МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

Є. Ю. БЕРКУНСЬКИЙ

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з дисципліни "Мова програмування Java"

Рекомендовано Методичною радою НУК

Електронне видання
комбінованого використання на DVD-ROM

МИКОЛАЇВ • НУК • 2015

УДК 004.43(076)
ББК 32.973-01я73
Б 48

Укладач Є. Ю. Беркунський, старш. викладач

Рецензент К. В. Кошкін, д-р техн. наук, професор

Беркунський Є. Ю.

Б 48 Методичні вказівки до виконання лабораторних робіт з дисципліни "Мова програмування Java" / Є. Ю. Беркунський. – Миколаїв : НУК, 2015. – 62 с.

Наведено теоретичні відомості та довідкову інформацію для виконання завдань лабораторних робіт з дисципліни: основні відомості про структуру програм, стандартні класи, управляючі структури мови Java та відповіді на основні питання, пов'язані з організацією та методикою виконання лабораторних робіт.

Призначено для студентів денної і заочної форм навчання спеціальності 8.05010101 "Інформаційні управляючі системи та технології". Також можуть бути використані студентами інших спеціальностей.

УДК 004.43(076)
ББК 32.973-01я73

Навчальне видання

БЕРКУНСЬКИЙ Євген Юрійович

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з дисципліни
"Мова програмування Java"

Комп'ютерне верстання *А. Й. Лихіна*
Коректор *М. О. Паненко*

Формат 60×84/16. Ум. друк. арк. 3,6. Обсяг даних 8353 кб. Тираж 15 прим.
Вид. № 35. Зам. № 67.

Видавець і виготівник Національний університет кораблебудування імені адмірала Макарова
просп. Героїв Сталінграда, 9, м. Миколаїв, 54025, e-mail : publishing@nuos.edu.ua

Свідоцтво суб'єкта видавничої справи ДК № 2506 від 25.05.2006 р.

ВСТУП

Мова програмування Java почала своє існування у 1995 році, коли компанія Sun Microsystems запропонувала використовувати її для створення програмних продуктів, що будуть працювати у найрізноманітніших пристроях. Основними властивостями мови стали: кросплатформовість, об'єктна орієнтованість та жорстка стандартизація компіляторів.

За час, що минув, мова Java набула можливостей використання у мобільних телефонах, стаціонарних комп'ютерах, серверах, та навіть у побутовій техніці. За допомогою Java розробляються програми, що використовуються у різних галузях людської діяльності – від ігрових продуктів до банківської сфери та наукових досліджень. Навіть космічні апарати, що досліджують інші планети, використовують програмні модулі, написані мовою Java.

Сучасна Java містить стандартну бібліотеку, що складається з великої кількості (декілька тисяч) класів, інтерфейсів та перелічень. Вони дозволяють вирішувати майже всі задачі, що стоять перед розробниками сучасних програмних продуктів.

Дисципліна "Мова програмування Java" призначена для початкового знайомства з цією, достатньо складною технологією. Можливості мови Java дозволяють використовувати її при вивченні таких дисциплін, як "Комп'ютерна графіка", "Web-технології" та інших.

Лабораторна робота № 1

Програмування лінійних алгоритмів
Стандартні класи і їхні методи у мові Java

1. Створити клас, що має методи для обчислення на ЕОМ значень змінних, що зазначені у таблиці, за даними розрахунковими формулами і наборами вхідних даних.

2. Доповнити клас методом, що виводить на екран значення вхідних даних і результати обчислень, супроводжуючи вивід найменуваннями виведених змінних.

3. Додати в клас метод, що друкує поточну дату і час у вказаному форматі.

4. Доповнити клас методом введення початкових значень.

5. Створити метод, що вводить дані, обчислює потрібні значення за вказаними формулами, та друкує потрібні результати.

6. Доповнити клас методом main, що є необхідним для використання класу, як автономної програми, та виконати цю програму.

Короткі теоретичні відомості

Огляд структури Java-програми

Всі Java-програми містять в собі 4 основні різновиди будівельних блоків: класи (classes), методи (methods), змінні (variables) і пакети (packages). На якій би мові Ви не програмували раніше, Ви скоріш за все вже добре знайомі з методами, які є не що інше, ніж функції чи підпрограми, та зі змінними, в яких зберігаються дані. З іншого боку, класи представляють собою фундамент об'єктно-орієнтованих властивостей мови. Поки що, для простоти, можна вважати клас деяким цілим, що містить у собі змінні и методи. Нарешті, пакети містять в собі класи і допомагають компілятору знайти ті класи, що потрібні йому для компіляції прикладної програми.

Java-програма може містити в собі будь-яку кількість класів, але один з них завжди має особливий статус, і безпосередньо взаємодіє з оболонкою часу виконання. Цей клас називають первинним класом (primary class).

Коли програма запускається з командного рядка, системі потрібен тільки один спеціальний метод, що повинен бути присутнім у первинному класі, – метод main. Коли ми будемо розглядати програмування аплетів, ми побачимо, що первинний клас аплета повинен містити вже

декілька таких спеціальних методів. Розглянемо приклад програми мовою Java:

```
import java.util.Date;
// імпортування класу Date зі стандартного пакету java.util

public class OurPrimaryClass {
    public static void main(String[] S) {
        System.out.println("Hello, Java!");
        Date d = new Date();
              System.out.println(
                       "Date: "+d.toString());
    }
}
```

Наведена програма виводить на екран повідомлення "Hello, Java!" та поточну системну дату.

Стандартні типи даних Java

Всі змінні та вирази у мові програмування Java можуть бути віднесені до однієї з двох великих груп типів: примітивних типів (primitive types), або посилальних типів (reference types), що містять у собі типи, визначені користувачем, і типи масивів. До примітивних типів відносяться стандартні, вбудовані в мову типи для представлення чисельних значень, одиночних символів і логічних значень. Навпаки, усі посилальні типи є динамічними типами. Головні розбіжності між двома згаданими групами типів перелічені у наступній таблиці:

Таблиця 1.1. **Порівняння примітивних і посилальних типів**

Характеристика	Примітивні типи	Посилальні типи
Чи визначені в самій мові Java?	Так	Ні
Чи мають визначений розмір?	Так	Ні
Чи повинна для змінних цих типів виділятися пам'ять під час роботи програми?	Ні	Так

На практиці найважливішим розходженням між примітивними і посилальними типами є те, про що свідчить останній рядок цієї таблиці, а саме – що пам'ять для змінних посилального типу повинна виділятися під час виконання програми. Використовуючи змінні посилальних типів, ми повинні явно вимагати необхідну кількість пам'яті для кожної змінної

перш, ніж ми зможемо зберегти в цієї змінний деяке значення. Причина цього проста: оболонка часу виконання сама по собі не знає, яка кількість пам'яті потрібна для того чи іншого посилального типу.

Усього в мові Java визначено вісім примітивних типів, що перелічені в таблиці 1.2.

Таблиця 1.2. **Примітивні типи мови Java**

Тип	Розмір	Діапазон	Приклад
`byte`	1 байт	від –128 до 127	125
`short`	2 байти	від –32768 до 32767	–23
`int`	4 байти	від –2147483648 до 2147483647	2002300
`long`	8 байт	від –922372036854775808 до 922372036854775807	1243565
`float`	4 байти	Залежить від розрядності числа	1.2f
`double`	8 байт	Залежить від розрядності числа	123.4
`boolean`		`false, true`	`true`
`char`	2 байти	Усі символи стандарту Unicode	‘z’

Стандартні математичні функції

Оскільки мова Java є об'єктно-орієнтованою, то математичні функції повинні належати до деякого класу. Фактично існують два класи, що визначають математичні операції: Math та StrictMath, Останній призначений для виконання обчислень із "підвищеною точністю", але через поширення вбудованих у процесори математичних модулів, "звичайна" і "підвищена" точність у сучасній Java не розрізняються. Тому найчастіше використовується саме клас Math.

Усі стандартні математичні функції в мові Java є статичними методами класу Math, який визначений з модифікатором **final**, тобто не припускає спадкування. Крім того, клас Math має декілька визначених констант, наведемо дві з них:

Таблиця 1.3. **Основні константи класу Math**

Константа	Значення
Math.PI	Число π = 3.14159...
Math.E	Число e = 2.71828...

Основні статичні методи класу Math наведені у наступній таблицю 1.4.

Таблиця 1.4. **Основні методи класу Math**

Функція – метод	**Пояснення**
`Math.abs(x)`	Модуль числа x
`Math.acos(x)`	Арккосинус x
`Math.asin(x)`	Арксинус x
`Math.atan(x)`	Арктангенс x
`Math.cbrt(x)`	Кубічний корінь з x
`Math.ceil(x)`	Найближче число до x, що не містить дробової частини і більше за x
`Math.cos(x)`	Косинус x
`Math.exp(x)`	Експонента від x
`Math.floor(x)`	Найближче число до x, що не містить дробової частини і менше за x
`Math.hypot(x,y)`	Гіпотенуза прямокутного трикутника зі сторонами x, y
`Math.log(x)`	Натуральний логарифм x
`Math.max(x,y)`	Більше з двох чисел
`Math.min(x,y)`	Менше з двох чисел
`Math.pow(x,y)`	X в степені Y
`Math.random()`	Випадкове число з проміжку [0;1)
`Math.rint(x)`	Найближче число до x, що не містить дробової частини
`Math.round(x)`	Найближче до x ціле число
`Math.sin(x)`	Синус x
`Math.sqrt(x)`	Квадратний корінь з x
`Math.tan(x)`	Тангенс x
`Math.toDegrees(x)`	Перевеження кута з радіанів у градуси
`Math.toRadians(x)`	Переведення кута з градусів у радіани

Примітка. Починаючи з версії j2sdk 5.0 (30.09.2004) у мові Java з'явилась можливість імпорту статичних змінних та методів класу за допомогою директиви **import static** на початку програми. Наприклад:

```
import static java.lang.Math.*;
// імпортування статичних змінних і методів класу Math

public class OurPrimaryClass {
    public static void main(String[] S) {
        double x;
        x = sin(PI/6);
           // без статичного імпорту треба писати
x=Math.sin(Math.PI/6);
        System.out.println(x);
    }
}
```

Клас Date

Для роботи з датами і часом у стандартній бібліотеці мови програмування Java є декілька класів. Одним з них є клас Date. Цей клас знаходиться у пакеті java.util. Тому, для роботи з ним, на початку програми треба додати директиву **import java.util.Date**; Приклад його використання наведений на початку теоретичних відомостей до цієї лабораторної роботи.

Виведення даних у консолі Java-програм

Для виведення інформації на консоль використовуються методи стандартного класу PrintStream:

– print
– println
– printf
– format (точна копія printf)

Кожна програма на мові Java містить стандартний об'єкт типу PrintStream – System.out. Таким чином, виведення інформації на екран буде записуватися як System.out.print(…), System.out.println(…), або System.out.printf(…).

Методи print та println повинні завжди мати один параметр – вираз будь-якого типу, що може бути автоматично приведений до рядкового типу.

Наприклад,

```
System.out.println("2+2="+(2+2)); // буде виведено 2+2=4
System.out.println("Значення суми="+s);
// буде виведено Значення суми=xxx , де xxx – значення змінної S
```

Методи **printf** та **format** можуть мати список параметрів, що розділяються комами. Перший параметр – рядок, що містить текст для виведення і форматні шаблони для виведення значень інших параметрів.

Наприклад, якщо a=2, b=3

```
System.out.printf("Значення %d + %d = %d", a, b, a+b);
// буде виведено Значення 2 + 3 = 5
```

Форматні шаблони для виведення звичайних, символьних та числових типів мають наступний синтаксис:

```
%[індекс_аргумента$][опції][ширина][.точність]-перетворення
```

Необов'язковий параметр індекс_аргумента є цілим числом, що вказує позицію в списку аргументів. Посилання на перший аргумент буде записане як "1$", на другий – "2$", і т.д.

Необов'язковий параметр опції – це набір символів, що змінюють формат виведення. Набір припустимих опцій залежить від типу перетворення.

Необов'язковий параметр ширина – це невід'ємне ціле число, що показує мінімальну кількість символів, що їх треба вивести.

Необов'язковий параметр точність – це невід'ємне ціле число, що зазвичай використовується для обмеження кількості символів, що будуть виведені. Його дія залежить від параметру перетворення.

Обов'язковий параметр перетворення – це один символ, що вказує як аргумент буде відформатований. Набір припустимих перетворень для вказаного аргументу залежить від типу даних аргументу.

Форматні шаблони для виведення типів, що означають дату і час мають такий синтаксис:

```
%[індекс_аргумента$][опції][ширина]перетворення
```

Індекс_аргумента, опції, ширина – описані вище, а перетворення – два символи, де перший – 't', або 'T', а другий – описує тип перетворення. Інформацію про всі типи перетворень можна знайти на сайті Oracle, у розділі присвяченому мові програмування Java[1]. У таблиці 1.5 наведено основні з них.

Для виведення даті/часу недостатньо вказати лише сивол форматування "t". Додатково треба вказати, яка саме частина дати/часу буде виводитись (день, місяць, рік, година, хвилина тощо). Для цього використовуються додаткові (уточнюючі) символи перетворень. Основні з них показані у таблиці 1.6.

