НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Основные классы стандартной библиотеки Java

Евгений Беркунский, НУК

eugeny.berkunsky@gmail.com

http://www.berkut.mk.ua

I/O Streams

- Byte Streams – ввод/вывод потоков двоичных данных
- Character Streams – ввод/вывод символьных данных, автоматическая обработка перевода в и из локальных кодировок.
- Buffered Streams – оптимизация ввода и вывода с помощью уменьшения количества обращений к нативным API.
- Scanning и Formatting – предоставляют программам возможности чтения и записи форматированного текста.
- Стандартные Streams и объект Console.
- Data Streams – двоичный ввод/вывод примитивных типов и строковых значений.
- Object Streams – двоичный ввод/вывод объектов.

Byte Streams

Программы используют Byte Streams для выполнения вводаи вывода 8-bit байтов. Все классы Byte Stream являются наследниками классов InputStream и OutputStream.

В стандартной библиотеке Java есть несколько классов Byte Stream.

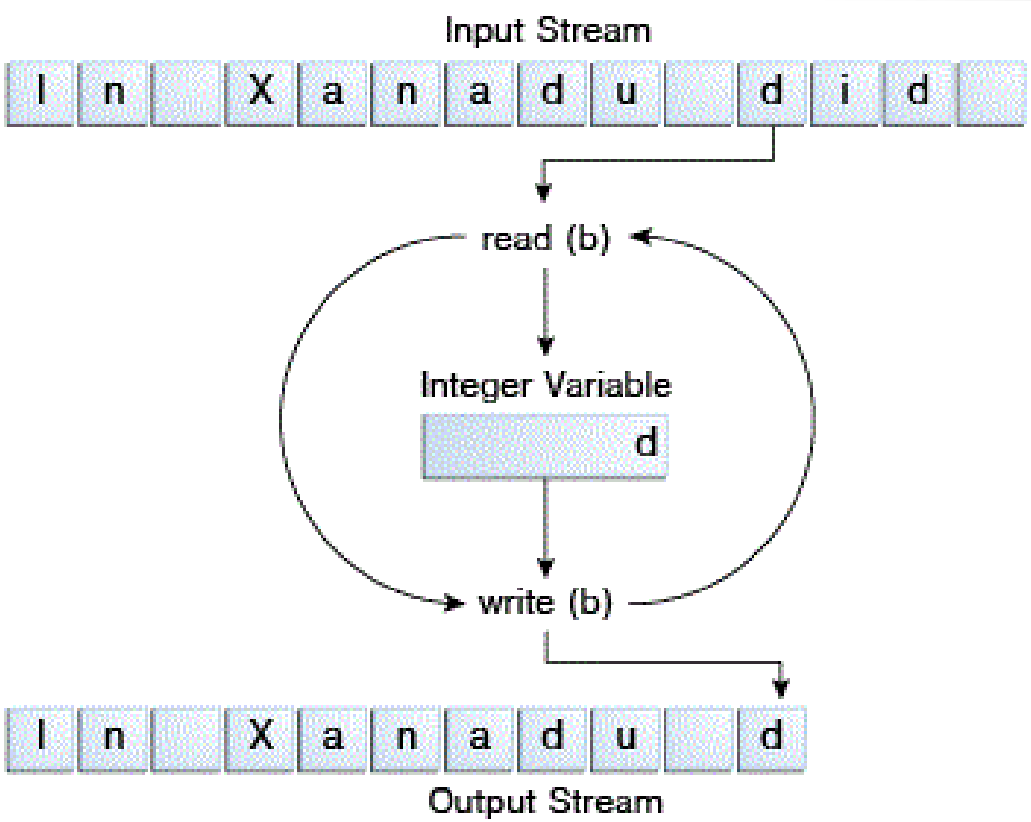
Рассмотрим, как работают Byte Streams на примере Byte Streams файлового ввода/вывода – FileInputStream и FileOutputStream.

Другие виды Byte Streams используются практически так же; они отличаются, в основном, только способом создания.

