном заражении вирусом ... 346
Пример 10.2. Гистограмма уровней осадков в Саванне ... 348
Пример 10.3. Несколько флагов ... 351
Пример 10.4. Предупреждение с двумя ответами ... 358
Пример 10.5. Светофор ... 362

Пример 11.1. Светофор с кнопками ... 372
Пример 11.2. Настройка светофора ... 384
Пример 11.3. Конвертер валют с GUI ... 388

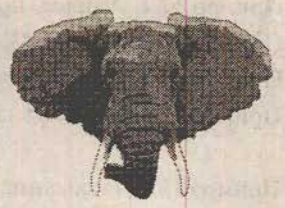
Пример 12.1. Апплет, предупреждающий о вирусе ... 413
Пример 12.2. Апплет "Касса супермаркета" ... 416
Пример 12.3. Счастливое имя ... 423
Пример 12.4. Поймать гнома ... 433

Пример 13.1. Рисование точек ... 453
Пример 13.2. Программа "Светофор" с мигающими огнями ... 458
Пример 13.3. Светофор для пешеходов ... 463
Пример 13.4. Автопарки, обслуживающие живописные места ... 471

Пример 14.1. Расчет операции конвертирования валюты в апплете ... 494
Пример 14.2. Считывание изображения и вывод его на экран ... 496
Пример 14.3. Открытие порта ... 498
Пример 14.4. Создание АТМ-сервера ... 499
Пример 14.5. База данных конвертера валют ... 509
Пример 14.6. Сервер, генерирующий "магические числа" ... 518

Пример 15.1. Интерактивный двоичный поиск ... 539
Пример 15.2. Перевернутое предложение ... 548
Пример 15.3. Приемная доктора ... 551
Пример 15.4. Фотография группы студентов ... 556
Пример 15.5. Курсы повышения квалификации ... 561

ПРИЛОЖЕНИЕ В



Список программ, апплетов и классов

`Animated`	Программа	Пример 15.1
`ATMServer`	Программа применяющая сокеты и подпроцессы	Пример 14.4
`Batch`	Класс	Пример 8.1, практикум 3
`ButtonTest`	GUI-программа	Пример 10.4
`CarPark`	Апплет	Пример 13.4
`CarThread`	Класс-подпроцесс	Пример 13.4
`CatchM`	Апплет	Пример 12.4
`ChatHandler`	Класс-подпроцесс	Практикум 10
`ChatServer`	Пограмма, использующая сокеты	Практикум 10
`Checkin`	Апплет	Практикум 11
`ChequeDetector`	Программа	Пример 7.2
`Coffee`	Класс	Пример 8.1
`Coffee2`	Класс	Практикум 3
`CoffeeShop`	Программа	Пример 8.1
`CoffeeShop2`	Программа	Практикум 3
`Competition`	Апплет	Пример 12.3
`Converter`	Программа	Пример 6.8
`ConverterDatabase`	Программа с базой данных	Пример 14.5
`ConverterGUI`	GUI-программа	Пример 11.3
`ConverterGUIApplet`	Апплет для работы в сети	Пример 14.1
`ConverterSwing`	GUI-программа	Практикум 7
`Counter`	Класс-подпроцесс	Практикум 9
`Country`	Класс	Пример 9.2
`Curio`	Класс	Примеры 2.3, 3.5, 5.2
`CurioStore1`	Программа	Примеры 2.3, 2.5
`CurioStore2`	Программа	Пример 3.5
`CurioStore3`	Программа	Примеры 5.2, 5.3, 5.5
`Currency`	Программа	Пример 7.4
`CustomDates`	Программа	Пример 7.6
`Display`	Класс пакета `javagently`	Разделы 4.2, 13.4
`DisplayWarning`	Программа	Пример 2.4
`Dive`	Класс	Пример 6.2