Приклади використання System.out.printf для виведення на екран

`System.out.printf(”Hello, World!”);`		`Hello, World!`

`System.out.printf(”Hello, World!%n”);`
або
`System.out.printf("Hello, World!\n");`
		`Hello, World!`

`System.out.printf("Sum %d + %d = %d", a,b,a+b);`		`Sum 15 + 2 = 17`
`System.out.printf( "Const of Pi = %5.2f",Math.PI);`		`Const of Pi = 3,14`

`Date d – new Date();`
`System.out.printf(`
`"Сьогодні %1$te %1$tB %1$tY\n", d);`
		`Сьогодні 8 вересня 2015`

Таблиця 1.5. **Основні типи – символи перетворень**

Перетворення	Категорія	Описання
`'b', 'B'`	boolean	Якщо аргумент *arg* є `null`, тоді результатом буде `"false"`. Якщо *arg* належить до типу `boolean` або **`Boolean`**, то результатом буде рядок – "true" або "false" в залежності від значення *arg*. У всіх інших випадках результатом буде "true".
`'s', 'S'`	general	Якщо аргумент *arg* є `null`, тоді результатом буде `"null"`. Якщо *arg* має метод **`formatTo`**, то він буде викликаний. Інакше, результат буде отриманий через виклик `arg.toString()`.
`'c', 'C'`	character	Результатом буде символ Unicode
`'d'`	integral	Результат буде відформатований, як ціле десяткове число
`'e', 'E'`	floating point	Результат буде відформатований, як число з плаваючою точкою у "науковому" форматі
`'f'`	floating point	Результат буде відформатований, як десяткове число
`'g', 'G'`	floating point	Результат буде відформатований, як число у "науковому" форматі, залежно від точності та значення після округлення.
`'t', 'T'`	date/time	Префікс для символу перетворень дати і часу.
`'%'`	percent	Результатом буде символ `'%'` (`'\u0025'`)
`'n'`	line separator	Результатом буде символ, що відокремлює рядки в залежності від платформи.

Таблиця 1.6. **Основні символи перетворення для дати і часу**

`'H'`	Година поточного дня, з нулем попереду, якщо потрібно, тобто `00 - 23`.
`'k'`	Година поточного дня, без нуля попереду, тобто `0 - 23`.
`'M'`	Хвилина поточної години, з нулем попереду, якщо потрібно, тобто `00 - 59`.

Продовж. таблиця 1.6

'S'	Секунда поточної хвилини, з нулем попереду, якщо потрібно, тобто 00 – 60
'L'	Мілісекунди поточної секунди, з нулями попереду, якщо потрібно, тобто 000 – 999.
'N'	Наносекунди поточної секунди, тобто 000000000 – 999999999.
'B'	Повна назва місяця, відповідно мовних налаштувань, наприклад "вересня", "січня".
'b'	Скорочена назва місяця, відповідно мовних налаштувань, наприклад "вер", "січ".
'A'	Повна назва дня тижня, відповідно мовних налаштувань, наприклад "неділя", "понеділок"
'a'	Скорочена назва дня тижня, відповідно мовних налаштувань, наприклад "нд", "пн"
'C'	Чотиризначне число року поділене на 100, як два знаки з нулем попереду, якщо потрібно, тобто 00 – 99
'Y'	Рік, у вигляді чотиризначного числа
'y'	Останні дві цифри року, з нулем попереду, якщо потрібно, тобто 00 – 99.
'j'	День року, відформатований як три знаки, з нулями попереду, якщо потрібно, тобто 001 – 366 для Григоріанського календаря.
'm'	Місяць, відформатований як два знаки, з нулем попереду, якщо потрібно, тобто 01 – 13.
'd'	День місяця, у форматі двох знаків, з нулем попереду, якщо потрібно, тобто 01 – 31.
'e'	День місяця, у форматі двох знаків, тобто 1 – 31.
'R'	Час у форматі 24-годин як "%tH:%tM"
'T'	Час у форматі 24-годин як "%tH:%tM:%tS".
'D'	Дата у форматі як "%tm/%td/%ty".
'c'	Дата і час форматовані як "%ta %tb %td %tT %tZ %tY", наприклад "нд вер 06 12:13:21 EEST 2015".

Введення даних з консолі

Для введення даних у мові програмування java можна скористатися різними засобами. Один з них використовує спеціальний об'єкт, що належить до класу Scanner. Цей клас містить методи для введення найрізноманітніших типів даних. Приклад його використання наведений нижче:

```
import java.io.*;
import java.util.*;

public class InOutExample {
    public static void main(String[] s) {
        Scanner s = new Scanner(System.in);
        // Читання цілого числа з рядка
        int i = s.nextInt();
        // Читання дійсного числа з рядку
        double x = s.nextDouble();
    ..................................
     }
}
```

Створення і виконання Java-програм у середовищі NetBeans

1. Створіть новий проект, для цього:

2. Після запуску NetBeans у головному меню програми оберіть **File -> New Project…**

3. У вікні, що відкриється, оберіть категорію **Java** та вид проекту **Java Application**, та натисніть кнопку **Next**

4. У наступному вікні введіть ім'я проекту (Project Name). Ім'я проекту оберіть так, щоб було зрозуміло його призначення (наприклад **First**).

5. Оберіть місце розміщення файлів проекту (**Project Location**) та ім'я головного класу проекту (ім'я може містити ім'я пакету), наприклад, **first.Main**. Залиште відмітку у обох CheckBox'ах.

6. Натисніть кнопку **Finish**.

7. Впишіть код вашої програми у вікно редактора коду NetBeans.

8. Для запуску програми натисніть кнопку "Run" (на ній зображено зелений трикутник).

Примітка 1. *Інші інструменти середовища NetBeans будуть розглянуті у наступних лабораторних роботах.*

Примітка 2. *Для запуску автономної програми на мові Java можна перейти в каталог, де розміщено скомпільований код програми – файли з розширенням class* (*у нашому випадку каталог classes проекту*) *та виконати таку команду* (*у режимі командного рядка*):

```
java <ім'я_класу_без_розширення>
```

Наприклад, **java first.Main**

Приклад програми, *що створена у середовищі NetBeans*

```
package first;
public class Main {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Main prog = new Main();
        prog.run();
    }

    private int calcSquare(int x) {
        return x*x;
    }

    private void print(int x, int y) {
        System.out.println("x="+x);
        System.out.println("x^2="+y);
    }

    private void run() {
        int x = 5;
        int y = calcSquare(x);

        print(x,y);
    }

}
```

Варіанти завдань

Варіант	*Розрахункові формули*	*Значення вхідних даних*	*Формат дати і часу*
1	$R = x^2(x+1)/b - \sin^2(x+a);\ \ s = \sqrt{\frac{xb}{a}} + \cos^2(x+b)^3$	a=0.7 b=0.05 x=0.5	Дата у форматі рр-мм-дд
2	$f = \sqrt[3]{m\mathrm{tg}\,t + \lvert c\sin t\rvert};\ \ z = m\cos(bt\sin t) + c$	m=2;c=-1 t=1.2 b=0.7	Дата і час з точністю до мілі-секунд
3	$y = b\mathrm{tg}^2 x - \frac{a}{\sin^2(x/a)};\ \ d = ae^{-\sqrt{a}}\cos(bx/a)$	a=3.2 b=17.5 x=-4.8	Місяць, день рік та день тижня
4	$s = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24};\ \ f = x(\sin x^3 + \cos^2 y)$	x=0.335 y=0.025	Час у форматі гг:хх:сс
5	$s = x^3\mathrm{tg}^2(x+b)^2 + \frac{a}{\sqrt{x+b}};\ \ Q = \frac{bx^2 - a}{e^{ax} - 1}$	a=16.5 b=3.4 x=0.61	Дата у форматі дд міс рррр
6	$y = e^{-bt}\sin(at+b) - \sqrt{\lvert bt + a\rvert};\ \ s = b\sin(at^2\cos 2t) - 1$	a=-0.5 b=1.7 t=0.44	Дата у форматі дд місяць рррр
7	$y = \sin^3(x^2 + a)^2 - \sqrt{\frac{x}{b}};\ \ z = \frac{x^2}{a} + \cos(x+b)^3$	a=1.1 b=0.004 x=0.2	День тижня число і місяць
8	$a = \frac{2\cos(x - \pi/6)}{1/2 + \sin^2 y};\ \ b = 1 + \frac{z^2}{3 + z^2/5}$	x=1.426 y=-1.220 z=3.5	Дата у форматі дд-мм-рр
9	$w = \sqrt{x^2 + b} - b^2\sin^3(x+a)/x;\ \ y = \cos^2 x^3 - \frac{x}{\sqrt{a^2 + b^2}}$	a=1.5 b=15.5 x=-2.8	День тижня та час
10	$c = \left\lvert x^{y/x} - \sqrt[3]{y/x}\right\rvert;\ \ f = (y-x)\frac{y - z/(y-x)}{1 + (y-x)^2}$	x=1.825 y=18.225 z=-3.298	Час у форматі гг:хх та дата дд-мм-рр

Лабораторна робота № 2

Основні алгоритмічні структури мови Java

1. У середовищі NetBeans створити новий проект. Додати до цього проекту новий клас.

2. У створеному класі описати метод, що обчислює значення функції, яка задана у таблиці.

Створити тестовий клас для тестування головного класу програми. Додати до нього методи тестування метода, який був створений у п.2. Виконати тестування цього метода.

3. Розробити метод, що за вказаними значеннями кроку, початку та кінця інтервалу обчислює кількість кроків для табулювання.

Створити тестові методи для нього і виконати тестування.

4. Створити методи, що створюють масиви значень функції (y) та її аргументу (x) в усіх точках вказаного інтервалу із заданим кроком. (розмір масивів обчислити програмно за допомогою метода з п.3). *Масиви повинні бути описані як* ***private*** *і для них потрібно створити методи доступу до їхніх елементів за номерами.*

Створити тестові методи для них і виконати тестування.

5. Створити методи, які після формування масивів, знаходять номери найбільшого та найменшого елементів масиву значень функції, та методи, що обчислюють та суму та середнє арифметичне елементів масиву значень функції.

6. Створити тестові методи для методів з п.5 і виконати тестування

7. Вивести найбільший та найменший елементи масиву значень функції, вказавши їхні номери і відповідні значення аргументу.

8. Додати до створеного класу метод main, перетворивши, таким чином, його на автономну програму. Скомпілювати і виконати програму

Короткі теоретичні відомості

Типи масиву

Типи масиву використовуються для визначення масивів – упорядкованих наборів однотипних змінних. Ви можете визначити масив над будь-яким існуючим у мові типом, включаючи типи, визначені користувачем. Крім того, можна користатися масивами масивів чи багатовимірними масивами. Коротко говорячи, якщо ми можемо створити змінну деякого типу, виходить, ми можемо створити і масив змінних цього типу. Разом з тим створення масивів у мові Java може показатися вам незвичним, тому що воно вимагає застосування оператора **new**.

Приклад 1. Описання масивів і виділення пам'яті для масивів

```
int[] myIntArray; //описання масиву цілих чисел
myIntArray = new int[8];
    // створення масиву з 8 цілих чисел
MyType[] myObjectArray;
    // описання масиву об'єктів типу MyType
myObjectArray = new MyType[5];
   // створення масиву з 5 елементів типу MyType
```

Оператор **new** дає команду оболонці часу виконання виділити необхідну кількість пам'яті під масив. Як видно з цього прикладу, не треба повідомляти розмір масиву тоді ж, коли ви створюєте змінну-масив. Після того, як створили масив оператором **new**, доступ до цього масиву здійснюється точно так само, як у мовах C чи Pascal.

Приклад 2. Присвоювання значень елементам масивів

```
myIntArray[0] = 0;
myIntArray[1] = 1;
myIntArray[2] = 2;
myObjectArray[0] = new MyType();
myObjectArray[1] = new MyType();
myObjectArray[2] = new MyType();
myObjectArray[0].myDataMember = 0;
myObjectArray[1].myDataMember = 1;
myObjectArray[2].myDataMember = 2;
```

Масиви в мові Java мають три важливих переваги перед масивами в інших мовах. По-перше, програмісту не треба вказувати розмір масиву при його оголошенні. По-друге, будь-який масив у мові Java є змінною – а це значить, що його можна передати як параметр методу і використовувати як значення, що повертається методом. І по-третє, завжди легко довідатися, який розмір даного масиву в будь-який момент часу. Наприклад, так визначається розмір масиву, що був оголошений вище.

Приклад 3. Отримання довжини масиву

```
int len = myIntArray.length;
System.out.println("Length of myIntArray=" + len);
```

Багатовимірні масиви у мові Java визначаються, як "масиви, елементами яких є масиви". Тобто двовимірний масив – це масив, елементами якого є лінійні масиви. Наприклад, так відбувається робота з двовимірним масивом.

Приклад 4. Описання та робота з двовимірним масивом

```
double[][] m;
m = new double[3][4];
// масив з трьох рядків, у кожному по 4 елементи
m[1][3] = 5.4;
// присвоювання значення елементу,
// що знаходиться у першому рядку під номером 3
double[][] z;
z = new double[3][];  // массив з трьох рядків
z[0] = new double[1]; // у першому рядку – один
елемент
z[1] = new double[2]; // у другому рядку – два
z[2] = new double[3]; // у третьому – три
z[2][2] = 1.5; // припустиме присвоювання
z[0][1] = 5.3; // ПОМИЛКА! У першому рядку є
лише один елемент
                  // із індексом 0
```

Управляючі структури мови Java

У мові Java існують такі управляючі структури:

1.***Оголошення, вирази та інструкції (оператори).***

Як інструкції, що завершуються символом "крапка с комою", можуть бути використані вирази таких категорій:

a. вирази присвоювання, в яких використовується базовий (=) та складений оператори присвоювання виду оп=;

b. префіксні та постфіксні форми виразів з операторами інкременту (++) та декременту (--);

c. конструкції виклику методів;

d. вирази створення об'єктів за допомогою оператора **new**.

Оголошення використовуються для описання (локальних) змінних та ініціалізації їх початковими значеннями.

2. ***Блоки***.