Byte Streams

```
import java.io.*;
public class CopyBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("xanadu.txt");
            out = new FileOutputStream("outagain.txt");
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

Byte Streams



Byte Streams

Потоки нужно закрывать

Закрытие потока, когда он более не нужен, очень важно — поэтому в примере используется блок finally для того, чтоб гарантировать, что оба потока будут закрыты даже, если возникнут ошибки. Такая практика поможет избежать серьезных утечек ресурсов.

Единственная возможная ошибка, если CopyBytes не сможет открыть один или оба файла. Когда это случится, переменная потока, соответствующая файлу не изменит своего начального значения null. Поэтому, CopyBytes проверяет что каждая переменная потока содержит ссылку на объект перед вызовом close().

Когда не нужно использовать Byte Streams

CopyBytes похожа на нормальную программу, но реально она использует способ низкоуровневого ввода/вывода, который следует избегать. Поскольку xanadu.txt содержит символьные данные, лучшим решением будет использование Character Streams, которые будут рассмотрены далее. Кроме того, есть потоки, предназначенные для более сложных типов данных.

Byte Streams следует использовать только для простейшего ввода/вывода.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Character Streams

Java-платформа сохраняет символьную информацию с использованием Unicode. Character Streams автоматически переводят этот внутренний формат в и из локальной кодировки. В западных локалях, местные кодировки, как правило, 8-битные и являются надмножеством ASCII.

Программа, которая использует символьные потоки вместо байтовых потоков сама адаптируется к местной кодировке и готова к интернационализации - все без дополнительных усилий со стороны программиста.

Если интернационализация не основной приоритет, вы можете просто использовать классы потоков символов, не обращая особого внимания на вопросы кодировок.
В дальнейшем, если интернационализация становится приоритетом, ваша программа может быть адаптирована без большого перекодирования.

Character Streams

```
import java.io.*;
public class CopyCharacters {
    public static void main(String[] args) throws IOException {
        FileReader inputStream = null;
        FileWriter outputStream = null;
        try {
            inputStream = new FileReader("xanadu.txt");
            outputStream = new FileWriter("characteroutput.txt");
            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Character Streams

InputStreamReader и OutputStreamWriter - два класса общего назначения, предназначенные для превращения байтовых потоков в символьные и наоборот.

Их стоит использовать, когда стандартных расфасованный классы потоков символов, которые соответствуют вашим потребностям. (Например, для Socket)

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Строковый ввод/вывод

Символьный ввод/вывод обычно включает большие единицы информации, чем единичные символы.
Обычно, единицей информации является строка: последовательность символов, заканчивающаяся признаком конца. Признаком конца строки может быть последовательность возврат-каретки/перевод-строки ("\r\n"), просто перевод-каретки ("\r"), или просто перевод-строки ("\n").
Поддержка всех возможных признаков окончания строки позволяет программам читать текстовые файлы, созданные в любой из известных ОС.

Строковый ввод/вывод

```
import java.io.*;
public class CopyLines {
    public static void main(String[] args) throws IOException {
        BufferedReader inputStream = null;
        PrintWriter outputStream = null;
        try {
            inputStream = new BufferedReader(
                              new FileReader("xanadu.txt"));
            outputStream = new PrintWriter(
                              new FileWriter("output.txt"));
            String line;
            while ((line = inputStream.readLine()) != null) {
                outputStream.println(l);
            }
        } finally {
            if (inputStream != null) { inputStream.close(); }
            if (outputStream != null) { outputStream.close(); }
        }
    }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Сканирование и форматирование

Объекты типа Scanner удобны для разбивки форматированного ввода на слова и преобразования отдельных токенов (слов) согласно их типам данных.
По умолчанию, scanner использует пробелы для разделения токенов.
Пробелы – это символы пробелов, табуляций и перевода строки.
Полный список можно узнать в документации для метода Character.isWhitespace()

