МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних завдань

з навчальної дисципліни «Програмування на Java»
для студентів денної та заочної форм навчання
за спеціальностями «122 Комп'ютерні науки», та
«113 Прикладна математика»

Затверджено
редакційно-видавничою
радою університету,
протокол № 3 від 24.10.2024 р.

Харків
НТУ «ХПІ»
2024

Методичні вказівки до виконання лабораторних завдань з навчальної дисципліни «Програмування на Java» для студентів денної та заочної форм навчання за спеціальностями «122 Комп'ютерні науки», та «113 Прикладна математика»/ уклад.: М. І. Шаповалова. – Харків: НТУ «ХПІ», 2024. – 67 с.

Укладач: М. І. Шаповалова

Рецензент: С. Ю. Місюра

Кафедра математичного моделювання на інтелектуальних обчислень в інженерії.

ЗМІСТ

ВСТУП

Методичні вказівки з курсу програмування мовою Java для студентів спеціальностей 122 «Комп'ютерні науки» та 113 «Прикладна математика» спрямовані на формування у студентів фундаментальних знань з об'єктно-орієнтованого програмування та розвиток практичних навичок у розробці програмного забезпечення. Лабораторні роботи дозволяють поступово освоювати ключові можливості мови Java, починаючи з простих програм на зразок "Hello World" і переходячи до складніших завдань, таких як робота з колекціями, обробка рядків і використання регулярних виразів. Цей підхід дозволяє студентам крок за кроком занурюватися в мову програмування та її можливості.

На початкових етапах студенти знайомляться з базовими елементами синтаксису Java, а також із концепціями об'єктно-орієнтованого програмування, такими як спадкування та поліморфізм. Поступово робота ускладнюється через завдання, які передбачають розробку консольних додатків та освоєння графічного інтерфейсу на основі JavaFX. Це допомагає студентам зрозуміти, як можна створювати більш масштабовані та функціональні програми, використовуючи ці підходи.

Додатково, у рамках лабораторних робіт студенти вивчають принципи створення динамічних веб-додатків із застосуванням сервлетів, що є важливим етапом для розуміння взаємодії між сервером і клієнтом у веб-розробці. Завдяки цьому курсу студенти отримують міцний фундамент для подальшого розвитку своїх навичок програмування та успішного застосування їх у професійній діяльності.

Лабораторна робота №1
ЗНАЙОМСТВО З JAVA
«HELLO WORLD»

Мета роботи:

Встановити програму Eclipse IDE (чи будь яку іншу IDE). Створити першу програму на мові Java. Ознайомитись із поняттями змінні та типи. Виконати завдання з використанням умовних операторів.

Вказівки до роботи:

1. Установка. Завантажте Eclipse IDE. Посилання та інструкції див. нижче:

- https://www.eclipse.org/
- Get Eclipse IDE 201812 | Install your favorite desktop IDE package |.Download 64 bit
- Download from: United States - OSU Open Source Lab (http)
- File: eclipse-inst-win64.exe SHA-512

2. Запустіть завантажений файл (відкриття установника може зайняти пару хвилин).

3. Виберіть місце на диску, куди необхідно встановити програму -> Install (Рис. 1).

4. Залишайте всі галочки і натисніть на «Accept» (Рис. 2). Встановлення розпочато.

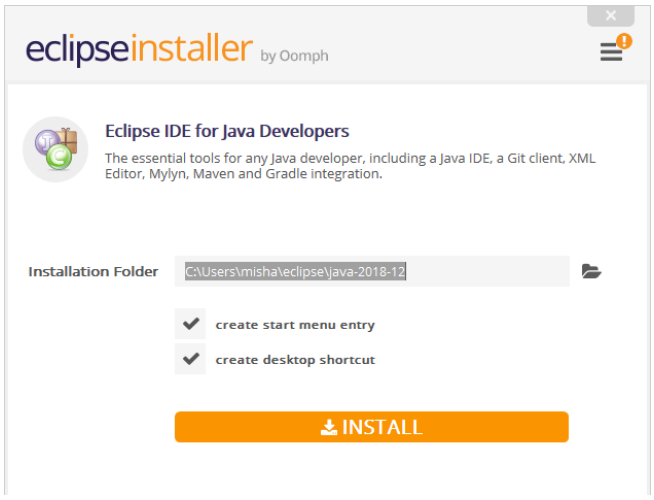

Рисунок 1 – Встановлення Eclipse IDE. Вибір розташування.