DivingCompetition	Программа	Пример 6.2
FetchImage	Сетевая программа	Пример 14.2
Filer	Класс пакета myutilities	Пример 4.9
Flag	Класс пакета graphics	Пример 2.2
FlagCanvas	Класс пакета graphics	Пример 10.3, практикум 5
FlagCanvasMotif	Класс пакета graphics	Практикум 5
FlagCanvasVerti	Класс пакета graphics	Практикум 5
FlagMaker1	Программа, использующая графику	Пример 2.2
FlagMaker2	Программа, использующая графику	Пример 10.3
FlagMakerGUI	GUI-программа	Практикум 5
FleetCalculator	Программа	Пример 2.6
Frequencies	Программа	Пример 6.1
GoldExploration	Программа	Пример 6.5
Graph	Класс пакета javagently	Раздел 6.3
Greetings	Программа	Пример 4.1
Greetings2	Программа	Пример 4.4
HCFRepeat	Программа	Пример 5.1
HighestValue	Программа	Пример 4/5
LargeTemperatureTable	Программа	Примеры 3.6, 4.3
LightBulbAnalysis	Программа	Пример 6.4
LinkTest	Программа	Пример 8.2
LotsaLabels	Программа	Пример 3.2
MagicNumber	Класс для работы с удаленным объектом	Пример 14.6
MagicNumberClient	Программа-клиент	Пример 14.6
MagicNumberServer	Программа-сервер	Пример 14.6
Museum	Класс, реализующий интерфейс synchronized	Практикум 9
Olympics	Программа	Пример 4.10
OurInteger	Класс	Пример 9.2
PartCodes	Программа	Примеры 4.7, 4.8
Photo	Программа	Пример 15.4
PizzaDelivery1	Программа	Практикум 1
PizzaDelivery2	Программа	Практикум 1
PlayGame	Класс	Практикум 2
Ports	Программа, использующая сокеты	Пример 14.3
Queue	Класс пакета myutilinies	Раздел 15.4
RainfallHistogram	Программа	Пример 3.4
RandomInvestigation	Программа	Пример 3.1
Rates	Класс	Пример 6.8, практикум 7
ReadingsAverages	Программа	Пример 4.6
Reverser	Программа	Пример 15.2
RSPGame	Программа	Практикум 2
SetOfLights	Класс-подпроцесс	Примеры 13.2, 13.3
SetUpDatabase	Класс, применяющий базу данных	Пример 14.5

Shopping	Программа	Пример 7.5
Sort	Класс пакета myutilities	Пример 9.2
Sortable	Интерфейс пакета myutilities	Пример 9.2
SortTest	Программа	Пример 9.2
SortTestInt	Программа	Пример 9.2
SortWords	Программа	Пример 6.7
Spaces	Программа	Пример 7.1
SpotTest	Программа с подпроцессами	Пример 13.1
Stats	Класс пакета myutilities	Пример 6.3
Stream	Класс пакета javagently	Разделы 4.2, 7.3
Student	Класс	Пример 15.4
Students	Класс	Пример 9.3
Tags	Класс	Практикум 4
TemperatureTable	Программа	Пример 3.3
TestFiles	Программа	Пример 9.3
TestNature	Программа	Пример 9.1
TestUnicode	Программа	Пример 5.4
Till	GUI-программа	Практикум 6
TillApplet	Апплет	Пример 12.2
Time	Класс	Практикум 1, пример 4.2
TimeDisplay	Программа	Пример 4.2
Traffic0	GUI-программа	Пример 10.5
Traffic1	GUI-программа	Примеры 11.1, 11.2
Traffic2	GUI-программа	Пример 13.2
Traffic3	GUI-программа	Пример 13.3
Training	Программа	Пример 15.5
Transferable	Интерфейс для активизации удаленных методов	Пример 14.6
TreesAp	Апплет	Практикум 8
VaryDates	Программа	Пример 7.8
VeterinaryTags	Программа	Практикум 4
ViewPoint	Класс, реализующий интерфейс synchronized	Пример 13.4
VirusWarning	Программа, использующая графику	Пример 10.1
Visitors	Клас-подпроцесс	Практикум 9
WaitingRoom	Программа	Пример 15.3
WalkmanHire	Программа	Практикум 9
WarningApplet	Апплет	Пример 12.1
Weather	Программа	Пример 6.6
WeatherChart	Программа, использующая графику	Пример 10.2
Welcome	Программа	Пример 2.1
WhereAmI	Программа	Пример 7.3
WorldTime	Программа	Пример 7.7
Xtags	Класс	Практикум 4

ПРИЛОЖЕНИЕ Г



Ответы на вопросы для самоконтроля

№ главы \ № вопроса	1	2	3	4	5	6	7	8	9	10
2	б	в	б	а	г	б	б	б	а	г
3	в	б	в	а	б	а	в	г	в	г
4	б	в	в	г	в	б	г	б	б	а
5	б	г	г	г	б	а	в	а	в	а
6	в	в	б	в	а	б	б	г	в	в
7	а	в	г	г	г	б	б	б	в	б
8	б	в	б	б	г	г	г	в	б	в
9	б	а	б	в	а	в	б	б	в	а
10	б	в	г	б	б	б	б	в	в	г
11	в	б	б	а	б	а	б	а	б	г
12	б	в	в	в	г	б	б	г	б	а
13	г	г	а	г	в	б	б	в	б	б
14	в	а	б	г	б	б	б	б	в	в
15	в	б	г	в	б	б	а	б	в	в



Предметный указатель

А

Абстрактный класс 323, 108
Абстрактный метод 323
Абстракция 310
Адаптер WindowAdapter 347

Б

Булева переменная 458
Булево выражение 85
Булево условие 64

В

Вложенный класс 292
Внутренний класс 292
Выражение 47
— булево 85
— логическое 64