Фігурні дужки, { та }, використовуються для поєднання декількох виразів у блок (він може бути і порожнім). Блок дозволяється використовувати в будь-якому місці коду, де передбачено застосування інструкції, оскільки блок, як такий, є складеною інструкцією.

3. ***if … else***.

Найбільш поширеною формою управляючих структур, що використовуються для зміни порядку обчислень в залежності від значення логічного виразу, є конструкція **if … else**, синтаксис якої виглядає таким чином:

```
if (ЛогічнийВираз) Інструкція1
else Інструкція2
```

Спочатку виконується перевірка значення логічного виразу. Якщо результат дорівнює true, виконується Інструкція1, інакше (і при наявності необов'язкового елементу **else**) – Інструкція2. Одна конструкція if … else може бути вкладена всередину елементу **else** іншої з метою перевірки послідовності логічних умов. Також така конструкція може бути вкладена замість Інструкції1, але тоді треба пам'ятати, що елемент **else** завжди буде відноситися до найближчого **if**, яке не "закрито" своїм **else**.

4. ***switch***.

Конструкція **switch** дозволяє передавати управління тому чи іншому блоку коду, що позначений іменованою міткою, в залежності від значення цілочисельного виразу. Загальний синтаксис **switch** можна представити таким чином:

```
switch (ЦілочисельнийВираз) {
          case n: Інструкції
          case m: Інструкції
          ...
          default: Інструкції
   }
```

Тіло **switch**, відоме як блок перемикачів, містить набори інструкцій, яким передують мітки, що починаються з службового слова **case**. Кожній мітці case ставиться у відповідність ціла константа. Якщо значення цілочисельного виразу співпаде з зі значенням деякої мітки, управління буде передано першій інструкції, що йде після цієї мітки. Якщо спів падань не знайдено, виконуються інструкції блоку **default**. Якщо ж мітка **default** відсутня, виконання **switch** завершується. При передаванні управління відповідній мітці виконуються всі наступні за нею інструкції, навіть ті, що мають свої власні мітки **case**. Якщо треба вийти з блоку **switch**, треба використати інструкцію **break**.

5. ***while*** та ***do … while***.

Синтаксис циклічної конструкції ***while*** виглядає так:

```
while (ЛогічнийВираз)
        Інструкція
```

Спочатку виконується перевірка значення логічного виразу. Якщо результат дорівнює **true**, виконується Інструкція (як інструкція може бути використаний блок), після чого логічний вираз перевіряється знов і процес повторюється до тих пір, поки в результаті перевірки не буде отримано значення **false**, – в цьому випадку управління буде передане першій інструкції коду, що іде за межами конструкції **while**.

Іноді виникає необхідність забезпечити циклічне виконання блоку інструкцій якнайменше один раз. З цією метою використовується конструкція **do … while**, синтаксис якої може бути описаний таким чином:

```
do
        Інструкція
   while (ЛогічнийВираз)
```

В цьому випадку перевірка істинності логічного виразу виконується після виконання тіла циклу. Цикл повторюється до тих пір, поки в результаті перевірки виразу не буде отримане значення false. В якості тіла циклу **do … while** найчастіше використовується блок інструкцій.

6.***for***.

Вираз for застосовується для організації циклічного переходу по значеннях із вказаного діапазону і в загальному випадку виглядає таким чином:

```
for ( СекціяІніціалізації;
        ЛогічнийВираз;
        СекціяЗмін) Інструкція
```

Секція ініціалізації дозволяє ініціалізувати (можливо, з попереднім об'явленням) змінні циклу і виконується лише один раз. Логічний вираз піддається перевірці, та її результат, якщо він дорівнює true, дає підставу для виконання тіла циклу, після чого обчислюються вирази секції змін, і логічний вираз перевіряється у черговий раз. Цикл повторюється доти,

поки результатом перевірки не стане значення false. Кожна з двох частин конструкції **for** – СекціяІніціалізації та СекціяЗмін – може бути представлена списком виразів, відокремлених символом ","(кома). Вирази у подібних списках обчислюються зліва направо.

Усі секції заголовка циклу **for** є необов'язковими. Якщо СекціяІніціалізації опускається, на її місці залишається лише символ крапки з комою. Якщо ж із заголовка виключається ЛогічнийВираз, то в якості логічної умови мається на увазі літерал true. Виключення усіх трьох секцій призводить до того, що цикл стає "нескінченим":

```
for ( ; ; )
      Інструкція
```

Вихід з такого циклу повинен бути забезпечений за допомогою інших засобів, таких як команда break (її ми розглянемо далі) або інструкції викидання виключення.

Цикли for, у відповідності до прийнятої угоди, застосовуються тільки в тих випадках, коли необхідно забезпечити "проходження" по значеннях із визначених діапазонів. Якщо вирази ініціалізації та зміни змінних застосовуються не за призначенням і не пов'язані з логічною умовою циклу, подібні дії можна кваліфікувати не інакше як порушення угоди і ознака порочного стилю програмування.

7. ***for* (*each*)**.

Починаючи з версії Java5 у мові Java з'явилась нова конструкція, призначена для виконання ітерації по масиву або колекції. Вона виглядає так:

```
   for (<тип елементу> <формальне ім'я> : <масив,
або колекція>)
       Інструкція
```

8. ***Мітки***.

Інструкції програми можуть бути позначені мітками (labels). Мітка являє собою змістовне ім'я, що дозволяє посилатися на відповідну інструкцію:

```
   Мітка: Інструкція
```

Звертатися до мітки дозволено тільки за допомогою команд **break** та **continue** (вони розглядатимуться далі).

9. ***break***.

Інструкція **break** застосовується для завершення виконання коду будь-якого блоку. Існують дві форми інструкції – безіменна:

```
break;
```

та іменована:

```
break мітка;
```

Безіменна команда **break** перериває виконання коду конструкцій **switch**, **for**, **while** або **do** і може використовуватися лише всередині цих конструкцій. Команда **break** у іменованій формі може перервати виконання будь-якої інструкції, що помічена відповідною міткою.

Команда **break** найчастіше використовується для примусового виходу з тіла циклу. А для виходу із вкладеного циклу чи блоку, достатньо позначити міткою зовнішній блок і вказати її в інструкції **break** як показано в наступному прикладі:

Приклад 2.4. Використання поміченого **break**

```
private float[][] matrix;
    public boolean workOnFlag(float flag) {
        int y, x;
        boolean found = false;
        search:
       for (y = 0; y < matrix.length; y++) {
       for (x = 0; x < matrix[y].length; x++) {
                if (matrix[y][x] == flag) {
                    found = true;
                    break search;
                }
            }
        }
        if (!found) {
            return false;
        }
      // А тут знайдене значення matrix[y][x]
        // деяким чином обробляється
        return true;
    }
```

Відмітимо, що іменована інструкція **break** – це зовсім не те ж саме, що й сумнозвісна команда **goto**. Інструкція **goto** дозволяє "стрибати" по коду без жодних обмежень, переплутуючи порядок обчислень і збиваючи читача з глузду. Команди же **break** і **continue**, що посилаються на мітку, дозволяють лише акуратно залишити відповідний блок і забезпечити його повторення, при цьому потік обчислень залишається цілком очевидним.

10. ***continue***. Команда **continue** застосовується лише у контексті циклічних конструкцій і передає управління на кінець тіла циклу. В ситуації з **while** і **do** це призводить до виконання перевірки умови циклу, а при використанні в тілі **for** інструкція **continue** провокує передавання управління секції змін значень змінних циклу.

Як і **break**, команда **continue** дозволяє використання в двох формах – без імені: **continue**; і іменованій: **continue** мітка. Команда **continue** у формі без імені мітки передає управління в кінець поточного циклу, а іменована – в кінець циклу, позначеного відповідною міткою. Мітка повинна відноситися до циклічного виразу.

11. ***return***. Інструкція **return** завершує виконання методу і передає управління до коду-ініціатору. Якщо метод не повертає значень, інструкція виглядає просто: **return**. Якщо ж в оголошенні методу вказано тип параметра, що повертається, до складу команди **return** повинен бути включеним вираз, який може бути присвоєний змінній оголошеного типу. Наприклад, якщо метод повертає значення типу **double**, в інструкції **return** дозволяється використовувати вирази, що відносяться до типів **double**, **float** або будь-якого цілих типів.

12. ***goto***. У мові Java НЕМАЄ інструкції **goto**, що має змогу передавати управління довільному фрагменту коду, хоча у споріднених мовах аналогічні засоби передбачені. Всі засоби, що були розглянуті раніше, дозволяють створювати зрозумілий і надійний код, а також обходитися без допомоги **goto**.

13. Для обробки виключень, тобто ситуацій, що могли б привести до краху програми (наприклад, ділення на нуль, помилка введення-виведення) використовують конструкцію **try…catch…finally…** Обробка виключень у Java спирається в основному, на конструкції C++ (хоча ідейно більше схожа на Object Pascal). У місці, де виникла проблема, ви, можливо, ще не знаєте що з нею робити, проте знаєте, що просто ігнорувати її не можна – треба зупинитись і передати управління блоку обробки.

Модульне тестування з використанням JUnit у середовищі NetBeans

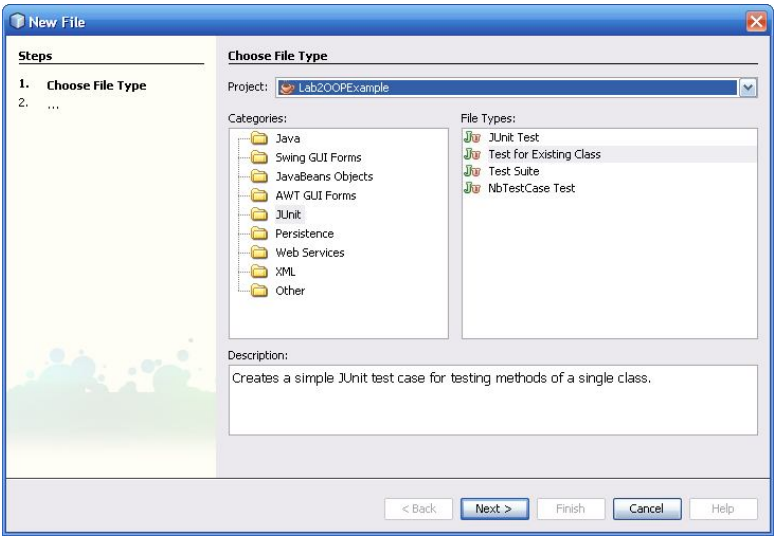
Розглянемо, як створюються та виконуються тести на прикладі програми, аналогічної до тієї, що була складена у лабораторній роботі № 1. Проте, на практиці, для того, щоб виконувати наступні кроки треба створити головний клас програми і додати до нього хоча б один метод.

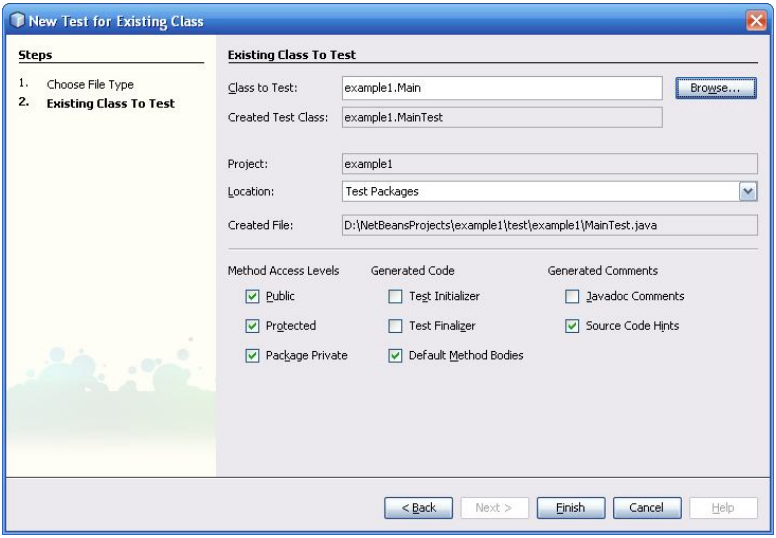
Нехай у нас є проект *Example1*, що складається з пакету **example1** в якому міститься єдиний клас Main (у файлі Main.java).

```
package example1;

/**
 * @author Eugeny
 */
public class Main {

    /**
     * Метод, що обчислює значення y за формулою
з завдання
     * @param a перший параметр
     * @param b другий параметр
     * @param x третій параметр
     * @return обчислене значення y
     */
   public double calcY(double a, double b, double x) {
      return (Math.pow(a, 2 * x) + Math.pow(b, -x) *
                  Math.cos(a + b) * x) / (x + 1);
     }
     /**
      * Метод, що обчислює значення r за формулою
з завдання
      * @param a перший параметр
      * @param b другий параметр
      * @param x третій параметр
      * @return обчислене значення r
      */
```

```
    public double calcR(double a, double b, double x) {
            return Math.sqrt(x * x + b - b * b *
                 Math.pow(Math.sin(x + a), 3) / x);
        }
        /**
         * метод присвоює a,b,x значення з завдання
          * та обчислює значення змінних r та y -
calcR() i calcY()
         * Після того выводяться початкові та об-
числені значення
         */
        public void printResults() {
            double a = 0.3;
            double b = 0.9;
            double x = 0.61;
            double r = calcY(a, b, x);
            double y = calcR(a, b, x);
                System.out.printf("x=%5.3f  y=%5.3f
z=%5.3f%n", a, b, x);
            System.out.printf("y=%8.4f%n", y);
            System.out.printf("r=%8.4f%n", r);
        }
        /**
         * @param args аргументи командного рядка
         */
        public static void main(String[] args) {
            Main program = new Main();
            program.printResults();
        }
    }
```

Для створення класу модульного тестування у середовищі NetBeans оберіть пункт New File… у меню File. У вікні, що відкриється, обрати категорію **JUnit** і в цій категорії тип файлу Test for Existing Class. Для продовження треба натиснути кнопку "**Next >**". Відкриється наступне вікно.