```
import java.io.*;
import java.util.Scanner;
public class ScanXan {
    public static void main(String[] args) throws IOException {
        Scanner s = null;
        try {
            s = new Scanner(new BufferedReader(
                    new FileReader("xanadu.txt")));
            while (s.hasNext()) { System.out.println(s.next()); }
        } finally {
            if (s != null) {
                s.close();
            }
        }
    }
}
```

Сканирование и форматирование

```
import java.io.*;
import java.util.*;
public class ScanSum {
    public static void main(String[] args) throws IOException {
        Scanner s = null;
        double sum = 0;
        try {
            s = new Scanner(new BufferedReader(
                                new FileReader("numbers.txt")));
                while (s.hasNext()) {
                if (s.hasNextDouble()) {
                    sum += s.nextDouble();
                } else {
                    s.next();
                }
            }
        } finally {
            s.close();
        }
        System.out.println(sum);
    }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Сканирование и форматирование

Объекты потоков, реализующие форматирование – это экземпляры классов PrintWriter (класс – символьный поток), или PrintStream (класс – поток байт).

Как байтовые и символьные потоки, объекты классов PrintStream и PrintWriter реализуют стандартный набор методов записи для простого байтового или символьного вывода.
Кроме того, PrintStream и PrintWriter реализуют одинаковый набор методов для преобразования внутренних данных в форматированный вывод.
Реализовано два уровня форматирования:

- **print** и **println** форматируют отдельные значения стандартным способом.
- **format** форматирует практически любое количество значений, основываясь на строке форматирования, в которой может быть указано множество опций для точного форматирования.

*Примечание. Метод **printf** – удобное название для копии метода **format***

Демонстрация

Ввод/вывод в командной строке

Программа часто запускается из командной строки и взаимодействует с пользователем в среде командной строки.
Платформа Java поддерживает этот вид взаимодействия в двух направлениях: через стандартные потоки и через консоль.

Платформа Java поддерживает три стандартных потока:

- Стандартный ввод - System.in
- Стандартный вывод - System.out
- Стандартный вывод ошибок - System.err

Ввод/вывод в командной строке

Другая (более продвинутая) альтернатива стандартным потокам – это объект Console. Существует один предопределенный объект Console. Console удобна, например, в использовании для безопасного ввода паролей. Кроме того, объект Console позволяет вводить и выводить данные с помощью своих методов.

Прежде, чем программа сможет использовать Console, она должна попытаться получить объект Console с помощью вызова System.console().

- Если объект Console доступен, этот метод его вернет.
- Если System.console() вернет NULL, значит консольные операции не поддерживаются.

Объект Console позволяет ввести пароль с помощью метода readPassword

- Во-первых, он подавляет эхо (вводимые символы не показываются).
- Во-вторых, он возвращает символьный массив, а не String, поэтому пароль можно перезаписать, при этом он сотрется из памяти.

Демонстрация

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Data Streams

Data Streams поддерживают двоичный ввод/вывод значений примитивных данных (boolean, char, byte, short, int, long, float, и double) а также String.
Все Data Streams реализуют интерфейс DataInput или DataOutput.

Type	Methods in the *DataInput* Interface	Methods in the *DataOutput* interface
boolean	readBoolean()	writeBoolean(boolean b)
char	readChar()	writeChar(int c)
byte	readByte()	writeByte(int b)
short	readShort()	writeShort(int s)
int	readInt()	writeInt(int i)
long	readLong()	writeLong(long l)
float	readFloat()	writeFloat(float f)
double	readDouble()	writeDouble(double d)
String	readLine()	writeChars(String str)
String	readUTF()	writeUTF(String str)

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Object Streams

Классы ObjectInputStream и ObjectOutputStream предназначены для ввода/вывода объектов.

- Эти классы реализуют интерфейсы ObjectInput и ObjectOutput, которые являются наследниками DataInput и DataOutput.
- Все примитивные операции ввода/вывода, реализованные в Data Streams также реализованы в Object Streams.
- Object Stream может содержать смесь примитивных и объектных значений.