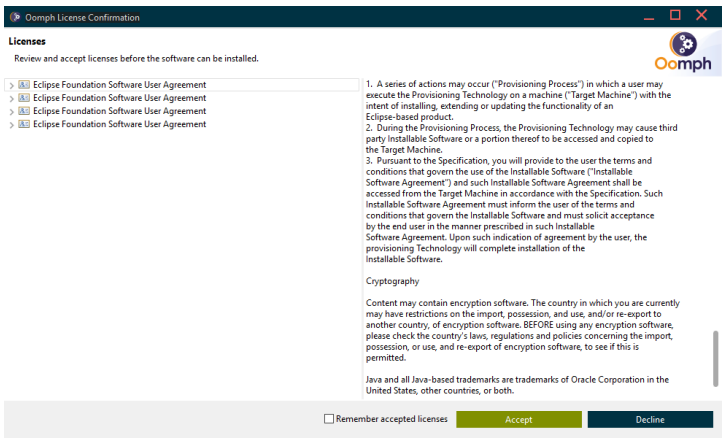

Рисунок 2 – Встановлення ліцензії Eclipse IDE.

5. Далі програма попросить вибрати компоненти. Виберіть всі «Select all» і натисніть «Accept selected» (Рис. 3).

6. Натисніть «Browse». Виберіть розташування робочої папки, де будуть зберігатись робочі проекти, далі натисніть «Launch».

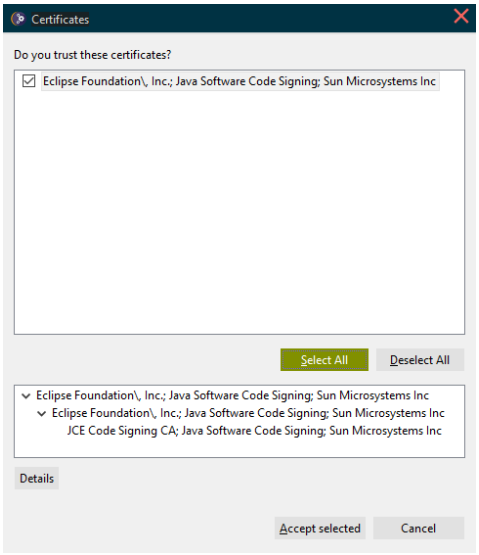

Рисунок 3 – Встановлення Eclipse IDE. Вибір компонентів.

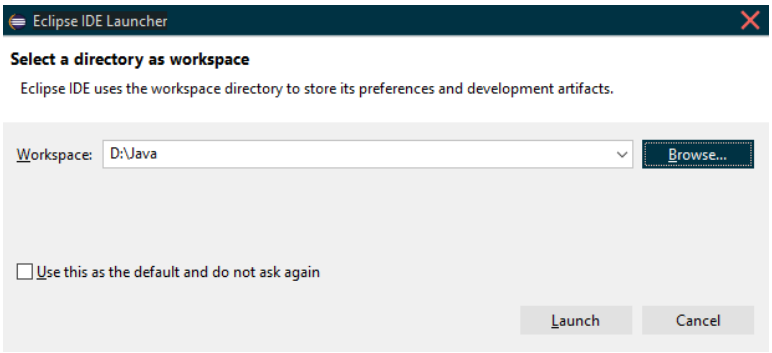

Рисунок 4 – Встановлення Eclipse IDE. Вибір розташування робочої папки.

Перша програма на Java:

File | New | Java Project. Вкажіть ім'я проекту (FirstLab). Потім Finish (Рис. 5).

Далі напискаємо правою кнопкою миші на назву «FirstLab» в лівому вікні. Створюємо новий клас: New | Class. У вікні вказуємо ім'я класу: «helloWorld» (Рис. 6).