Д

Действительный параметр 72
— объявление 73
Динамическое связывание 316

И

Идентификатор 35
Интерфейс 312, 314, 317
— actionListener 356
— AudioClip 430
— Cloneable 288
— Image 431
— ImageProducer 496
— JDBC 508
— LayoutManager 342, 356
— Observer 431
— ODBC 508
— Remote 518
— RMI 517
— Runnable 453
— Serializable 332
— Sortable 317
— Transferable 519, 520
— WindowListener 347
Исключение 133
— обработка 132, 133, 135, 137, 139, 141
— определяемое пользователем 162, 163
— ArithmeticException 139
— EOFException 134, 135
— FileNotFound 136
— FileNotFoundException 139
— IOException 135
— NumberFormatException 139
Итератор 295

К

Класс:
— абстрактный 343, 108
— вложенный 292
— внутренний 292
— Animated 540
— Applet 414
— ArrayList 294
— Arrays 535
— ArrayUtils 192
— ATMServer 500
— Batch 276
— Boolean 243
— BufferedReader 108, 109, 236, 372
— Button 355
— ButtonTest 358
— Canvas 352, 361
— CarPark 471
— CatchM 434
— Character 165, 243
— ChatHandler 506

— ChatServer ... 505
— Checkbox ... 361
— Checkin ... 525
— ChequeDetector ... 239
— Choice ... 361, 374
— Coffee ... 277
— Coffee2 ... 299
— CoffeeShop ... 279
— Color ... 345
— Competition ... 427
— Component ... 342, 344
— Container ... 342
— ConverterDatabase ... 514
— ConverterGUIApplet ... 494
— Counter ... 484
— Country ... 319
— Curio ... 290
— Currency ... 257
— CustomDates ... 262
— Date ... 258
— DateFormatSymbols ... 267
— DecimalFormat ... 251, 256
— Display ... 474, 477
— Dive ... 194
— Double ... 243, 245
— DriverManager ... 508
— EventObject ... 374
— Filer ... 199
— FlagMaker2 ... 352
— FlagMakerGUI ... 379
— Float ... 243
— Font ... 349
— FontMetrics ... 342, 346
— Frame ... 343
— Graph ... 121, 207
— Graphics ... 342, 345
— Hashtable ... 221
— Herbivores ... 311
— Image ... 342, 496
— InetAddress ... 500
— InputStream ... 108, 244
— InputStreamReader ... 493
— IntArray ... 191
— Integer ... 243
— ItemNotFoundException ... 537
— JButton ... 400
— JColorChooser ... 400
— JLabel ... 400, 402
— JSlider ... 400
— Judge ... 194
— JComboBox ... 400
— Label ... 355
— LightCanvas ... 363, 457
— ligths ... 363
— LinkedList ... 294, 556
— LinkTest ... 283
— Locale ... 252, 262
— MagicNumber ... 519
— MagicNumberClient ... 519, 520
— MagicNumberServer ... 519, 520
— Math ... 61, 172
— MediaTracker ... 432
— MenuComponent ... 342
— Menus ... 361
— MouseEvent ... 374, 432
— MouseHandler ... 433
— Museum ... 482, 483
— Nature ... 311
— NumberFormat ... 251, 254
— Object ... 221, 286, 536, 548
— Panel ... 363
— Photo ... 558
— PopupMenu ... 361
— Ports ... 498
— Print ... 361
— PrintJob ... 342
— PrintStream ... 42
— Queue ... 554
— Rates ... 223, 401
— ResultSet ... 508
— Reverser ... 548
— ScrollBar ... 360
— ScrollPane ... 360
— ServerSocket ... 498
— Set ... 561
— SetOfLights ... 459, 467
— SetUpDatabase ... 512
— Shopping ... 259
— Socket ... 498
— Sort ... 318
— SortTest ... 320
— SortTestInt ... 321
— Spaces ... 237
— Spots ... 453
— SpotTest ... 453
— Stack ... 548
— Statement ... 508
— Stats ... 199
— Stream ... 239, 246, 372
— String ... 234

— StringTokenizer ... 242, 562
— Student ... 558
— System ... 42
— Tags ... 325
— TestDates ... 261
— TestFiles ... 334
— TestNature ... 312
— Text ... 562
— TextArea ... 360, 370
— TextComponent ... 360, 370
— TextField ... 360, 370
— Thread ... 453, 455
— Till ... 395
— TillApplet ... 417
— Toolkit ... 342
— Traffic0 ... 363
— Traffic1 ... 385
— Traffic2 ... 461
— Traffic3 ... 464
— Training ... 563
— TreesAp ... 441
— URLConnection ... 492
— VaryDates ... 265
— Vector ... 219, 548
— VeterinaryTags ... 327
— VirusWarning ... 346
— Visitors ... 481, 485
— WaitingRoom ... 552
— WalkmanHire ... 482, 485
— WarningApplet ... 413
— Watcher ... 476
— WeatherChart ... 350
— WhereAmI ... 253
— Window ... 343
— WindowAdapter ... 347
— WorldTime ... 264
— XTags ... 324
Клонирование ... 287
Ключевое слово ... 35
Конкатенация ... 45
Константа ... 118
Конструктор ... 41