Укажіть в ньому, для якого класу створюються тестовий клас (**Class to Test:**). Для обрання потрібного класу натисніть кнопку **Browse…** та оберіть потрібний клас. В нашому випадку це клас ***example1.Main.*** Після налаштування створюваного тестового класу треба натиснути "Finish". Якщо Ви робите це перший раз, то буде запропоновано вибрати версію бібліотеки модульного тестування JUnit 3.x або JUnit 4.x. Оберемо JUnit 4.x.

Буде створено клас (у нашому випадку це буде клас MainTest), який буде розміщено у розділі Test Packages.

При створенні тестового класу, середовище NetBeans формує шаблони тестових методів, їх треба відредагувати таким чином:

- Метод для тестування метода main() (він буде називатися testMain()) можна (або навіть, треба) видалити.
- Оскільки метод printResults() лише викликає інші методи і виводить результати їх виконання, то його можна не тестувати (тобто метод testPrintResults() теж можна видалити).
- На початку тестового класу додайте описання константи EPS – точності для порівнянь.
- Методи testCalcY() та testCalcR() треба змінити так, щоб за їх допомогою можна було тестувати відповідні методи головного класу програми. Спочатку треба видалити останні два рядки кожного метода – вони були додані туди тільки для того, щоб пересвідчитися, що тести насправді писалися людиною, а не були використані шаблони.
- У кожному з методів впишіть початкові значення, що треба присвоїти змінним, які передаються цим методам (в нашому випадку – змінні a, b, x). Ці значення треба підібрати таким чином, щоб можна було протестувати різні випадки в процесі обчислення функцій.
- Змінній **expResult** треба присвоїти значення, що очікується у результаті виконання метода, що буде тестуватися (його треба обчислити вручну, або за допомогою калькулятора).
- Після виклику метода, що тестується, змінній **result** буде присвоєне значення, яке обчислить метод.
- Значення змінних **expResult** та **result** треба порівняти за допомогою методу assertEquals(arg1,arg2,eps), де arg1, arg2 – значення, що порівнюються, eps – похибка порівняння. Цей метод перевіряє, чи виконується нерівність $|arg1 - arg2| < eps$, тобто якщо eps – мале число, то можна вважати, що arg1 та arg2 приблизно рівні.
- Для кожного з методів класу, що тестується, можна (і треба) створити декілька тестових методів (якщо ж в методі, який тестується, є оператори розгалуження, то потрібно створити стільки тестів, скільки є різних шляхів виконання метода, а крім того, ОБОВ'ЯЗКОВО ТРЕБА перевірити умови на межі переходу в умовному операторі).
- Виконати тести можна обравши в головному меню NetBeans у пункті Run підпункт Test "<Ім'я проекту>" (в нашому прикладі – Test "example1".

• Після виконання тесту у вікні тестування будуть показані результати: **passed** (зеленим кольором) означає, що відповідний тест "пройшов", **FAILED** (червоним кольором) означає, що тест "не пройшов" і треба перевірити, чому так сталося, та знайти помилку. Якщо хоча б один тест не пройшов, то вважається, що проект в цілому не пройшов тестування і не може здаватися, як завершений продукт. Треба виправляти помилки до тих пір, поки ВСІ тести будуть "проходити".

• Відмітимо, що тестовий клас найчастіше виходить більшим, ніж той клас, який він тестує, проте тестовий клас складається з простих методів і тому його розроблювати нескладно.

Приклад тестового класу:

```
package example1;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Eugeny
 */
public class MainTest {

    public static final double EPS = 1e-6;
    // припустима точність порівнянь

    public MainTest() {
    }

    @BeforeClass
     public static void setUpClass() throws
Exception {
    }

    @AfterClass
     public static void tearDownClass() throws
Exception {
    }
```

```
    @Test // Тест загального випадку, очікуємо
результат 1.0
       public void testCalcY() {
      System.out.println("calcY (1.0, -1.0, 0.0)");
           double a = 1.0;
           double b = -1.0;
           double x = 0.0;
           Main instance = new Main();
           double expResult = 1.0;
          double result = instance.calcY(a, b, x);
           assertEquals(expResult, result, EPS);
       }

       @Test
   // Тест окремого випадку, очікуємо результат
"+НЕСКІНЧЕНОСТЬ"
       public void testCalcY0() {
            System.out.println("calcY (1.0, -1.0,
0.0)");
           double a = 1.0;
           double b = -1.0;
           double x = -1.0;
           Main instance = new Main();
          double expResult = Double.POSITIVE_INFINITY;
          double result = instance.calcY(a, b, x);
           assertEquals(expResult, result, EPS);
       }

       @Test // Тест загального випадку, очікуємо
результат 2.0
       public void testCalcR() {
       System.out.println("calcR (0.0, 0.0, 2.0)");
           double a = 0.0;
           double b = 0.0;
           double x = 2.0;
           Main instance = new Main();
           double expResult = 2.0;
           double result = instance.calcR(a, b, x);
```

```
            assertEquals(expResult, result, EPS);
        }

        @Test
    // Тест окремого випадку, очікуємо результат
"НЕ-ЧИСЛО", 0/0
        public void testCalcR0() {
       System.out.println("calcR (0.0, 0.0, 0.0)");
            double a = 0.0;
            double b = 0.0;
            double x = 0.0;
            Main instance = new Main();
            double expResult = Double.NaN;
            double result = instance.calcR(a, b, x);
            assertEquals(expResult, result, EPS);
        }
    }
```

Варіанти завдань

№	Функція	Умова	Вхідні дані	Діапазон та крок зміни аргумента	Номери елементів, для тестування
1	$y=\begin{cases} ax^2 \ln x \\ 1 \\ e^{ax}\cos bx \end{cases}$	$0.7 < x \le 1.4$ $x \le 0.7$ $x > 1.4$	a=-0.5 b=2	$x \in [0;3]$ $\Delta x = 0.004$	175, 350, 750
2	$y=\begin{cases} px^2 - 7/x^2 \\ ax^3 + 7\sqrt{x} \\ \lg(x+7\sqrt{x}) \end{cases}$	$x < 1.7$ $x - 1.7$ $x > 1.7$	a=1.5	$x \in [0.8;2]$ $\Delta x = 0.005$	0, 180, 240
3	$y=\begin{cases} ax^2 + bx + c \\ a/x + \sqrt{x^2+1} \\ (a+bx)/\sqrt{x^2+1} \end{cases}$	$x < 1.4$ $x - 1.4$ $x > 1.4$	a=2.8 b=-0.3 c=4	$x \in [0;2]$ $\Delta x = 0.002$	0, 700, 1000
4	$y=\begin{cases} px^2 - 7/x^2 \\ ax^3 + 7\sqrt{x} \\ \ln(x+7\sqrt{\lvert x+a \rvert}) \end{cases}$	$x < 1.3$ $x - 1.3$ $x > 1.3$	a=1.65	$x \in [0.7;2]$ $\Delta x = 0.005$	0, 120, 260

Продовж. варіанта

№	Функція	Умова	Вхідні дані	Діапазон та крок зміни аргумента	Номери елементів, для тестування
5	$y=\begin{cases}1.5a\cos^2 x\\(x-2)^2+6a\\3a\cdot\operatorname{tg} x\end{cases}$	$x\le 0.3$ $0.3<x\le 2.3$ $x>2.3$	*a=2.3*	$x\in[0.2;2.8]$ $\Delta x=0.002$	50, 1050, 1300
6	$y=\begin{cases}x\sqrt{x-a}\\x\sin ax\\e^{ax}\cos ax\end{cases}$	$x>a$ $x=a$ $x<a$	*a=2.4*	$x\in[1;5]$ $\Delta x=0.01$	0, 140, 400
7	$y=\begin{cases}bx-\operatorname{tg} bx\\bx+\lg bx\end{cases}$	$bx\le 0.45$ $bx>0.45$	*b=1.5*	$x\in[0.1;1]$ $\Delta x=0.001$	0, 200, 900
8	$y=\begin{cases}\sin x\lg x\\\cos^2 x\end{cases}$	$x>3.4$ $x\le 3.4$		$x\in[2;5]$ $\Delta x=0.005$	0, 280, 600
9	$y=\begin{cases}\lg(x+1)\\\sin^2\sqrt{ax}\end{cases}$	$x>1.2$ $x\le 1.2$	*a=20.3*	$x\in[0.5;2]$ $\Delta x=0.005$	0, 140, 300
10	$y=\begin{cases}(\ln^3 x+x^2)/\sqrt{x+t}\\\cos x+t\sin^2 x\end{cases}$	$x\le 0.9$ $x>0.9$	*t=2.2*	$x\in[0.2;2]$ $\Delta x=0.004$	0, 175, 450

Лабораторна робота № 3

Посилальні типи даних мови Java
Короткі теоретичні відомості

Описання власних типів

У мові програмування Java, програміст може визначати власні типи через створення відповідних класів. Опис класу найчастіше розміщується в окремому файлі, ім'я якого повинно співпадати з іменем класу.

Приклад описання класу

```
package lab3;
import java.util.*;

   public class Lab3 {
    private int x; // змінна екземпляра класу
    private int y = 71; // змінна екземпляра класу
     public final int CURRENT_YEAR = 2007; // константа
     protected static int bonus; // змінна класу
     static String version = "Java SE 7"; // змінна класу
       protected Calendar now;

       public int method(int z) { // параметр метода
          z++;
          int a; // локальна змінна метода
          //a++; // помилка компіляції, значення не задано
           a = 4; // ініціалізація
           a++;
           now = Calendar.getInstance(); // ініціалізація
            return a + x + y + z;
       }
   }
```

У розглянутому прикладі в якості змінних екземпляра класу, змінних класу та локальних змінних методу використані дані примітивних типів, що не є посиланнями на об'єкти (крім String). Дані можуть бути посилан-

нями, призначити яким реальні об'єкти можна за допомогою оператора **new**.

Конструктори

Конструктор – це метод, який автоматично викликається при створенні об'єкта класу і виконує дії з ініціалізації об'єкта. Конструктор має те ж ім'я, що й клас; викликається не по імені, а тільки разом з ключовим словом new при створенні екземпляра класу. конструктор не повертає значення, але може мати параметри і бути перевантаженим.

Приклад описання класу, що містить конструктори

```
package lab2;

public class Quest {
    private int id;
    private String text;
    // конструктор без параметрів (за замовчуванням)
    public Quest() {
        super(); /* якщо клас буде оголошений
без конструктора, то
                компілятор надасть його саме
в такому вигляді */
    }
    // конструктор з параметрами
    public Quest(int idc, String txt) {
        super();
/* виклик конструктора суперкласу явним чином
   не є обов'язковим, компілятор вставить його
автоматично */
        id = idc;
        text = txt;
    }
}
```

Об'єкт класу Quest може бути створений двома способами, які викликають один з конструкторів:

```
    Quest a = new Quest();
    //ініціалізація полів значеннями за замовчуванням
    Quest b = new Quest(71, "Сколько бит занимает boolean?");
```

Оператор new викликає конструктор, тому в круглих дужках можуть стояти аргументи, що передаються конструктору. Якщо конструктор в класі не визначений, Java надає конструктор за замовчуванням без параметрів, який ініціалізує поля класу значеннями за замовчуванням, наприклад: 0, false, null. Якщо ж конструктор з параметрами визначений, то конструктор за замовчуванням стає недоступним і для його виклику необхідно явне оголошення такого конструктора.

В наступному прикладі оголошено клас Point з двома полями (атрибутами), конструктором і методами для ініціалізації та отримання значень атрибутів.

Приклад програми, що описує та використовує клас Point

```
package lab2;
public class Point {

    double x;
    double y;

    public Point(double xx, double yy) {
        x = xx;
        y = yy;
    }
    public double getX() {
        return x;
    }
    public double getY() {
        return y;
    }
}

package lab2;
public class LocateLogic {
```

```
  public double calculateDistance(Point t1, Point t2) {
        /* обчислення відстані */
        double dx = t1.getX() - t2.getX();
        double dy = t1.getY() - t2.getY();
        return Math.hypot(dx, dy);
    }
}

package lab2;
public class Runner {
    public static void main(String[] args) {
        // локальні змінні не є членами класу
        Point t1 = new Point(5, 10);
        Point t2 = new Point(2, 6);
       System.out.print("расстояние равно : " +
   new LocateLogic().calculateDistance(t1, t2));
    }
}
```

Рядки

Рядок у мові Java – це основний носій текстової інформації. Це не масив символів типу char, а об'єкт відповідного класу. Системна бібліотека Java містить класи String, StringBuilder і StringBuffer, що підтримують роботу з рядками і визначені в пакеті java.lang, що підключається автоматично. Ці класи оголошені як final, що означає неможливість створення власних породжених класів з властивостями рядків.

Клас String

Кожен рядок, створюваний за допомогою оператора **new** або за допомогою літерала (укладений в подвійні апострофи), є об'єктом класу String. Особливістю об'єкта класу String є те, що його значення не може бути змінено після створення об'єкту за допомогою якого методу класу, так як будь-яка зміна рядка приводить до створення нового об'єкта. При цьому посилання на об'єкт класу String можна змінити так, щоб воно вказувало на інший об'єкт і тим самим на інше значення.

Клас String підтримує кілька конструкторів, наприклад:

String(), String(String str), String(byte asciichar[]), String(char[] unicodechar), String(StringBuffer sbuf), String(StringBuilder sbuild) та інші.