Object Streams

Что будет, если ссылку на один объект дважды записать в один поток вывода?

```
Object ob = new Object();
out.writeObject(ob);
out.writeObject(ob);
```

```
Object ob1 = in.readObject();
Object ob2 = in.readObject();
```

В результате – две переменных ob1 и ob2 будут содержать ссылку на один объект.

Что будет, если ссылку на один объект дважды записать в разные потоки вывода?

Демонстрация

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Некоторые классы пакета java.util

Класс	Описание
Arrays	Класс для выполнения базовых операций с массивами
Calendar	Абстрактный класс. Его наследники представляют различные календари
Calendar.Builder	Используется для создания календаря с соответствующими настройками
Currency	Тип данных для представления денег.
Date	Дата – устаревший, но поддерживаемый объект даты
Formatter	Форматирование данных
GregorianCalendar	Григорианский календарь. Наследник класса Calendar
Locale	Локаль – работа с локальными языками и стандартами
Objects	Содержит статические утилитные методы для работы с объектами
Random	Генератор случайных чисел
Scanner	Сканнер (был рассмотрен ранее)
StringTokenizer	Разделяет строки на токены

Демонстрация

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Некоторые классы пакета java.time

Класс	Описание
Clock	Объект, предоставляющий доступ к дате и времени используя часовые пояса
Duration	Длительность «time-based» промежутка времени, например ‘34,5с’
Instant	Мгновение – ‘точка’ на шкале времени
LocalDate	Дата без учета часового пояса, вроде 2015-07-18
LocalDateTime	Дата-время без учета часового пояса, вроде 2015-07-18T10:30:00
LocalTime	Время без учета часового пояса, вроде 10:30:00
MonthDay	Месяц-день в текущей календарной системе, вроде --10-17
OffsetDateTime	Дата-время со смещением относительно UTC/Greenwich в календарной системе ISO-8601, например 2015-07-18T10:15:30+01:00.
OffsetTime	Время со смещением относительно UTC/Greenwich в календарной системе ISO-8601, например 10:15:30+01:00.
Period	Длительность «date-based» промежутка времени, например '2 года, 3 месяца и 4 дня'.
Year	Год в календарной системе ISO-8601, например 2015
YearMonth	Год-месяц в календарной системе ISO-8601, например 2015-07
ZonedDateTime	Дата-время с учетом часового пояса, например 2015-12-03T10:15:30+02:00 Europe/Kiev
ZoneId	A time-zone ID, например Europe/Kiev.
ZoneOffset	Смещение часового пояса относительно Greenwich/UTC, например as +02:00.

Clock

Clock предоставляет доступ к текущей дате и времени.

- Этот тип знает о часовых поясах и может использоваться вместо вызова System.currentTimeMillis() для возвращения миллисекунд.
- Такая точная дата также может быть представлена классом Instant.
- Объекты этого класса могут быть использованы для создания объектов устаревшего типа java.util.Date.

```
Clock clock = Clock.systemDefaultZone();
long millis = clock.millis();

Instant instant = clock.instant();
Date legacyDate = Date.from(instant);  // legacy java.util.Date
```

Часовые пояса

- Часовые пояса представлены типом ZoneId.
- Доступ к ним можно получить при помощи статических фабричных методов.
- Часовые пояса содержат смещения, которые важны для конвертации дат и времени в местные.

```
System.out.println(ZoneId.getAvailableZoneIds());
// prints all available timezone ids

ZoneId zone1 = ZoneId.of("Europe/Kiev");
ZoneId zone2 = ZoneId.of("Brazil/East");
System.out.println(zone1.getRules());
System.out.println(zone2.getRules());

// ZoneRules[currentStandardOffset=+02:00]
// ZoneRules[currentStandardOffset=-03:00]
```