Л

Лексема ... 241
Логическое выражение ... 64

М

Массив:
— использование ... 187, 198
— многомерный ... 202
— простой ... 184
— создание ... 184
— характеристики ... 188
Менеджер размещения ... 356
— BorderLayout ... 356, 360
— BoxLayout ... 360
— CardLayout ... 360
— FlowLayout ... 356, 361
— GridBagLayout ... 360, 525
— GridLayout ... 360, 393
Метод:
— абстрактный ... 323
— вызов ... 71
— объявление ... 70, 71
— перегрузка ... 315
— переопределение ... 315
— синхронизированный ... 470
— создание ... 70, 71
— типизированный ... 63, 78, 79, 81, 83
— accept ... 498
— actionPerformed ... 356
— add ... 207, 294, 295, 355, 370, 551, 556
— addImage ... 432
— addWindowListener ... 347
— after ... 258
— and ... 561
— assessScores ... 195
— binarySearch ... 535, 539
— bind ... 518
— charAt ... 235
— clear ... 294, 561
— clearRect ... 345
— clone ... 287
— close ... 244
— compareTo ... 87, 235, 258
— connectTo ... 494
— contains ... 221, 294
— containsKey ... 221
— copyArea ... 345
— createImage ... 496
— destroy ... 413
— display ... 540
— draw ... 453
— drawChars ... 345
— drawImage ... 431
— drawLine ... 345
— drawOval ... 365
— drawPolygon ... 379
— drawRect ... 345

— drawString ... 345, 346
— empty ... 548, 551
— equals ... 235, 258, 287, 535
— executeQuery ... 508
— executeUpdate ... 508
— fill ... 536
— fillOval ... 345
— fillPolygon ... 379
— fillRect ... 28, 345
— format ... 244
— full ... 551
— get ... 561
— getActionCommand ... 374
— getAudioClip ... 430
— getCodeBase ... 430, 493
— getDocumentBase ... 493
— getDouble ... 475
— getEntry ... 475
— getGraphics ... 457
— getInt ... 475
— getItemSelectable ... 375
— getParameter ... 422
— getSource ... 374
— getString ... 475
— getText ... 371
— getTime ... 258
— hasMoreTokens ... 242
— hasNext ... 295
— hasPrevious ... 295
— indexOf ... 235
— init ... 414, 453
— initializeDisplay ... 475
— insertPrompt ... 475
— intValue ... 243
— itemStateChanged ... 374
— length ... 235
— lessThan ... 319
— loop ... 430
— main ... 36, 63, 453
— makeFlag ... 352, 379
— mean ... 198, 540
— mousePressed ... 432, 433, 464
— next ... 295
— nextElement ... 221
— nextGraph ... 207
— nextToken ... 242
— notify ... 474
— notifyAll ... 474
— or ... 561
— paint ... 28, 344
— parseInt ... 243
— peek ... 547
— play ... 430
— pop ... 547
— previous ... 295
— print ... 42, 244, 498
— printLine ... 88
— println ... 42, 109, 244
— prompt ... 475
— push ... 547
— read ... 498
— readChar ... 239, 240, 244
— readInt ... 244
— readObject ... 332
— readString ... 241, 244
— rebind ... 518
— remove ... 221, 294, 295, 551
— repaint ... 344
— run ... 453
— search ... 216
— selectionSort ... 215
— sendMessage ... 527
— set ... 295, 561
— setBackground ... 346
— setColor ... 207, 345
— setFont ... 345
— setForeground ... 346
— setLayout ... 356
— setline ... 207
— setLocation ... 352
— setMaximum ... 255
— setScore ... 195
— setSize ... 343
— setSymbol ... 207
— setTime ... 258
— setTitle ... 207, 343, 354
— setVisible ... 343
— size ... 294
— sleep ... 456
— sort ... 535
— start ... 414, 456
— stddev ... 198, 540
— stop ... 415, 430
— Stream.format ... 255
— substring ... 235
— System.exit ... 347
— toCharArray ... 235, 241, 425
— toLowerCase ... 425
— toString ... 50, 60, 190, 243
— transactions ... 224

— trim ... 245, 425
— valueOf ... 236, 243
— void ... 70
— wait ... 474
— waitForID ... 432
— windowClosing ... 347
— writeObject ... 332
— xor ... 561
— yield ... 456
Модификатор:
— abstract ... 289
— final ... 289
— private ... 289
— protected ... 314
— public ... 289
— static ... 289
— synchronized ... 289