Коли Java зустрічає літерал, укладений в подвійні лапки, автоматично створюється об'єкт типу String, на який можна встановити посилання. Таким чином, об'єкт класу String можна створити, присвоюючи посиланню на клас значення існуючого літерала, або за допомогою оператора new і конструктора, наприклад:

```
String s1 = "berkut.homelinux.com";
String s2 = new String("berkut.homelinux.com");
```

Клас String містить наступні (тут наведені основні, але не всі) методи для роботи з рядками:

`String concat(String s)` або "+" – злиття рядків;

`boolean` `equals(Object ob)` та `equalsIgnore Case(String s)` – порівняння рядків з урахуванням та без урахування регістра відповідно;

`int` `compareTo(String s)` та `compareToIgnore Case(String s)` – лексикографічне порівняння рядків з урахуванням та без урахування регістра. Метод виконує віднімання кодів символів рядка, що викликає метод та рядка, що передається до методу та повертає ціле значення. Метод повертає значення нуль у випадку, якщо equals() повертає значення true;

`boolean` `contentEquals(StringBuffer ob)` – порівняння рядка та вмісту об'єкта типу StringBuffer;

`String substring(`**`int`** `n,` **`int`** `m)` – отримання з рядку підрядка довжини m-n, починаючи з позиції n. Нумерація символів в рядку починається з нуля;

`String substring(`**`int`** `n)` – отримання з рядку підрядка, починаючи з позиції n;

`int` `length()` – визначення довжини рядка;

`int` `indexOf(`**`char`** `ch)` – визначення позиції символу у рядку;

`static` `String valueOf(значення)` – перетворення змінної примітивного типу у рядок;

`String toUpperCase()/toLowerCase()` – перетворення всіх символів рядка, який викликає метод у верхній/нижній регістр;

`String replace(`**`char`** `c1,` **`char`** `c2)` – заміна у рядку всіх входжень першого символу другим символом;

`String intern()` – заносить рядок в "пул" літералів та повертає його об'єктне посилання;

`String trim()` – вилучення всіх пропусків на початку та в кінці рядка;

char charAt(**int** position) – отримання символу із вказаної позиції (нумерація з нуля);

boolean isEmpty() – повертає true, якщо довжина рядка дорівнює 0;

static String format(String format, Object... args), format(Locale l, String format, Object... args) – генерує форматований рядок, отриманий з використанням формату, інтернаціоналізації та ін.;

String[] split(String regex), split(String regex, **int** limit) – пошук входження в рядок заданого регулярного виразу (розділителя) та поділ початкового рядка у відповідності з цим на масив рядків.

У всіх випадках виклику методів, яків потрібно змінити рядок, створюється новий об'єкт типу String.

Класи StringBuilder та StringBuffer

Класи StringBuilder та StringBuffer є "близнюками" та за призначенням наближені до класу String, але, на відмінність від останнього, вміст та розміри об'єктів класів StringBuilder та StringBuffer можна змінювати.

За допомогою відповідних методів та конструкторів об'єкти класів StringBuffer, StringBuilder та String можна перетворювати один до одного. Конструктор класу StringBuffer (так саме як і StringBuilder) може приймати в якості параметра об'єкт String або невід'ємний розмір буфера. Об'єкти цього класу можна перетворювати в об'єкт класу String методом toString() або за допомогою конструктора класу String.

Слід звернути увагу на такі методи:

void setLength(**int** n) – установка розміру буфера;

void ensureCapacity(**int** minimum) – установка гарантованого мінімального розміру буфера;

int capacity() – отримання поточного розміру буфера;

StringBuffer append(параметри) – додавання до вмісту об'єкта рядкового представлення аргументу, який може бути символом, значенням примітивного типу, масивом та рядком;

StringBuffer insert(параметри) – вставка символу, об'єкта або рядка в указану позицію;

StringBuffer deleteCharAt(**int** index) – вилучення символу;

StringBuffer delete(**int** start, **int** end) – вилучення підрядку;

`StringBuffer reverse()` – перестановка вмісту об'єкта у зворотному порядку.

В класі присутні також методи, аналогічні методам класу String, такі як replace(), substring(), charAt(), length(), getChars(), indexOf() та ін.

Варіанти завдання

Завдання 3.1.

Варіант 1. В кожному слові тексту k-у літеру замінити вказаним символом. Якщо k більше довжини слова, коригування не виконувати.

Варіант 2. В тексті кожну літеру замінити її порядковим номером в алфавіті. При виведенні в одному рядку друкувати текст з двома пропусками між літерами, в наступному рядку внизу під кожною літерою друкувати її номер.

Варіант 3. В тексті після літери Р, якщо вона не остання в слові, помилково надрукована літера А замість О. Внести виправлення в текст.

Варіант 4. В тексті слова заданої довжини замінити вказаним рядком, довжина якого може не співпадати з довжиною слова.

Варіант 5. В тексті після k-го символу вставити заданий підрядок.

Варіант 6. Після кожного слова тексту, яке закінчується заданим підрядком, вставити вказане слово.

Варіант 7. В залежності від ознаки (0 або 1) в кожному рядку тексту вилучити вказаний символ скрізь, де він зустрічається, або вставити його після k-го символу.

Варіант 8. З невеликого тексту вилучити всі символи, крім пропусків, які не є літерами. Між послідовностями літер, що йдуть одна за одною, залишити хоча би один пропуск.

Варіант 9. З тексту вилучити всі слова вказаної довжини, які починаються на приголосну літеру.

Варіант 10. Вилучити з тексту його частину, що обмежена двома символами, які вводяться (наприклад, між дужками(' та ')' або між зірочками '*' і т.д.).

Завдання 3.2.

Створити класи, специфікації яких наведені нижче. Визначити конструктори та методи setТип(), getТип(), toString(). Визначити додатково методи в класі, що створює масив об'єктів. Задати критерій вибору даних та вивести ці дані на консоль.

Варіант 1. **Student:** id, Прізвище, Ім'я, По батькові, Дата народження, Адреса, Телефон, Факультет, Курс, Група.

Скласти масив об'єктів. Вивести:

a) список студентів заданого факультету;

b) списки студентів для кожного факультету та курсу;

c) список студентів, які народились після заданого року;

d) список навчальної групи.

Варіант 2. **Customer:** id, Прізвище, Ім'я, По батькові, Адреса, Номер кредитної картки, Номер банківського рахунку.

Скласти масив об'єктів. Вивести:

a) список покупців в алфавітному порядку;

b) список покупців, у яких номер кредитної картки знаходиться в заданому інтервалі.

Варіант 3. **Patient:** id, Прізвище, Ім'я, По батькові, Адреса, Телефон, Номер медичної карти, Діагноз.

Скласти масив об'єктів. Вивести:

a) список пацієнтів, які мають вказаний діагноз;

b) список пацієнтів, номер медичної карти у яких знаходиться в заданому інтервалі.

Варіант 4. **Abiturient:** id, Прізвище, Ім'я, По батькові, Адреса, Телефон, Оцінки.

Скласти масив об'єктів. Вивести:

a) список абітурієнтів, які мають незадовільні оцінки;

b) список абітурієнтів, середній бал у яких вище заданого;

c) вибрати задане число n абітурієнтів, що мають найвищий середній бал (вивести також повний список абітурієнтів, що мають напівпрохідний бал).

Варіант 5. **Book:** id, Назва, Автор(и), Видавництво, Рік видання, Кількість сторінок, Ціна, Обкладинка.

Скласти масив об'єктів. Вивести:

a) список книг заданого автора;

b) список книг, що видані заданим видавництвом;

c) список книг, що випущені після заданого року.

Варіант 6. **House:** id, Номер квартири, Площа, Поверх, Кількість кімнат, Вулиця, Тип будівлі, Термін експлуатації.

Скласти масив об'єктів. Вивести:

a) список квартир, які мають задане число кімнат;

b) список квартир, які мають задане число кімнат та розташовані на поверсі, який знаходиться в заданому проміжку;

c) список квартир, які мають площу, що перевищує задану.

Варіант 7. **Phone:** id, Прізвище, Ім'я, По батькові, Адреса, Номер кредитної картки, Дебет, Кредит, Час міських та міжміських розмов.

Скласти масив об'єктів. Вивести:

a) відомості про абонентів, у яких час міських розмов перевищує заданий;

b) відомості про абонентів, які користувались міжміським зв'язком;

c) відомості про абонентів в алфавітному порядку.

Варіант 8. **Car:** id, Марка, Модель, Рік випуску, Колір, Ціна, Реєстраційний номер.

Скласти масив об'єктів. Вивести:

a) список автомобілів заданої марки;

b) список автомобілів заданої моделі, які експлуатуються більше n років;

c) список автомобилів заданого року випуску, ціна яких більше вказаної.

Варіант 9. **Product:** id, Найменування, Виробник, Ціна, Термін зберігання, Кількість.

Скласти масив об'єктів. Вивести:

a) список товарів для заданого найменування;

b) список товарів для заданого найменування, ціна яких не перевищує задану;

c) список товарів, термін зберігання яких більше заданого.

Варіант 10. **Train:** Пункт призначення, Номер поїзду, Час відправки, Число місць (загальних, купе, плацкарт, люкс).

Скласти масив об'єктів. Вивести:

a) список поїздів, які прямують до заданого пункту призначення;

b) список поїздів, які прямують до заданого пункту призначення та відправляються після заданої години;

c) список поїздів, які відправляються до заданого пункту призначення та мають загальні місця.

Лабораторна робота № 4

Застосування стандартної бібліотеки мови Java

1. Використовуючи програму з лабораторної роботи № 2 як допоміжний клас, розробити програму з графічним інтерфейсом користувача, яка у головному вікні дозволяє вводити дані та виводити результати відповідно до варіанту. Головне вікно програми повинне створюватись, як таке, що наслідується від стандартного класу JFrame, який належить до стандартної бібліотеки інтерфейсних елементів мови Java (Swing). Елементи, що розміщуються у цьому вікні, також повинні створюватись на основі бібліотеки Swing.

2. Результати роботи також вивести у вікні програми за допомогою стандартної бібліотеки класів мови Java.

3. В усіх варіантах треба обов'язково передбачити можливість введення даних для задачі за допомогою елементів графічного інтерфейсу користувача.

4. Якщо в процесі написання програми знадобиться переробити структуру методів головного класу з другої лабораторної роботи, то для з'ясування коректності цих змін виконати тестування цього класу за допомогою тестового класу, що був створений у другій роботі.

5. По закінченню виконання роботи, виконати її тестування.

Варіанти завдання

Варіант	Завдання
1	1. Обчислити суму найбільшого та найменшого. 2. Обчислити значення функції при аргументі, що дорівнює найбільшому елементу масиву, найменшому та середньому значенням елементів масиву. 3. Всі результати вивести у цьому же вікні.
2	1. Обчислити суму номерів найбільшого та найменшого елементів масиву. 2. Обчислити суму всіх від'ємних елементів масиву. 3. Обчислити середнє значення всіх додатних елементів масиву.
3	1. Визначити "відстань" (кількість елементів) між найбільшим та найменшим елементами масиву. 2. Визначити кількість від'ємних елементів масиву. 3. Обчислити суму та середнє арифметичне всіх додатних елементів масиву.

Продовж. варіанта

Варіант	Завдання
4	1. Визначити кількість елементів, ціла частина яких є непарним числом. 2. Визначити значення елемента, що йде після найбільшого елемента. Якщо такого немає, вивести NaN 3. Обчислити суму та середнє арифметичне елементів масиву, які більше вказаного числа.
5	1. Визначити, скільки разів у масиві зустрічається число, що дорівнює найбільшому елементу. 2. Обчислити суму всіх від'ємних елементів. 3. Знайти середнє арифметичне всіх елементів, що менше найбільшого елемента.
6	1. Знайти розв'язання лінійного рівняння $ax + b = 0$, де $\boldsymbol{a}$ – найбільший, $\boldsymbol{b}$ – найменший елементи масиву. 3. Знайти суму та середнє арифметичне всіх елементів масиву. 4. Обчислити кількість додатних елементів, які мають номери, більші, ніж найменший елемент масиву.
7	1. Знайти найбільший від'ємний та найменший додатний елементи масиву. 2. Обчислити середнє арифметичне всіх елементів масиву із парними номерами. 3. Обчислити суму натуральних логарифмів від модулів найбільшого та найменшого елементів.
8	1. Знайти найбільший елемент серед тих, що мають непарні номери. 2. Знайти суму і середнє арифметичне значення всіх елементів масиву, без урахування того, що був знайдений у п.2. 3. Обчислити суму усіх від'ємних елементів масиву, що мають номери, менші за найбільший елемент.
9	1. Знайти середнє арифметичне від'ємних елементів масиву. 2. Знайти суму елементів, більших значення, знайденого у п.2 3. Визначити кількість додатних елементів, що розташовані між найбільшим та найменшим елементами.
10	1. Знайти суму всіх елементів масиву. 2. Знайти суму тих елементів, що більше ніж різниця найбільшого та найменшого елементів. 3. Визначити кількість елементів, ціла частина яких є непарною та більше середнього арифметичного всіх елементів.

Порядок виконання роботи.

• У середовищі NetBeans створити новий проект. Для цього у головному меню програми оберіть: *File -> New Project*, *у діалоговому вікні, що відкриється оберіть категорію проекту "Java" та в ній проект – "Java Application"*.

• Додайте до цього проекту головний клас – вікно (JFrame). Для цього у головному меню програми оберіть: *File -> New File*, *у діалоговому вікні, що відкриється оберіть категорію файлу "Swing GUI Forms"* (*Форми графічного інтерфейсу користувача*), *в цій категорії оберіть "JFrame Form"* (*форма на основі класу JFrame*).