Демонстрация

LocalTime

- Тип LocalTime представляет собой время с учетом часового пояса, например, 10pm или 17:30:15.
- В следующем примере создаются два местных времени для часовых поясов, определенных выше.
- Затем оба времени сравниваются, и вычисляется разница между ними в часах и минутах.

```
LocalTime now1 = LocalTime.now(zone1);
LocalTime now2 = LocalTime.now(zone2);

System.out.println(now1.isBefore(now2));  // false

long hoursBetween = ChronoUnit.HOURS.between(now1, now2);
long minutesBetween = ChronoUnit.MINUTES.between(now1, now2);

System.out.println(hoursBetween);       // -3
System.out.println(minutesBetween);     // -239
```

LocalTime

- Тип LocalTime содержит различные фабричные методы, которые упрощают создание новых экземпляров, а также парсинг строк.

```
LocalTime late = LocalTime.of(23, 59, 59);
System.out.println(late);       // 23:59:59

DateTimeFormatter uaFormatter =
        DateTimeFormatter
                .ofLocalizedTime(FormatStyle.SHORT)
                .withLocale(new Locale("uk","UA"));

LocalTime leetTime = LocalTime.parse("13:37", uaFormatter);
System.out.println(leetTime);   // 13:37
```

Демонстрация

LocalDate

- Тип LocalDate представляет конкретную дату, например, 2014-03-18.
- Объекты LocalDate неизменяемы и являются аналогом LocalTime.
- Пример демонстрирует вычисление новой даты путем сложения или вычитания дней, месяцев или годов.
- Помните, что каждая операция возвращает новый экземпляр.

```
LocalDate today = LocalDate.now();
LocalDate tomorrow = today.plus(1, ChronoUnit.DAYS);
LocalDate yesterday = tomorrow.minusDays(2);

LocalDate independenceDay = LocalDate.of(2015,Month.AUGUST,24);
DayOfWeek dayOfWeek = independenceDay.getDayOfWeek();
System.out.println(dayOfWeek);    // MONDAY
```

LocalDate

- Создание экземпляра LocalDate путем парсинга строки:

```
 DateTimeFormatter uaFormatter =
         DateTimeFormatter
                 .ofLocalizedDate(FormatStyle.LONG)
                 .withLocale(new Locale("ru", "UA"));

LocalDate d = LocalDate.parse("17 июля 2015 г.",uaFormatter1));
System.out.println(d);
```

Демонстрация

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

LocalDateTime

- Тип LocalDateTime представляет собой комбинацию даты и времени.
- Объекты LocalDateTime неизменяемы и работают аналогично LocalTime и LocalDate. Мы
- Можно использовать различные методы для извлечения конкретных значений из даты-времени:

```
LocalDateTime sylvester = LocalDateTime.of(
        2015, Month.DECEMBER, 31, 23, 59, 59);

DayOfWeek dayOfWeek = sylvester.getDayOfWeek();
System.out.println(dayOfWeek);      // THURSDAY

Month month = sylvester.getMonth();
System.out.println(month);          // DECEMBER

long minuteOfDay = sylvester.getLong(ChronoField.MINUTE_OF_DAY);
System.out.println(minuteOfDay);    // 1439
```

Instant

Путем добавления информации о часовом поясе мы можем получить Instant.

```
Instant instant = sylvester
        .atZone(ZoneId.systemDefault())
        .toInstant();

Date legacyDate = Date.from(instant);
System.out.println(legacyDate);  // Wed Dec 31 23:59:59 CET 2014
```

Демонстрация

Форматирование даты-времени

Форматирование даты-времени работает так же, как и форматирование даты или времени.
Можно использовать библиотечные или свои собственные шаблоны.

```
DateTimeFormatter formatter =
    DateTimeFormatter
        .ofPattern("MMM dd, yyyy - HH:mm");

LocalDateTime parsed = LocalDateTime.parse(
        "Nov 03, 2014 - 07:13", formatter);
String string = formatter.format(parsed);
System.out.println(string);     // Nov 03, 2014 - 07:13
```

Демонстрация

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Основные классы стандартной библиотеки Java

Евгений Беркунский, НУК

eugeny.berkunsky@gmail.com

http://www.berkut.mk.ua