Н

Наблюдатель ... 476
Наследование ... 310, 312

О

Объект ... 37
— инициализация ... 152
— объявление переменных ... 38
— создание ... 38
— System.in ... 108, 137
— System.out ... 108
Объект-исключение:
— IllegalMonitorStateException ... 474
— InterruptedException ... 457
— MalformedURLException ... 492
— ParseException ... 372
— RemoteException ... 521
Оператор:
— булев ... 85
— break ... 153
— do ... 160
— for ... 64, 65, 67, 69
— switch ... 167, 169, 171
Операция:
— конкатенации ... 45
— присваивания ... 48
Отображение рисунков ... 496

П

Пакет:
— java.awt ... 341
— java.lang ... 453
— java.net ... 492
— java.rmi ... 492, 517
— java.sql ... 492
— java.text ... 246, 251, 258
— java.util ... 172, 219, 242, 252, 258
— javagently ... 199
— javax.swing ... 341
— jgeslib ... 423
— Swing ... 342, 399
Параметр:
— действительный ... 72
— действительный, объявление ... 73
— формальный ... 72
— формальный, объявление ... 73
Перегрузка метода ... 315
Переменная ... 37
— булева ... 458
— скрытие ... 315
— super ... 313
— this ... 286
Переопределение метода ... 315
Подключение к базе данных ... 508
Подпакет:
— java.awt.event ... 341, 365
— java.awt.image ... 365
— java.rmi.server ... 521
Подпроцесс ... 451
— создание ... 455
Поиск:
— двоичный ... 537
— линейный ... 536
Порт ... 497
Программа:
— jar ... 423
— keytool ... 422
— rmic ... 519
— rmiregistry ... 519

Р

Разделитель ... 241
Региональный стандарт ... 251
Рекурсия ... 538

С

Связывание, динамическое ... 316
Сериализация ... 332
Сигнализация ... 474
Символ, преобразование ... 165
Синхронизация ... 469

Системный вид:
— CrossPlatform ... 403
— Macintosh ... 403
— Metal ... 403
— Motif ... 403
— System ... 403
— Windows ... 403
Скрытие переменной ... 315
Слово, ключевое ... 35
Слушатель:
— ActionListener ... 374
— ItemListener ... 374
— MouseListener ... 432, 464
— MouseMotionListener ... 433
Событие ... 347
— ActionEvent ... 357, 374, 375
— ItemEvent ... 374
Соединение URL ... 492
Создание:
— апплета ... 412
— гистограммы ... 348
— графика ... 208
— иерархии ... 312
— кнопки выбора ... 380
— окна ... 358
— панели ... 361
— подпроцесса ... 455
— простейшего HTML ... 413
— раскрывающегося списка ... 380
— связного списка ... 282
— текстового поля ... 370
— утилиты ... 198
— хеш-таблицы ... 221
— ATM-сервера ... 499
Сокет ... 497
Сортировка:
— Quicksort ... 542
— методом отбора ... 214, 545
Стиль форматирования ... 260
Структура данных:
— битовое множество ... 561
— древовидная ... 535
— массив ... 184, 535
— множество ... 535
— очередь ... 535, 549
— последовательный файл ... 535
— расширяемый массив ... 535
— связный список ... 293, 535
— словарь ... 219, 535
— стек ... 535, 547
— файл произвольного доступа ... 535
— хеш-таблица ... 220, 535

Т

Тип данных ... 59
— преобразование ... 63
— boolean ... 59, 84, 85, 87
— byte ... 59
— char ... 59, 164, 165
— double ... 59
— float ... 59
— int ... 59
— long ... 59
— short ... 59

У

Условие булево ... 64

Ф

Формальный параметр ... 72
— объявление ... 73
Форматирование данных при выводе ... 255

Ц

Цикл ... 64
— бесконечный ... 70
— вложенный ... 70
— прерывание ... 70
— пустой ... 69
— с обратным отсчетом ... 69
— условный ... 64
— do ... 152, 153, 155, 157, 159, 161
— for ... 64
— while ... 152-153, 155, 157, 159, 161



Содержание

Предисловие к третьему изданию 5

ГЛАВА 1. Введение **13**

1.1. Добро пожаловать в Java 13
1.2. Разработка программного обеспечения 17
1.3. Методика 20
1.4. Начинаем работать с Java 22
Резюме 25
Вопросы 25

ГЛАВА 2. Простые программы **26**

2.1. Демонстрационные программы 26
2.2. Структура программы 31
2.3. Основы объектного ориентирования 36
2.4. Знакомимся с методами вывода 42
2.5. Простые вычисления 46
Резюме 54
Вопросы 55
Задачи 57

ГЛАВА 3. Типы и методы **59**

3.1. Поговорим о типах 59
3.2. Повторение с помощью цикла for 64
3.3. Создание методов 70
3.4. Передача параметров и возвращение результатов 72
3.5. Типизированные методы 78
3.6. Логический тип boolean 84
3.7. Практикум 1. Доставка пиццы 2U 88
Резюме 101
Вопросы 102
Задачи 104