• *За допомогою маніпулятора "Миша" додайте з палітри "Palette" до головного вікна проекту елементи інтерфейсу*:

• *JPanel* (*для розділення вікна на окремі області – панелі*),

• *JLabel* (*для виведення повідомлень та написів у вікні*),

• *JTextField* (*для введення даних у програму*),

• *JButton* (*Для надання користувачеві можливості давати команди програмі*),

• *JTextArea*, *JList чи JTable* (*для відображення великої кількості даних*),

• *Будь-які інші, які вважаєте за потрібні.*

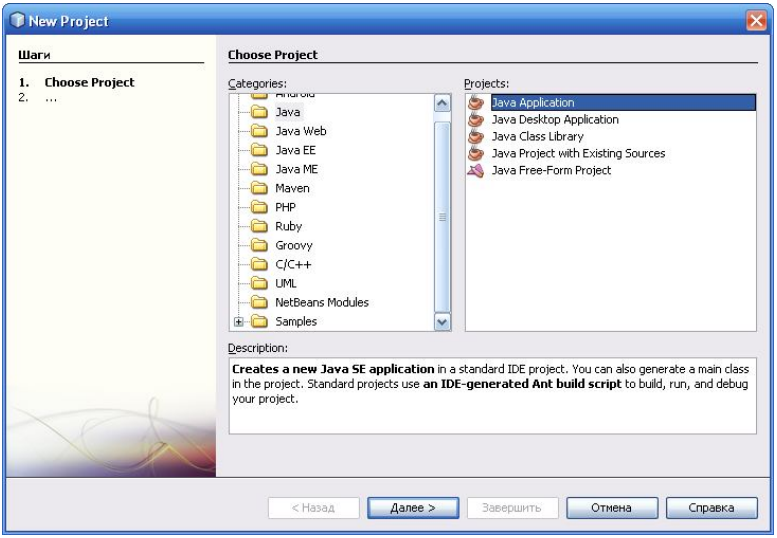
• *Скопіюйте у каталог src проекту файл з текстом класу, який був розроблений для Лабораторної роботи № 2 та використайте його для виконання наступного пункту.*

• *У контекстному меню елементу управління JButton* (*що ви розмістили у вікні у пункті С*) *оберіть*

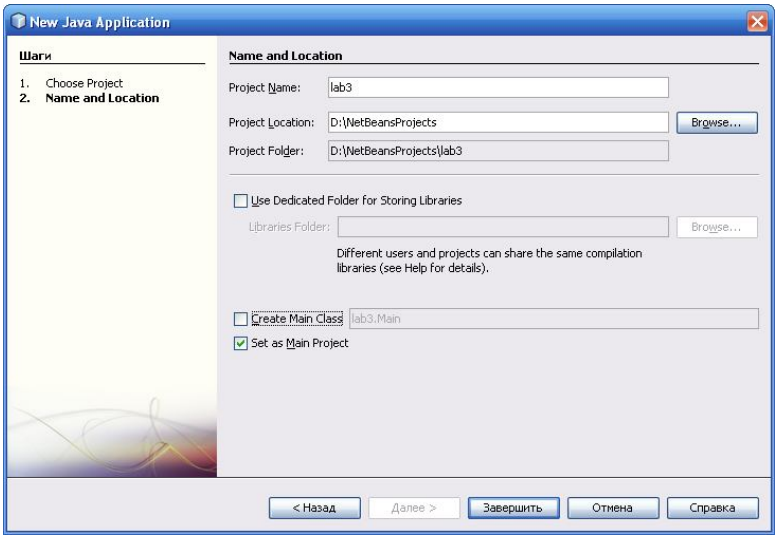
Events->Action->actionPerformed *та у тексті метода, що відкриється для редагування опишіть реакцію на натискання цієї кнопки.*

Приклад створення програми, що розв'язує квадратні рівняння, з використанням стандартних класів для створення графічного інтерфейсу користувача мови Java.

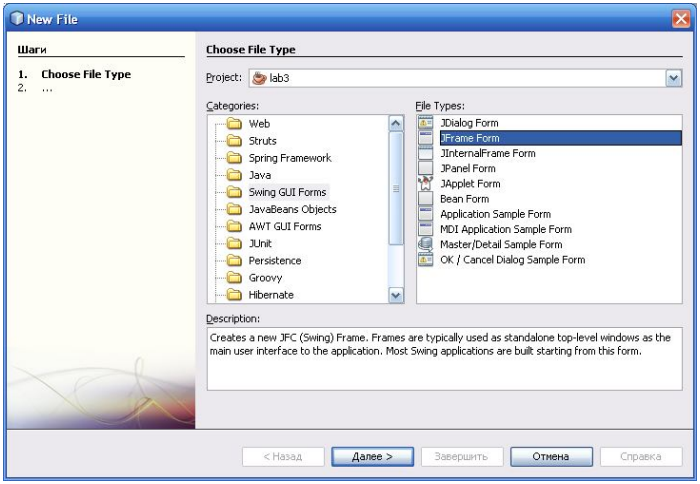
1. Створюємо новий проект: **File(Файл) -> New Project**, обираємо категорію (Categories) **Java**, а в ній тип проекту (Projects) – **Java Application**. Натискаємо "**Next >**" ("**Далее >**")…



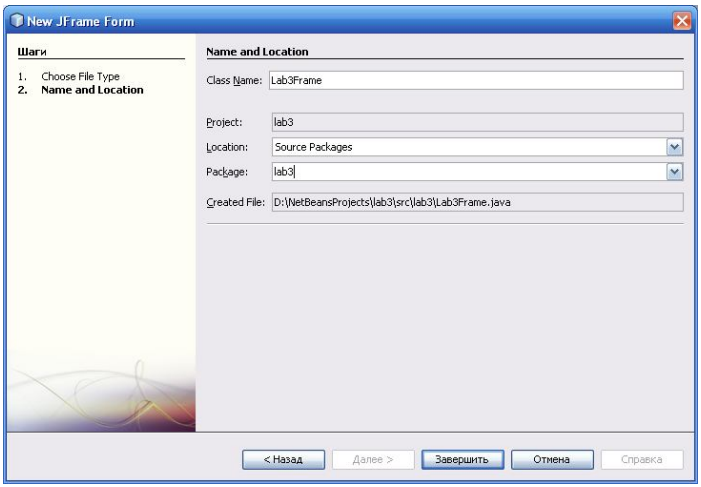
2. Вказуємо його ім'я, та розміщення. Знімаємо прапорець навпроти **Create Main Class**, залишаємо встановленим прапорець **Set as Main Project** та натискаємо **Finish** (Завершить):



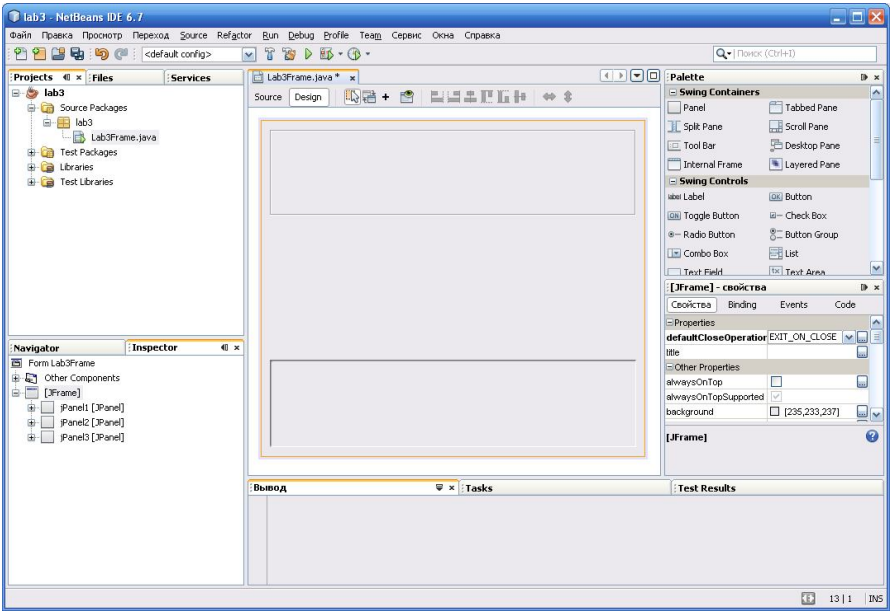
3.Додаємо до цього проекту новий файл типу JFrame Form з категорії Swing GUI Forms: (**File(Файл) -> New File…**). Натискаємо **"Next >" (Далее >)…**



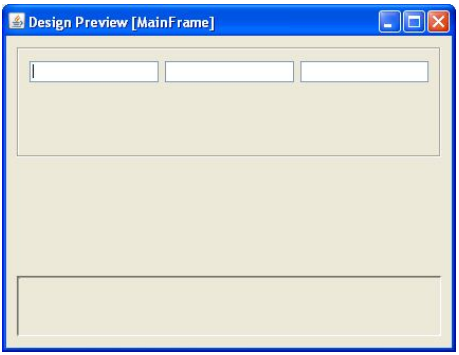
4. Вказуємо ім'я класу головної форми (Class Name) та пакету (Package), до якого ми її віднесемо:



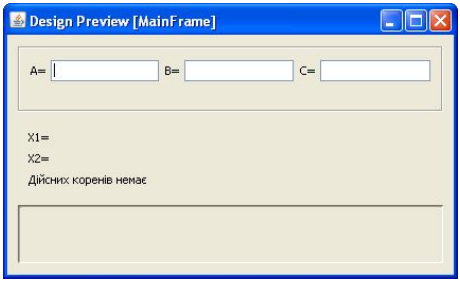
5. За допомогою компонентів JPanel розділяємо вікно на три горизонтальні області – панелі. При розміщенні панелей користуємося направляючими. Для верхньої панелі встановлюємо властивість Border у значення EtchedBorder, для середньої – залишаємо як є, а для нижньої обираємо SoftBevelBorder та у додатковому вікні тип границі – Lowered. Для всіх панелей у меню, що з'являється при натисканні правої кнопки миші обираємо "Auto Resizing ->Horizontal", а для середньої ще і "Vertical".



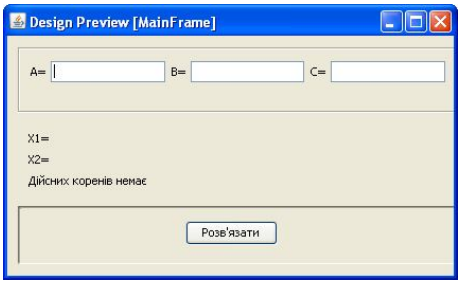
6. Розміщуємо на верхній панелі компоненти для введення коефіцієнтів A, B, C квадратного рівняння (JTextField), для них у вікні "Properties" очистити значення властивості "Text", та потім, за допомогою "Миші" відкоригувати їхні розміри у вікні так, щоб отримати результат, подібний до зображеного нижче. Крім того, оберіть ці компоненти та за допомогою правої кнопки миші у меню, що з'являється оберіть Auto Resizing -> Horisontal. Випробуйте це вікно, натиснувши кнопку "Preview Design".



7. Розміщуємо на верхній панелі компоненти JLabel, міняємо їм властивість "Text" так, щоб вони підказували де вводити відповідні коефіцієнти, та на середній панелі такі же компоненти для відображення результатів. Коригуємо розміри панелей так, щоб в них не залишалось "зайвого порожнього місця". Переглядаємо результат за допомогою "Preview Design":



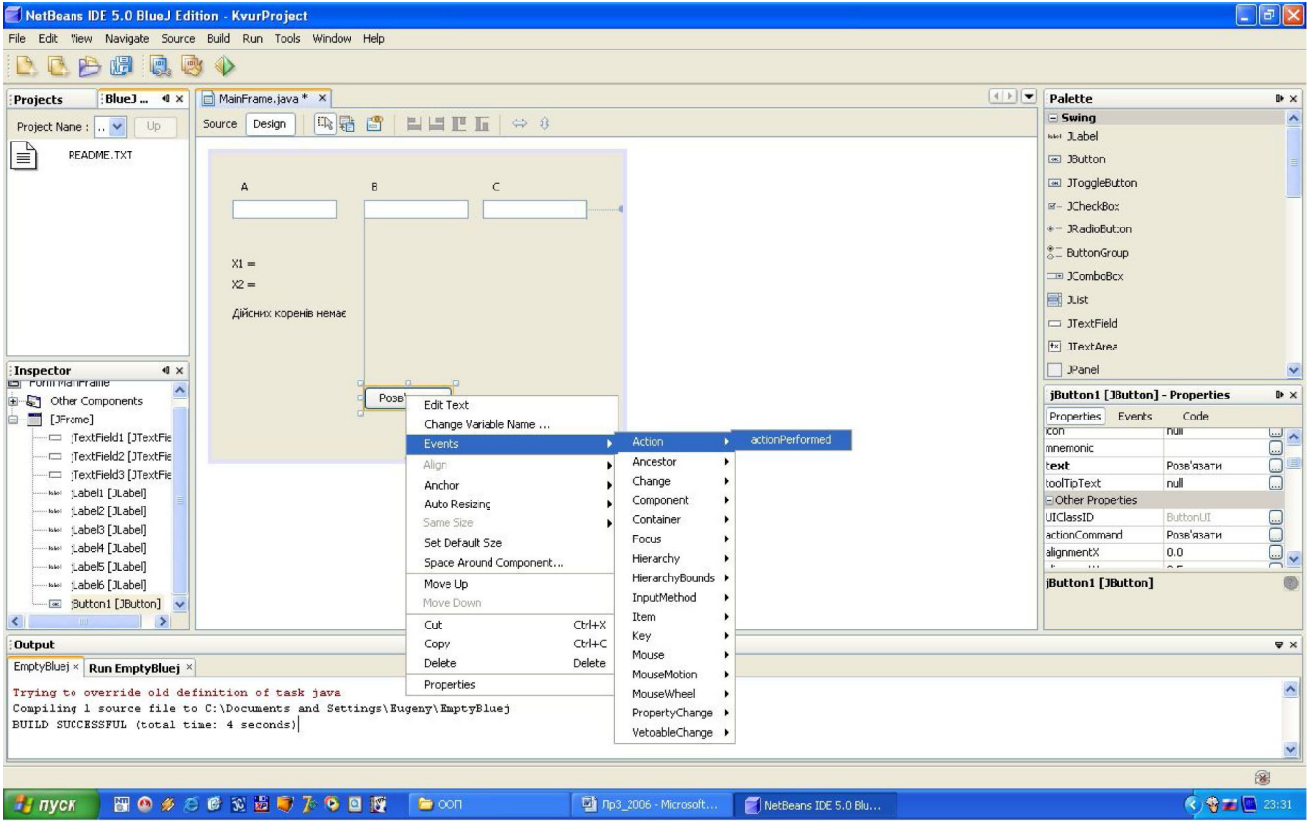
8. Додаємо до нижньої панелі вікна кнопку (JButton) та змінюємо напис на ній. Далі, за допомогою правої кнопки миші у меню, що з'являється оберіть Auto Resizing -> Horizontal. Випробуйте це вікно, натиснувши кнопку "Preview Design".. :