Прописуємо тіло класу:

```
public class HelloWorld {
 public static void main(String[] args) {
 System.out.println("Hello world");
 }
}
```

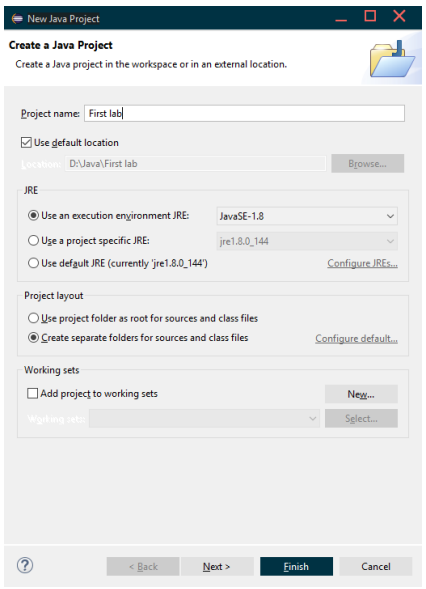

Рисунок 5 – Перша програма на Java.

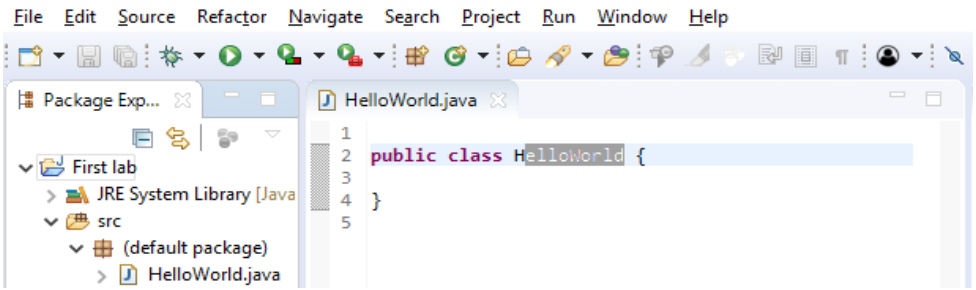

Рисунок 6 – Інтерфейс робочої області Eclipse IDE.

Запускаємо програму на виконання. У верхньому меню «Run» або (Ctrl + F11). У нижньому вікні повинно з'явитися наші пропозиції «Hello world». Вітаємо! Перша програма готова.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Примітивні типи в Java.

Незважаючи на те, що мова Java об'єктно-орієнтована, не всі типи - об'єкти. Існують звані примітиви (primitives). Ось список всіх примітивів у Java:

- byte (число, 1 байт)
- short (число, 2 байти)
- int (число, 4 байти)
- long (число, 8 байтів)
- float (число з плаваючою точкою, 4 байти)
- double (число з плаваючою точкою, 8 байтів)
- char (символ, 2 байти)
- boolean (true (істина) або false (брехня), 1 байт)

Java – строго типізована мова, це означає, що повинні оголосити змінні, перш ніж використовувати їх.

Числа Java.

Щоб оголосити та присвоїти число, використовуйте наступний синтаксис, де = це оператор присвоєння:

```
int myNumber;
myNumber = 5;
```

Можна об'єднати ці операції:

```
int myNumber = 5;
```

Щоб оголосити число з плаваючою точкою, використовуйте наступний синтаксис:

```
double d = 4.5;
d = 3.0;
```

Якщо Ви хочете використовувати **float**, то:

```
float f = (float) 4.5;
Або:
float f = 4.5f
```

(**f** – більш короткий спосіб оголосити float)

Символи та рядки в Java.

Java символ – свій власний тип, і це не просто число. Синтаксис:

```
char c = 'g';
```

String – не примітив. Це справжній тип. Ось кілька способів використання рядка:

- Створення рядка за допомогою конструктора:

```
String s1 = new String("Who let the dogs out?");
```

- За допомогою подвійних лапок (" ").

```
String s2 = "Who who who who!";
```

У Java є конкатенація (об'єднання) рядків за допомогою оператора +.

```
String s3 = s1 + s2;
```

У Java оператор + визначено лише для рядків, ви ніколи не побачите його з іншими об'єктами, лише з примітивами.

```
int num = 5;
String s = "I have" + num + "cookies";
```

Зауважте, що лапки з примітивами не використовуються.

Логічний тип boolean в Java.