ГЛАВА 4. Ввод и вывод **108**

4.1. Интерактивный ввод данных 108
4.2. Дружественный интерфейс ввода-вывода 110
4.3. Создание пакетов и доступ к ним 120

4.4. Организация выбора посредством оператора if-else 122
4.5. Ввод данных в файл и вывод данных из файла 129
4.6. Обработка исключений 132
Резюме 142
Вопросы 143
Задачи 144

ГЛАВА 5. Управление потоком 148

5.1. Требования к программе 148
5.2. Циклы while и do с выходом по условию 152
5.3. Исключения, определяемые пользователем 162
5.4. Тип char 164
5.5. Оператор switch 167
5.6. Практикум 2. Игра "Камень-ножницы-бумага" 172
Резюме 178
Вопросы 178
Задачи 181

ГЛАВА 6. Массивы и таблицы 184

6.1. Простые массивы 184
6.2. Таблицы 202
6.3. Класс Graph 207
6.4. Сортировка и поиск 213
6.5. Таблицы, не зависящие от класса 219
Резюме 227
Вопросы 227
Задачи 229

ГЛАВА 7. Форматирование 233

7.1. Строки и их обработка 233
7.2. Разделители и оболочки 241
7.3. Структура класса Stream 244
7.4. Форматирование чисел 251
7.5. Даты, календари и время 258
Резюме 268
Вопросы 269
Задачи 271

ГЛАВА 8. Объекты: применение на практике 274

8.1. Разработка объектно-ориентированной программы 274
8.2. Свойства объектов 282
8.3. Списки и итераторы 293
8.4. Практикум 3. Кофейный магазин Нельсона 297
Резюме 305
Вопросы 306
Задачи 308

ГЛАВА 9. Абстракция и наследование 310
9.1. Способы использования классов 310
9.2. Наследование 312
9.3. Абстракция через интерфейсы 317
9.4. Абстрактные методы и классы 323
9.5. Практикум 4. Ветеринарные бирки 324
9.6. Сериализация 332
Резюме 337
Вопросы 338
Задачи 339
ГЛАВА 10. Графика и пользовательские интерфейсы 341
10.1. Введение в awt и Swing 341
10.2. Размещение графики в окне 343
10.3. Компоновка пользовательского интерфейса 354
Резюме 365
Вопросы 366
Задачи 368
ГЛАВА 11. Управление событиями 370
11.1. Взаимодействие с текстовыми полями 370
11.2. События слушатели и обработчики событий 374
11.3. Практикум 5. Разработчик флагов 375
11.4. Работа с последовательностями событий 383
11.5. Практикум 6. Касса супермаркета 393
11.6. Практикум 7. Программа "Конвертер валют", использующая пакет Swing 399
Резюме 407
Вопросы 408
Задачи 410
ГЛАВА 12. Апплеты в действии 411
12.1. От приложения к апплету 411
12.2. Апплеты в броузерах 416
12.3. Звук и изображения 430
12.4. Практикум 8. Проект по охране природы 436
Резюме 446
Вопросы 447
Задачи 449
ГЛАВА 13. Работа с несколькими подпроцессами 451
13.1. Работа с несколькими подпроцессами. Введение 451
13.2. Назначение подпроцессов 453
13.3. Синхронизация работы подпроцессов 469
13.4. Класс Display 474
13.5. Практикум 9. Использование в музее аудиоплееров 481
Резюме 487
Вопросы 488
Задачи 490

ГЛАВА 14. Работа в сети 492
14.1. Соединение через Internet 492
14.2. Порты и сокеты 497
14.3. Практикум 10. Система Chat 503
14.4. Подключение к базам данных 508
14.5. Доступ к удаленным объектам 517
14.6. Практикум 11. Информационная система аэропорта 521
Резюме 529
Вопросы 530
Задачи 531
ГЛАВА 15. Структуры данных и алгоритмы 534
15.1. О структурах данных 534
15.2. Линейный и двоичный поиск 536
15.3. Ускоренная сортировка Quicksort и эффективность работы 542
15.4. Стеки и очереди 547
15.5. И вновь связные списки 555
15.6. Битовые множества 561
Резюме 566
Вопросы 566
Задачи 568
ПРИЛОЖЕНИЕ А. Обозначения, используемые в схемах 570
ПРИЛОЖЕНИЕ Б. Перечень описаний конструкций языка Java 573
ПРИЛОЖЕНИЕ В. Список программ, апплетов и классов 576
ПРИЛОЖЕНИЕ Г. Ответы на вопросы для самоконтроля 579
Предметный указатель 580

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM

СПЕЦИАЛИСТАМ КНИЖНОГО БИЗНЕСА!