9. Через контекстне меню кнопки JButton (права кнопка "миші") обираємо Events -> Action -> actionPerformed:

10. У редакторі коду описуємо поведінку програми при натисканні цієї кнопки – розв'язання задачі:

```
   private void jButton1 Action Performed
(java.awt.event. ActionEvent evt) {
              /* отримуємо значення коеффіцієнтів
рівняння, введені користувачем */
   double a = Double.parseDouble(jTextField1.
getText());
   double b = Double.parseDouble(jTextField2.
getText());
   double c = Double.parseDouble(jTextField3.
getText());

   /* обчислюємо дискримінант */
   double d = b * b - 4 * a * c;

     /* перевіряємо дискримінант на від'ємність */
   if (d<0) {
     /* дійсних коренів немає - мітки для їхнього
відображення невидимі */
                jLabel4.setVisible(false);
                jLabel5.setVisible(false);
   /* робимо видимим напис "Дійсних коренів немає" */
```


NetBeans IDE 5.0 BlueJ Edition - KvurProject
File Edit View Navigate Source Build Run Tools Window Help
Projects
BlueJ ...
Project Name :
Up
README.TXT
MainFrame.java *
Source
Design
A
B
C
X1 =
X2 =
Дійсних коренів немає
Pose
Edit Text
Change Variable Name ...
Events
Align
Anchor
Auto Resizing
Same Size
Set Default Size
Space Around Component...
Move Up
Move Down
Cut Ctrl+X
Copy Ctrl+C
Delete Delete
Properties
Action
Ancestor
Change
Component
Container
Focus
Hierarchy
HierarchyBounds
InputMethod
Item
Key
Mouse
MouseMotion
MouseWheel
PropertyChange
VetoableChange
actionPerformed
Palette
Swing
JLabel
JButton
JToggleButton
JCheckBox
JRadioButton
ButtonGroup
JComboBox
JList
JTextField
JPanel
Inspector
Other Components
[JFrame]
Label1 [JLabel]
Label2 [JLabel]
Label3 [JLabel]
Label4 [JLabel]
Label5 [JLabel]
Label6 [JLabel]
Button1 [JButton]
jButton1 [JButton] - Properties
Properties
Events
Code
mnemonic
text
Розв'язати
toolTipText
null
Other Properties
UIClassID
ButtonUI
actionCommand
Розв'язати
alignmentX
0.0
jButton1 [JButton]
Output
EmptyBluej
Run EmptyBluej
Trying to override old definition of task java
Compiling 1 source file to C:\Documents and Settings\Eugeny\EmptyBluej
BUILD SUCCESSFUL (total time: 4 seconds)
пуск
ООП
Лр3_2006 - Microsoft...
NetBeans IDE 5.0 Blu...
23:31

```
                jLabel6.setVisible(true);
            } else {
                /* є дійсні корені – обчислюємо їх і
показуємо */
                double x1 = (-b-Math.sqrt(d))/(2*a);
                double x2 = (-b+Math.sqrt(d))/(2*a);
                jLabel4.setText(String.format("X1 =
%6.2f",x1));
                jLabel5.setText(String.format("X2 =
%6.2f",x2));
                jLabel4.setVisible(true);
                jLabel5.setVisible(true);
        /* прибираємо напис "Дійсних коренів немає",
оскільки вони є :) */
                jLabel6.setVisible(false);
            }
        }
```

11. Запускаємо програму на виконання, та виконуємо її тестування.

Лабораторна робота № 5

Робота з файлами. Застосування колекцій.

Розробити програму, що матиме зручний інтерфейс користувача і дозволятиме виконати дії відповідно до варіанту.

Примітка. *У завданнях усіх варіантів передбачити можливість програмного створення файлу даних, перегляду всіх даних у файлі, додавання елементу даних у файл та виконання запиту вказаного в умові задачі. Для тимчасового збереження інформації у оперативній пам'яті використовувати колекції.*

Короткі теоретичні відомості

В Java широко використовуються колекції (Collections) – "розумні" масиви з довжиною, що визначається динамічно, які підтримують ряд важливих додаткових операцій у порівнянні з масивами. Базовим для ієрархії колекцій є клас java.util.AbstractCollection. (У загальному випадку клас колекції не повинен бути наслідником AbstractCollection – він може бути будь-яким класом, що реалізує інтерфейс Collection).

Основні класи колекцій:

- Set, SortedSet, HashSet,TreeSet – множини (набори елементів, що не повторюються)
- List, ArrayList,LinkedList,Vector – списки (впорядковані набори елементів, які можуть повторюватися в різних місцях списку)
- Map, Sorted Map – таблиці (списки пар "ім'я" – "значення")

Доступ до елементів колекції в загальному випадку не може здійснюватися за індексом, через те що не всі колекції підтримують індексацію елементів. Цю функцію здійснюють за допомогою спеціального об'єкта – ітератора (Iterator). У кожної колекції collection є свій ітератор, який вміє з нею працювати, тому ітератор вводять таким чином:

```
Iterator iter = collection.iterator()
```

У ітераторів є такі три методи:

`boolean hasNext()` – надає інформацію, чи є колекції наступний об'єкт.

`Object next()` – повертає посилання на наступний об'єкт колекції.

`void` `remove()` – вилучає з колекції поточний об'єкт, тобто той, посилання на який було отримано останнім викликом `next()`.

Приклад перетворення масиву в колекцію та цикл з доступом до елементів цієї колекції, що здійснюється за допомогою ітератора:

```
java.util.List components=
java.util.Arrays.asList(this.getComponents());
for (Iterator iter = components. iterator();
iter.hasNext();) {
Object elem = (Object) iter.next();
javax.swing.JOptionPane.showMessageDialog(null,"Компонент:
"+elem.toString());
}
```

Основні методи колекцій:

Ім'я методу	Дія
`boolean add(Object obj)`	Додавання об'єкта в колекцію (в кінець списку). Повертає **true** у випадку успішного додавання – змінення колекції. Колекція може не дозволити додавання елементів несумісного типу чи таких що не підходять за будь-якою іншою ознакою.
`boolean addAll(Collection c)`	Додавання в колекцію всіх об'єктів з іншої колекції. Повертає **true** у випадку успішного додавання, тобто якщо додано хоча б один елемент.
`void clear()`	Очистка колекції – видалення з неї посилань на всі елементи, що входять в колекцію. При цьому ті об'єкти, на які є посилання у інших елементів програми, не видаляються з пам'яті.
`boolean contains(Object obj)`	Повертає **true** у випадку, якщо колекція містить об'єкт obj. Перевірка здійснюється за допомогою послідовного виклику метода obj.equals(e) для елементів e, що входять в колекцію.
`boolean containsAll(Collection c)`	Повертає **true** у випадку, якщо колекція містить всі елементи колекції c.
`boolean isEmpty()`	Повертає **true** у випадку, якщо колекція є порожньою, тобто не містить жодного елементу.

Ім'я методу	Дія
`Iterator iterator()`	Повертає посилання на ітератор – об'єкт, що дозволяє отримувати послідовний доступ до елементів колекції. Для однієї колекції дозволено мати довільну кількість об'єктів-ітераторів, в тому числі – різних типів. В процесі роботи вони можуть вказувати на різні елементи колекції. Після створення ітератор завжди вказує на початок колекції – виклик його метода **next()** дає посилання на початковий елемент колекції.
`boolean remove(Object obj)`	Вилучає з колекції перше, що зустрілося, входження об'єкта obj. Пошук та видалення здійснюється за допомогою ітератора. Повертає **true** у випадку, якщо видалення вдалося, тобто якщо колекція змінилась.
`boolean removeAll(Collection c)`	Вилучає з колекції всі елементи колекції c. Повертає true у випадку, якщо вилучення відбулося, тобто якщо колекція змінилась.
`Boolean retainAll(Collection c)`	Залишає в колекції тільки ті з елементів, що входять до неї, які входять в колекцію c.
`int size()`	Повертає число елементів в колекції.
`Object[] toArray()`	Повертає масив посилань на об'єкти, що містяться в колекції. Тобто перетворює колекцію в масив.
`T[] toArray(T[n] a)`	Повертає масив елементів типу T, які будуть отримані в результаті перетворення елементів, що містяться в колекції. Тобто перетворює колекцію в масив. Якщо кількість елементів колекції не перевищує розмір n масиву a, розміщення даних виконується в існуючих комірках пам'яті, відведених для масиву. Якщо перевищує n – в пам'яті динамічно створюється та заповнюється новий набір комірок, та їхня кількість буде рівною числу елементів колекції. Після чого змінна a починає посилатися на новий набір комірок.
`String toString()`	Метод перевизначен – він повертає рядок зі списком елементів колекції. В списку виводяться рядкові представлення елементів, які взяті у квадратні дужки, які розділяються комбінацією ", " – кома з пропуском після неї.

Найпоширенішими варіантами колекції є списки (Lists). Вони багато у чому схожі на масиви, але відрізняються від масивів тим, що в списках основними операціями є додавання та вилучення елементів, а не доступ до елементів за індексом, як в масивах.

В класі List є методи колекції, а також ряд додаткових методів:

list.get(i) – отримання посилання на елемент списку list за індексом i.

list.indexOf(obj) – отримання індексу елемента obj у списку list. Повертає –1 якщо об'єкт не знайдено.

list.listIterator(i) – отримання посилання на ітератор типу ListIterator, що має додаткові методи у порівнянні з ітераторами типу Iterator.

list.listIterator(i) – теж саме з позиціонуванням ітератора на елемент з індексом i.

list.remove(i) – вилучення зі списку елемента з індексом i.

list.set(i,obj) – заміна в списку елемента з індексом i на об'єкт obj.

list.subList(i1,i2) – повертає посилання на підсписок, що складається з елементів списку з індексами від i1 до i2.

Крім них в класі List є ще багато інших корисних методів.

Ряд корисних методів для роботи з колекціями міститься в класі Collections:

Collections.addAll(c,e1,e2,…,eN) – додавання в колекцію c довільної кількості елементів e1,e2,…,eN.

Collections.frequency(c,obj) – повертає кількість входжень елемента obj в колекцію c.

Collections.reverse(list) – перевертає порядок слідування елементів в списку list (перші стають останніми і навпаки).

Collections.sort(list) – сортує список в порядку зростання елементів. Порівняння іде викликом методу e1.compareTo(e2) для чергових елементів списку e1 та e2.

Крім них в класі Collections є ще багато інших корисних методів.

В класі Arrays є метод

Arrays.asList(a) – повертає посилання на список елементів типу T, що є "обгорткою" над масивом T[] a. При цьому і масив, і список містять ті ж самі елементи, і зміна елемента списку призводить до зміни елемента масиву, и навпаки.

Робота з файлами

• Об'єкти типу File забезпечують роботу з іменами файлів та папок (перевірка існування файлу чи папки з вказаним іменем, знаходження абсолютного шляху за відносним і навпаки, перевірка та встановлення атрибутів файлів та папок).

• При роботі з файлами в більшості програм потрібен виклик файлового діалогу JFileChooser (пакет javax.swing).

• Потоки введення-виведення забезпечують роботу не тільки з файлами, але і з пам'яттю, а також різноманітними пристроями введення-виведення. Відповідні класи розміщені в пакеті java.io. Абстрактний клас InputStream ("вхідний потік") інкапсулює модель вхідних потоків, що дозволяють зчитувати з них дані. Абстрактний клас OutputStream ("вихідний потік") – модель вихідних потоків, що дозволяють записувати до них дані.

• Для читання рядків (у вигляді масиву символів) використовуються нащадки абстрактного класу Reader ("читач"). Наприклад, для читання з файлу – клас FileReader. Аналогічно, для запису рядків використовуються класи, які наслідуються від абстрактного класу Writer ("записувач"). Наприклад, для запису масиву символів в файл – клас FileWriter.

• Є ще один важливий клас для роботи з файлами – RandomAccessFile ("файл з довільним доступом"), який призначений для читання і запису даних у довільному місці файлу. Такий файл з точки зору класу RandomAccessFile представляє масив байт, які збережені на зовнішньому носії. Клас RandomAccessFile не є абстрактним, тому можна створювати його екземпляри.

Приклад використання класів для введення даних:

```
try {
    BufferedReader in = new BufferedReader(
            new FileReader("file.txt"));
    String line = null;
    while ((line = in.readLine())!=null){
        System.out.println(line);
    }
    in.close();
} catch (IOException ex) {
    ex.printStackTrace();
}
```

В цьому прикладі вміст файлу file.txt роздруковується на екрані.