Кожен оператор порівняння Java повертає булеву змінну (boolean), яка може прийняти тільки два значення: **true** (істина) або **false** (брехня).

```
boolean b = false;
b = true;

boolean toBe = false;
b = toBe | !toBe;
if (b) {
     System.out.println(toBe);}
```

Оператор | це логічне «**чи**», оператор **!** – це логічне «**не**», **&** – це логічне «**і**»,

Наступний код *не працюватиме* через несумісність типів:

```
int children = 0;
b = children; // Не працюватиме, потрібно boolean, а знайдено
int
if (children) { // Не буде працювати, потрібно boolean, а
знайдено int
      // Не працюватиме, потрібно boolean, а знайдено int
}
```

Умовні оператори в Java.

Java використовує булеві (логічні) змінні, щоб оцінювати умови. Значення **true** або **false** повернеться після того, як вираз буде оцінений.

Наприклад:

```
int a = 4;
boolean b = a == 4;
if (b) {
      System.out.println("It's true!");}
```

Зазвичай не присвоюємо умовне вираження булевій змінній, а просто використовуємо коротку версію:

```
int a = 4;
if (a == 4) {
      System.out.println("Ohhh! So a is 4!");}
```

Логічні оператори.

Є не так багато операторів, які можна використовувати за умов. Можна видылити наступне:

```
int a = 4;
int b = 5;
boolean result;
result = a < b; // Істина result = a > b; // брехня
result = a <= 4; // менше чи одно - істина result = b >= 6;
// більше чи одно - брехня
result = a == b; // одно - брехня
result = a! = b; // нерівно – істина
result = a > b || a < b; // Логічне АБО - істина
result = 3 < a && a < 6; // Логічне І - істина
result = ! // Логічне НЕ – брехня
```

Оператор if – else.

Синтаксис оператора **if - else** досить простий:

```
if (a == b) {
     // Тіло методу. Виконується якщо a та b рівні.
}
```

Також можемо додати ще один вираз, на випадок, якщо умова не виконується:

```
if (a == b) {
     // Ми вже знаємо цю частину
} else {
     // a та b не рівні... :/
}
```

Якщо тіло методу можна розмістити в один рядок, можна не використовувати { }

```
if (a == b) System.out.println("Yeah!");
else System.out.println("Ohhh...");
Або

if (a == b)
     System.out.println("Another line Wow!");
else
     System.out.println("Double rainbow!");
```

Незважаючи на те, що такий метод міг би зробити ваш код коротшим, рекомендуємо не використовувати коротку версію умовного оператора, щоб не заплутатись напочатку.

Інша сторона if.

Є ще один спосіб записати **if – else** в один рядок – за допомогою оператора **?** (але це «advance» рівень, на перших кроках краще не використовувати:

```
int a = 4;
int result = a == 4? 1: 8;
// result дорівнюватиме 1
// Або звичайна форма запису:
int result;
if (a == 4) {
     result = 1;
} else {
```

```
        result = 8;
    }
```

Оператори == та equals.

Оператор працює трохи по-іншому на об'єктах, ніж на примітивах. Коли використовуєте об'єкти і хочете перевірити, чи вони рівні, оператор == скаже що вони рівні, тільки якщо об'єкти однакові, але якщо хочете перевірити їх на *логічну відповідність*, використовуйте метод **equals**.

Наприклад:

```
String a = new String("Wow");
String b = new String("Wow");
String sameA = a;

boolean r1 = a == b; // Брехня, тому що a і b не один і той же об'єкт
boolean r2 = a.equals(b); // Істина, оскільки a і b логічно рівні
boolean r3 = a == sameA; // Істина, тому що a і sameA дійсно один і той же об'єкт
```

Завдання до лабораторної роботи №1:

Завдання 1. Робота у консолі. Отримати від користувача числа та вивести на екран результат (обрати **два** варіанти на вибір, приклад класу дивитись нижче):

- вивести на екран подвоєнний добуток двох чисел збільшених на 1;
- перше число збільшити на 1, друге число зменшити на 1. Вивести на екран результат у вигляді «було-стало» для обох чисел;
- вивести на екран суму першого числа поділеного на третє, та другого числа помноженого на третє;
- вивести на екран частку від суму двох чисел поділену на їх різницю;
- вивести на екран квадратний корінь першого числа помножений на різницю першого і другого числа;
- вивести на екран різницю першого і другого числа підведену у степінь, що дорівнює третьому введеному числу;
- вивести на екран суму чисел першого, підведеного у степінь, що дорівнює третьому введеному числу, плюс друге число, поділене на третє введене число;
- вивести на екран суму двох чисел мінус їх різницю, підведену у степінь 3;

- вивести на екран корінь квадратний суми двох чисел мінус їх різницю;
- вивести на екран перше число помножене на різницю другого і третього, та вивести через знак тире друге число, як добуток першого і третього введеного користувачем числа.