УВАЖАЕМЫЕ ГОСПОДА!
ИЗДАТЕЛЬСКИЙ ДОМ «ПИТЕР» ПРИГЛАШАЕТ ВАС К ВЗАИМОВЫГОДНОМУ СОТРУДНИЧЕСТВУ. МЫ ПРЕДЛАГАЕМ ЭКСКЛЮЗИВНЫЙ АССОРТИМЕНТ КОМПЬЮТЕРНОЙ, МЕДИЦИНСКОЙ, ПСИХОЛОГИЧЕСКОЙ, ЭКОНОМИЧЕСКОЙ И ПОПУЛЯРНОЙ ЛИТЕРАТУРЫ. МЫ ГОТОВЫ РАБОТАТЬ ДЛЯ ВАС НЕ ТОЛЬКО В САНКТ-ПЕТЕРБУРГЕ. ЗА ДОПОЛНИТЕЛЬНОЙ ИНФОРМАЦИЕЙ ОБРАЩАЙТЕСЬ ПО СЛЕДУЮЩИМ АДРЕСАМ:

Россия, г. Москва
Представительство издательства «Питер»,
м. «Калужская», ул. Бутлерова, д. 17б, оф. 207 и 240, тел./факс (095) 777-54-67.
E-mail: sales@piter.msk.ru

Россия, г. С.-Петербург
Представительство издательства «Питер»,
м. «Электросила», ул. Благодатная, д. 67,
тел. (812) 327-93-37, 387-54-65.
E-mail: sales@piter.com

Украина, г. Харьков
Представительство издательства «Питер»,
тел. (0572) 14-96-09, факс: (0572) 28-20-04, 28-20-05. Почтовый адрес: 61093, г. Харьков, а/я 9130. E-mail: piter@tender.kharkov.ua

Украина, г. Киев
Филиал Харьковского представительства издательства «Питер», тел./факс: (044) 490-35-68, 490-35-69. Адрес для писем: 04116, г. Киев-116, а/я 2. Фактический адрес: 04073, г. Киев, пр. Красных Казаков, д. 6, корп. 1.
E-mail: office@piter-press.kiev.ua

Беларусь, г. Минск
Представительство издательства «Питер»,
тел./факс (37517) 239-36-56. Почтовый адрес: 220100, г. Минск, ул. Куйбышева, 75.
ООО «Питер М», книжный магазин «Эврика».
E-mail: piterbel@tut.by

КАЖДОЕ ИЗ ЭТИХ ПРЕДСТАВИТЕЛЬСТВ РАБОТАЕТ С КЛИЕНТАМИ ПО ЕДИНОМУ СТАНДАРТУ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР».

Ищем зарубежных партнеров или посредников, имеющих выход на зарубежный рынок.
Телефон для связи: **(812) 327-93-37**.
E-mail: grigorjan@piter.com

Редакции компьютерной, психологической, экономической, юридической, медицинской, учебной и популярной (оздоровительной и психологической) литературы **Издательского дома «Питер»** приглашают к сотрудничеству авторов.
Обращайтесь по телефонам: **Санкт-Петербург — тел. (812) 327-13-11, Москва — тел.: (095) 234-38-15, 777-54-67.**

ИЗДАТЕЛЬСКИЙ ДОМ
ПИТЕР®
WWW.PITER.COM

**УВАЖАЕМЫЕ ГОСПОДА!
КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
ВЫ МОЖЕТЕ ПРИОБРЕСТИ
ОПТОМ И В РОЗНИЦУ
У НАШИХ РЕГИОНАЛЬНЫХ ПАРТНЕРОВ.**

Башкортостан
Уфа, «Азия», ул. Зенцова, д. 70 (оптовая продажа), маг. «Оазис», ул. Чернышевского, д. 88, тел./факс (3472) 50-39-00.
E-mail: asiaufa@ufanet.ru

Дальний Восток
Владивосток, «Приморский торговый дом книги», тел./факс (4232) 23-82-12. Почтовый адрес: 690091, Владивосток, ул. Светланская, д. 43.
E-mail: bookbase@mail.primorye.ru

Хабаровск, «Мирс», тел. (4212) 30-54-47, факс 22-73-30. Почтовый адрес: 680000, Хабаровск, ул. Ким-Ю-Чена, д. 21.
E-mail: postmaster@bookmirs.khv.ru

Хабаровск, «Книжный мир», тел. (4212) 32-85-51, факс 32-82-50. Почтовый адрес: 680000, Хабаровск, ул. Карла Маркса, д. 37.
E-mail: postmaster@worldbooks.knt.ru

Европейские регионы России
Архангельск, «Дом книги», тел. (8182) 65-41-34, факс 65-41-34. Почтовый адрес: 163061, пл. Ленина, д. 3. E-mail: book@atnet.ru

Калининград, «Вестер», тел./факс (0112) 21-56-28, 21-62-07. Почтовый адрес: 236040, Калининград, ул. Победы, д. 6. Магазин «Книги & книжечки».
E-mail: nshibkova@vester.ru;
http://www.vester.ru