Якщо рядок, що прочитаний з файлу містить декілька елементів, то для розділення їх можна скористатися класами розбору рядків Scanner або StringTokenizer, наприклад:

```
Scanner s = new Scanner(line);
int n = s.nextInt();
String str = s.next();
double x = s.nextDouble();
```

або

```
StringTokenizer st = new StringTokenizer(line);
int n = Integer.parseInt(st.nextToken());
String str = st.nextToken();
double x = Double.parseDouble(st.nextToken());
```

Дано файл, що містить відомості про іграшки, указується назва іграшки (лялька, кубики, м'яч, конструктор), її вартість у копійках і вікові межі дітей, для яких іграшка призначена (наприклад, для дітей від двох до п'яти років). Крім того, для ляльки зазначено її розмір у сантиметрах, для кубиків – їхня кількість у наборі, для м'яча – його вага у грамах, для конструктора – кількість конструкцій, які з нього можна збудувати згідно інструкції. Одержати наступні відомості:

Варіанти завдання

Варіант 1.

Перелік іграшок, ціна яких не перевищує вказану і які підходять дітям 5 років у порядку зростання ціни

Варіант 2.

Перелік конструкторів у порядку зростання ціни. (Ціну виводити за зразком …грн…коп).

Варіант 3.

Перелік найбільш коштовних іграшок (ціна яких відрізняється від ціни найкоштовнішої іграшки не більш ніж на 10 грн.) у порядку спадання ціни.

Варіант 4.

Перелік іграшок, що підходять дітям від 4 років до 10 років. Виводити в порядку алфавіту

Варіант 5.

Перелік усіх кубиків, у порядку зростання цін. Ціну виводити за зразком …грн…коп.

Варіант 6.

Вивести перелік іграшок, будь яких, крім м'яча, що підходять дитині 3 років, щоб вартість іграшки не перевищувала задану суму. Перелік виводити у порядку спадання ціни.

Варіант 7.

Вивести перелік м'ячів із ціною, що вказана, або менша за неї, призначених дітям від 3 до 8 років. Перелік виводити у порядку зростання ваги м'ячів

Варіант 8.

Вивести перелік ляльок, що підходять дитині 3 років, щоб розмір іграшки не перевищував заданий. Перелік виводити у порядку збільшення розміру ляльки

Варіант 9.

Вивести перелік кубиків, що підходять дитині 2 років, щоб їх вартість перевищувала задану суму, але була не більше 200 грн. Перелік виводити у порядку зростання цін.

Варіант 10.

Перелік найбільш дешевих іграшок (ціна яких відрізняється від ціни найдешевшої іграшки не більш ніж на 10 % її вартості) у порядку зростання ціни.

Лабораторна робота № 6

Потоки виконання (threads) і синхронізація.
Короткі теоретичні відомості

Клас Thread і інтерфейс Runnable

Є два способи створити клас, екземплярами якого будуть потоки виконання: успадкувати клас від java.lang.Thread або реалізувати інтерфейс java.lang.Runnable. Цей інтерфейс має декларацію єдиного методу public void run(), який забезпечує послідовність дій при роботі потоку. При цьому клас Thread вже реалізує інтерфейс Runnable, але з порожньою реалізацією методу run(). Так що при створенні екземпляра Thread створюється потік, який нічого не робить. Тому в нащадку треба перевизначити метод run(). У нім слід написати реалізацію алгоритмів, які повинні виконуватися в даному потоці. Відзначимо, що після виконання методу run() потік припиняє існування – "вмирає".

Розглянемо перший варіант, коли ми успадковуємо клас від класу Thread, перевизначивши метод run().

Об'єкт-потік створюється за допомогою конструктора. Є декілька перевантажених варіантів конструкторів, найпростіший з них – з порожнім списком параметрів. Наприклад, в класі Thread їх заголовки виглядають так:

public Thread() – конструктор за умовчанням. Підпроцес отримує ім'я "system".

public Thread(String name) – потік отримує ім'я, що міститься в рядку name.

У класі-нащадку можна викликати конструктор за замовчуванням (без параметрів), або задати свої конструктори, використовуючи виклики конструкторів батьківських класів за допомогою виклику super(список параметрів). Із-за відсутності спадкування конструкторів в Java доводиться в спадкоємцеві заново задавати конструктори з тією ж сигнатурою, що і в класі Thread. Це є простою, але утомливою роботою. Саме тому зазвичай віддають перевагу способу завдання класу з реалізацією інтерфейсу Runnable, про що буде розказано далі.

Створення та запуск потоку здійснюється наступним чином:

```
public class T1 extends Thread{
   public void run(){
      ...
   }
```

```
    ...
}
Thread thread1 = new T1();
thread1.start();
```

Другий варіант – використання класу, в якому реалізований інтерфейс java.lang.Runnable. Цей інтерфейс, як вже говорилося, має єдиний метод public void run(). Реалізувавши його в класі, можна створити потік за допомогою перевантаженого варіанту конструктора Thread:

```
public class R1 implements Runnable{
   public void run(){
      ...
   }
   ...
}
Thread thread1 = Thread( new R1() );
thread1.start();
```

Зазвичай в такий спосіб користуються набагато частіше, оскільки в класі, що розробляється, не доводиться займатися дублюванням конструкторів класу Thread. Крім того, цей спосіб можна застосовувати у разі, коли вже є клас, що належить ієрархії, в якій базовим класом не є Thread або його спадкоємець, і ми хочемо використовувати цей клас для роботи усередині потоку. В результаті від цього класу ми отримуємо метод run(), у якому реалізований потрібний алгоритм, і цей метод працює усередині потоку типа Thread, що забезпечує необхідну поведінку в багатопотоковому середовищі. Проте в даному випадку ускладнюється доступ до методів з класу Thread – потрібне приведення типа.

Поля та методи, які задані в класі Thread

В класі Thread маємо декілька полів даних та методів, про які треба знати для роботи з потоками.

Найважніші константи та методи класа Thread:

- MIN_PRIORITY – мінімально можливий пріоритет потоків. (як описано у довідковій системі jdk, дорівнює 1)
- NORM_PRIORITY – нормальний пріоритет потоків. Головний потік створюється з нормальним пріоритетом, а потім пріоритет може бути змінено. (дорівнює 5)
- MAX_PRIORITY – максимально можливий пріоритет потоків. (як описано у довідковій системі jdk, дорівнює 10)
- static int activeCount() – вертає число активних потоків застосунку.

• static Thread currentThread() – вертає посилання на поточний потік.

• static boolean interrupted() – вертає стан статуса переривання поточного потоку, після чого встановлює його у значення false.

Найважливіші методи об'єктів типу Thread:

• void run() – метод, який забезпечує послідовність дій під час життя потоку. У класі Thread задана його порожня реалізація, тому в класі потоку він має бути перевизначений. Після виконання методу run() поток вмирає.

• void start() – викликає виконання поточного потоку, у тому числі запуск його методу run() у потрібному контексті. Може бути викликаний всього один раз.

• void setDaemon(boolean on) – у випадкуе on==true встановлює потоку статус демона, інакше – статус користувальницького потоку.

• boolean isDaemon() – вертає true у випадку, коли поточний потік є демоном.

• void yield() – "поступитися правами" – викликає тимчасове призупинення потоку, з передаванням прав іншим потокам виконувати необхідні для них дії.

• long getId() – повертає унікальний ідентифікатор потоку. Унікальність відноситься лише до часу життя потоку – після його завершення (смерті) даний ідентифікатор може бути привласнений іншому створюваному потоку.

• String getName() – повертає ім'я потоку, яке йому було задано при створенні або методом setName.

• void setName(String name) – встановлює нове ім'я потоку.

• int getPriority() – вертає пріоритет потоку.

• void setPriority(int newPriority) – встановлює пріоритет потоку.

• void checkAccess() – здійснення перевірки з поточного потоку на дозволеність доступу до іншого потоку. Якщо потік, з якого йде виклик, має право на доступ, метод не робить нічого, інакше – збуджує виключення SecurityException.

• String toString() – вертає рядкове представлення об'єкта потока, в тому числі – його им'я, групу, пріоритет.

• void sleep(long millis) – викликає призупинення ("засинання") потоку на millis мілісекунд. При цьому всі блокування (монітори) потоку зберігаються. Перевантажений варіант sleep(long millis,int nanos) – параметр nanos вказує число наносекунд. Дострокове пробудження здійснюється методом interrupt() – зі збудженням виключення InterruptedException.

• void interrupt() – переривает "сон" потоку, спричинений викликами wait(...) або sleep(...), встановлюючи йому статус перерваного (статус переривання=true). При цьому збуджується перевіряєма виключна ситуація InterruptedException.

• boolean isInterrupted() – вертає поточний стан статусу переривання потоку без зміни значення статусу.

• void join() – "злиття". Переводить потік в режим вмирання – очікування завершення. Це очікування – виконання метода join() – може відбуватися достатньо довго, якщо відповідний потік на момент виклику метода join() блокований. Тобто якщо в ньому виконується синхронізований метод або він очікує завершення синхронізованого метода. Перевантажений варіант join(long millis) – очікувати завершення потока протягом millis милісекунд. Виклик join(0) еквівалентний виклику join(). Зазвичай використовується для завершення головним потоком роботи всіх дочірніх потоків ("злиття" їх з головним потоком).

• boolean isAlive() – вертає true у випадку, коли поточний потік живий (ще не вмер). Відмітимо, що навіть якщо потік завершився, від нього залишається об'єкт – "привид", який відповідає на запит isAlive() значенням false – тобто повідомляє, що об'єкт вмер.

Крім того, слід знати, що в класі Object визначені ще деякі методи, які можуть бути корисними для написання багатопотокової програми:

• public final void wait() throws InterruptedException – переводить потік в режим очікування доти, поки інший потік не викликає метод notify() або notyfyAll() для цього потоку

• public final void wait(long timeout) throws InterruptedException – те ж саме, що і попередній, але виконання буде автоматично продовжено після timeout мілісекунд

• public final void wait(long timeout, int nanos) throws InterruptedException – те ж саме, що і попередній, але виконання буде автоматично продовжено після timeout мілісекунд та nanos наносекунд

• public final void notify() – пробуджує потік, що перебуває в стані очікування після виклику wait()

• public final void notifyAll() – пробуджує всі потоки, що перебувають в очікування поточного монітору.

Слід зазначити, що всі провідні розробники процесорів перейшли на багатоядерну технологію. У зв'язку з цим, програмування в багатопотоковому середовищі визнане найбільш перспективною моделлю паралелі-

зування програм і стає одним з найважливіших напрямів розвитку програмних технологій. Модель багатопоточності Java дозволяє вельми елегантно реалізувати переваги багатоядерних процесорних систем. У багатьох випадках програми Java, написані з використанням багатопоточності, ефективно розпаралелюються автоматично на рівні віртуальної машини – без зміни не лише вихідного, але навіть скомпільованого байт-коду програмного продукту.

Завдання.

Обчислити значення визначеного інтеграла відповідно до варіанту. Реалізацію програми виконувати таким чином:

1. Створити клас "Функція" (з єдиним методом "обчислити") для реалізації підінтегральної функції.

2. Створити клас "Обчислювач інтегралів", який може працювати у багатопотоковому режимі і має метод "обчислити" з параметрами: a, b – кінці інтервалу, n – кількість кроків та f – підінтегральна функція.

3. Для цих класів розробити модульні тести і виконати тестування

4. Створити віконну програму, яка буде дозволяти вводити кількість інтервалів розбиття відрізку інтегрування і кількість потоків виконання.

5. Як результати роботи програми вивести обчислене значення інтегралу і час, який знадобився для її виконання.

6. Виконати обчислення декілька разів для різних (від 1 до 20 кількостей потоків виконання) при малій (менше 103) та великій (більше 106) кількості інтервалів розбиття відрізка.

7. Зробити висновки

Примітка. *Формули для обчислення визначеного інтеграла наближеними методами наведено нижче:*

- *Метод лівих прямокутників* $\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} f(x_i)h$
- *Метод правих прямокутників* $\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(x_i)h$
- *Метод середніх прямокутників* $\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} f(x_i + h/2)h$

- *Метод трапецій* $\int_a^b f(x)dx \approx \frac{f(x_0)+f(x_n)}{2}h + \sum_{i=1}^{n-1} f(x_i)h$

- *Метод Сімпсона* $\int_a^b f(x)dx \approx$

$$\approx \frac{h}{3}\left(\frac{1}{2}f(x_0)+\sum_{i=1}^{n-1} f(x_i)+2\sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right)+\frac{1}{2}f(x_n)\right)$$

- *в усіх методах* $h=\frac{b-a}{n},\ x_i = a+i\cdot h$

Варіанти завдань

№ варіанта	Інтеграл	Метод
1	$\int_1^9 3\sqrt{t}dt$	Метод лівих прямокутників
2	$\int_1^4 \frac{1+t}{\sqrt{2t}}dt$	Метод трапецій
3	$\int_1^9 3\sqrt{x}(1+\sqrt{x})dx$	Метод Сімпсона
4	$\int_0^1 \ln(t+1)dt$	Метод правих прямокутників
5	$\int_1^{\sqrt{2}} \sqrt{2-x^2}dx$	Метод середніх прямокутників
6	$\int_1^2 \frac{e^x-1}{e^x+1}dx$	Метод Сімпсона
7	$\int_0^{\pi/2} \frac{dt}{\sin^2(2t)}$	Метод лівих прямокутників
8	$\int_0^1 e^t\sqrt{1-e^t}\,dt$	Метод правих прямокутників
9	$\int_0^{\pi/3} \cos(4t)\cos(2t)dt$	Метод трапецій
10	$\int_0^{\pi/2} \sin(2t)\cos(3t)dx$	Метод середніх прямокутників

ЗМІСТ