Приклад робои з двома класами та вводом-виводом інформації за допомогою `Scanner`. Створити новий проект (LabaFirstExpression). У лівому меню з'явився проект. У ньому створити 2 клас «Expression» і «Reader».

```
public class Expression {
 public static void main(String[] args) {
 Reader r = new Reader();
 r.Scan();
 r.i = count(r.i);
 r.k = count(r.k);
 System.out.println(r.i);
 System.out.println(r.k);
 }
 private static int count(int x) {
 x = x + 1;
 return x;
 }
}
```

Другий клас:

```
import java.util.*;
public class Reader {
 int i;
 int k;
 public void Scan() {
 System.out.println("Введіть перше число:");
 Scanner scn1 = new Scanner(System.in);
 i = scn1.nextInt();
 System.out.println("Введіть друге число:");
 Scanner scn2 = new Scanner(System.in);
 k = scn2.nextInt();
 }
}
```

Завдання 2. Вивести на екран таблицю множення, в залежності від основи, яку ввів користувач (наприклад, таблицю множення на «3», якщо користувач ввів число «3»). Обмежити можливість введеня лише цифр від 1 до 9 за допомогою умовного оператора if – else.

Завдання 3. Створити програму, яка виводитиме на екран менше за модулем з трьох введених користувачем дійсних чисел.

Лабораторна робота №2
РОЗРОБКА КОНСОЛЬНИХ JAVA-ДОДАТКІВ

Мета роботи:

У лабораторній роботі розробляється консольний додаток для реалізації найпростішої програми з використанням масивів, рядків і файлів.

Вказівки до роботі:

Консольні програми у Java являють собою відкомпільований клас, що містить точку входу.

Розглянемо простий приклад:

```
public class First {
      public static void main(String[] args) {
      System.out.println ("Перша програма на Java!");
      }
}
```

Тут клас **First** використовується тільки для того, щоб визначити метод main(), котрий і є точкою входу і з якого починається виконання програми інтерпретатором Java. Метод main() містить аргументи-параметри командного рядка String[] args у вигляді масиву рядків і є відкритим (Public) членом класу. Це означає, що метод main() видимий і доступний будь-кому класу. Ключове слово *static* оголошує методи та змінні класу, що використовуються для роботи з класом в цілому, а не тільки з об'єктом класу. Символи верхнього і нижнього регістрів в Java — різняться.

Виведення рядка «Перша програма на Java!» у прикладі здійснює метод **println()** (**ln** — перехід до нового рядка після виведення) властивості *out* класу System, який доступний в програмі автоматично разом з пакетом java.lang. Наведену програму необхідно помістити у файл, ім'я якого збігається з ім'ям класу з розширенням .java. У разі успішної компіляції створюється файл First.class. Цей файл можна запустити на виконання з командного рядка за допомогою інтерпретатора Java наступним чином: java First

! Для розробки програми можливе використання спеціальних засобів розробника:

- **NetBeans IDE**
- **Eclipse IDE**
- **IntelliJ IDEA**
- **JDeveloper**
- **JBuilder**
- **BlueJ**

ТЕОРЕТИЧНІ ВІДОМОСТІ

Клас File.

Клас **java.io.File** — для роботи з файлами в програмах Java можуть бути використані класи з пакету java.io, одним із яких є клас File, що служить для зберігання та обробки як об'єктів каталогів так і імен файлів. Цей клас не визначає способи роботи з вмістом файлу, але дозволяє маніпулювати такими властивостями файлу, як права доступу, дата та час створення, шлях в ієрархії каталогів, створіння, видалення, зміна імені файлу і каталогу.

Каталог (директорія), як об'єкт класу File, має додаткову властивість — перегляд списку імен файлів з допомогою методів list(), listFiles(), listRoots().