Ростов-на-Дону, ПБОЮЛ Остроменский, пр. Соколова, д. 73, тел./факс (8632) 32-18-20.
E-mail: ostrom@don.sitek.net

Северный Кавказ
Ессентуки, «Россы», ул. Октябрьская, 424, тел./факс (87934) 6-93-09.
E-mail: rossy@kmw.ru

Сибирь
Иркутск, «ПродаЛитъ», тел. (3952) 59-13-70, факс 51-30-70. Почтовый адрес: 664031, Иркутск, ул. Байкальская, д. 172, а/я 1397.
E-mail: prodalit@irk.ru; http:/.www.prodalit.irk.ru

Иркутск, «Антей-книга», тел./факс (3952) 33-42-47. Почтовый адрес: 664003, Иркутск, ул. Карла Маркса, д. 20. E-mail: antey@irk.ru

Красноярск, «Книжный мир», тел./факс (3912) 27-39-71. Почтовый адрес: 660049, Красноярск, пр. Мира, д. 86. E-mail: book-world@public.krasnet.ru

Нижневартовск, «Дом книги», тел. (3466) 23-27-14, факс 23-59-50. Почтовый адрес: 628606, Нижневартовск, пр. Победы, д.12.
E-mail: book@nvartovsk.wsnet.ru

Новосибирск, «Топ-книга», тел. (3832) 36-10-26, факс 36-10-27. Почтовый адрес: 630117, Новосибирск, а/я 560.
E-mail: office@top-kniga.ru; http://www.top-kniga.ru

Тюмень, «Друг», тел./факс (3452) 21-34-39, 21-34-82. Почтовый адрес: 625019, ул. Республики, д. 211. E-mail: drug@tyumen.ru

Тюмень, «Фолиант», тел. (3452) 27-36-06, факс 27-36-11. Почтовый адрес: 625039, Тюмень, ул. Харьковская, д. 83а. E-mail: foliant@tyumen.ru

Татарстан
Казань, «Таис», тел. (8432) 72-34-55, факс 72-27-82. Почтовый адрес: 420073, Казань, ул. Гвардейская, д. 9а. E-mail: tais@bancorp.ru

Урал
Екатеринбург, магазин № 14, ул. Челюскинцев, д. 23, тел./факс (3432) 53-24-90.
E-mail: gvardia@mail.ur.ru

Екатеринбург, «Валео-книга», ул. Ключевская, д. 5, тел. (3432) 42-07-75, факс 42-56-00.
E-mail: valeo@etel.ru

ВСЕ цены на ВСЕ компьютерное

в еженедельнике

Atlant издательство

КомпьютерМаркет

БИЗНЕС

- ☑ ВЕСЬ КОМПЬЮТЕРНЫЙ ПЕТЕРБУРГ
- ☑ СТАТЬИ НА КОМПЬЮТЕРНУЮ ТЕМАТИКУ
- ☑ РЕКОМЕНДАЦИИ ПО ВЫБОРУ
- ☑ ON-LINE ВЕРСИЯ
- ☑ OFF-LINE ВЕРСИЯ



http://www.atlant.ru/comar
E-mail: comar@atlant.ru,
Тел./факс: (812) 326-96-68
Для писем: Россия, 195274,
Санкт-Петербург, а/я 24

Подписка по всем регионам России через каталог "Роспечать" (ЗАО Прессинформ)подписной индекс 83023

Джуди Бишоп

Эффективная работа

Java 2

Эта книга является примером удачной реализации известного принципа обучения — от простого к сложному. В ней вы найдете информацию по следующим темам:

- синтаксис языка
- общие принципы программирования
- библиотеки и вспомогательные программы
- структурное и объектно-ориентированное программирование
- разработка и реализация графического пользовательского интерфейса
- обработка событий
- реализация сетевых функций
- параллельная обработка данных

а также:

- вопросы и задачи
- образцы готовых приложений

Ваши друзья признают вас знатоком Java 2 еще до того, как вы дочитаете эту книгу до конца!

УРОВЕНЬ ПОЛЬЗОВАТЕЛЯ: **НАЧИНАЮЩИЙ/ОПЫТНЫЙ**

КАТЕГОРИЯ КНИГИ: **УЧЕБНОЕ ПОСОБИЕ**

Посетите наш web-магазин: **www.piter.com**



Книга, которую вы держите в руках, посвящена вопросам программирования на языке Java. Книга начинается с изложения общих принципов программирования и базовых сведений о языке Java 2, основных библиотеках и утилитах. Затем раскрываются такие важные темы, как структуры данных, разработка объектно-ориентированного программного обеспечения, структурное программирование, создание графического пользовательского интерфейса, организация обработки событий, реализация сетевых функций, а также одновременное использование в программе нескольких подпроцессов. Автор предлагает подборку протестированных образцов программ, в которых рассматриваются пути решения реальных задач.

ISBN 5-318-00741-4