Основні *методи класу File* і способи їх застосування:

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

public class Main {
public static void main(String[] args) throws IOException
/*відмова від обробки виключення в main()*/
{
//З об'єктом типу File асоціюється файл на диску.
//Способи створення об'єкту (прибрати "//" по черзі)
//File fp = new File( "com\learn\FileTest.java" );
//File fp = new File("\\com\\learn", "FileTest.java");
//File fp = new File("d:\temp\demo.txt");
File fp = new File ("demo.txt");

if (fp.isFile()){ //якщо об'єкт дисковий файл
System.out.println("Ім'я файлу:\t" + fp.getName());
System.out.println("Шлях до файлу:\t" + fp.getPath());
System.out.println("Абсолютний шлях:\t" +
fp.getAbsolutePath());
System.out.println("Канонічний шлях:\t" +
fp.getCanonicalPath());
System.out.println("Розмір файлу:\t" + fp.length());
System.out.println("Остання модифікація файлу:\t" +
fp.lastModified());
System.out.println("Файл доступний для читання:\t" +
fp.canRead());
System.out.println("Файл доступний для запису:\t" +
fp.canWrite());
System.out.println("Файл видалено:\t" + fp.delete());
}

if (fp.createNewFile()) {
System.out.println("Файл" + fp.getName() + "створено");
```

```
    }

    if (fp.exists()) {
    System.out.println ("temp файл" + fp.getName() + " існує");
    }

    else
    System.out.println("temp файл" + fp.getName() + " не існує");
    //в об'єкт типу File міститься каталог\директорія
    File dir = new File("com\learn");

    if (dir.isDirectory()) /*якщо об'єкт оголошено як каталог на
диску*/
    System.out.println("Директорія!");

    if (dir.exists )) { // якщо каталог існує
    System.out.println("Dir " + dir.getName() + " існує " );
    File [] files = dir.listFiles();
    System.out.println("");

    for (int i = 0 ; i < files.length; i ++) {
    Date date = new Date (files[i].lastModified ());
    System.out.println(files[i].getPath()    +    "    \t|"    +
files[i].length()
                        + "| " + date.toString());
    }}}}
```

Клас System.

Клас System містить набір корисних статичних методів та полів системного рівня. Екземпляр цього класу не може бути створено або отримано.

Найбільш широко використовується вивід інформації, що доступний через змінну System.out. Стандартний вивід можна перенаправити в інший потік (файл, масив байт та ін., головне, щоб це був об'єкт PrintStream, дивись документацію JSDK: http://docs.oracle.com/javase/6/docs/api/):

```
    import java.io.FileNotFoundException;
    import java.io.FileOutputStream;
    import java.io.PrintStream;

    public class Second {
    public static void main(String[] args) {
    System.out.println("Study Java");

    try {
    PrintStream print = new PrintStream(new
       FileOutputStream("d:\\file2.txt"));
    //місце на диску, куди зберігається файл
```

```
System.setOut(print);
System.out.println("Study well");
}

catch(FileNotFoundException e) {
e.printStackTrace();
}}}
```

При запуску цього коду на екран буде виведено тільки:
`Study Java`

А у файл "d:\file2.txt" буде записано:
`Study well`

Аналогічно може бути переправлене стандартне введення System.in — викликом System.setIn(InputStream) та потік виведення повідомлень про помилки System.err — викликом System.setErr (за замовчуванням усі потоки — in, out, err — працюють із консоллю програми).

Клас String.

Клас String містить основні методи для роботи зі рядками:

- **concat(String s)** або **+ —** злиття рядків;
- **equals(Object ob), equalsIgnoreCase(String s)** — порівняння рядків з урахуванням і без урахування регістру;
- **compareTo(String s), compareToIgnoreCase (String s)** — лексикографічнепорівняння рядків з урахуванням та без урахування регістру;
- **contentEquals(StringBuffer ob)** — порівняння рядків і вмісту об'єкта типу StringBuffer;
- **charAt(int n)** — вилучення з рядка символу із зазначеним номером (нумерація починається з нуля);
- **substring(int n, int m)** — вилучення з рядка підрядка довжиною mn, починаючи з n;
- **length()** — визначення довжини рядки;
- **valueOf(об'єкт)** — перетворення примітивного об'єкта до рядку;
- **toUpperCase() / доLowerCase()** — перетворення всіх символів зухвалою рядки в верхній нижній регістр;
- **replace(char з 1, char c2)** — заміна в рядку всіх входжень першого символу другим символ;
- **getBytes(параметри), getChars(параметри)** — вилучення символів рядки в вигляді масиву байт або символи.

У наступному прикладі масив символів і ціле число перетворюються в об'єкти типу String з використанням методів цього класу:

```
public class DemoString {
public static void main(String[] args) {
char s[] = { 'J', 'a', 'v', 'a' };
int i = 2;
// коментар містить результат виконання коду
String str = new String(s);  //str ="Java"
i = str.length();  // i=4
String num = String.valueOf(2);  //num="2"
str = str.toUpperCase();  //str="Java"
num = str.concat(num);  //num="Java2"
str = str + "C";  //str="JavaC";
char ch = str.charAt(2);  //ch='V'
i = str.lastIndexOf('A');  //i=3 (-1 якщо немає)
num = num.replace('2', 'H');  // num="JavaH"
i = num.compareTo(str);  //i=5 (між символами 'H' та 'C')
str.substring(0, 3).toLowerCase();  //java
}}
```

Масиви.

Масив (англ. Array) це об'єкт, що зберігає фіксовану кількість значень одного типу. Іншими словами, масив — це нумерований набір змінних. Змінна в масиві називається елементом масиву, а її позиція в масиві визначається індексом.

Масиви в Java теж є об'єктами. Вони мають бути оголошені, а потім створені. Щоб оголосити змінну, яка міститиме масив цілих чисел, використовуємо наступний синтаксис:

```
int[] arr;
```

Зверніть увагу, розмір не вказано, так як ще не створили масив.

```
arr = new int[10];
```

Тепер створили новий масив розміром 10. Перевірити розмір масиву можна вивівши на екран його довжину:

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

Щоб отримати доступ до масиву та встановити значення використовується наступний код:

```
arr[0] = 4;
arr[1] = arr[0] + 5;
```

Рахунок елементів масиву починається з **нуля**, тобто доступ до першого

елементу можна отримати за індексом 0 (наприклад, arr[0]).

```
int[] arr = new int[5]
arr[4] = 4; // Отримання доступу та присвоєння значення останньому елементу
```

Є також можливість створити масив із значень в один рядок:

```
int[] arr = {1, 2, 3, 4, 5};
```

* Якщо спробувати вивести масив на екран, буде отримано: [I@f7e6a96.

Для виведення всіх значень масиву використовується метод Arrays.toString(), який перетворює **масив у рядок**:

```
System.out.println(Arrays.toString(arr));
```

Або пишеться **цикл**, який послідовно виводить елементи масиву:

```
for(int i=0; i<arr.length; i++) {
        System.out.println(arr[i]);
 }
```

Цикли в Java.

Є два види циклів в Java, for і while.

Цикл FOR. Цикл for складається з трьох секцій:

```
for (int i = 0; i < 3; i++) {}
```

Першу секцію виконують один раз, коли входимо в цикл. У прикладі вище задається початкове значення змінної *i*. Друга секція перевіряє логічну умову, якщо вона повертає *true*, виконуються оператори у циклі, якщо *false*, вихід із циклу. Друга секція вперше запускається відразу після першої секції, і виконується щоразу, поки умова вірна, викликаючи третю секцію. Третя секція — заключний оператор, його дію виконується щоразу під час циклу. У наведеному прикладі це **інкремент**, який при кожному виконанні підвищує значення змінної на одиницю.

Таким чином, цикл працюватиме 3 рази. Ось порядок команд:

```
int i = 0;
i < 3 // 0 < 3 = true
      // Inside of loop
i++ // i is now 1
i < 3 // 1 < 3 = true
```

```
        // Inside of loop
i++ // i is now 2
i < 3 // 2 < 3 = true
        // Inside of loop
i++ // i is now 3
i < 3 // 3 < 3 = false
        // Loop is done...
```

Можна опустити першу і третю секції циклу, і цикл все ще працюватиме:

```
for (; i < 5;) {}
```

Цикл WHILE. Для випадків, де потрібно використовувати цикл подібних повторюваних дій, використовуємо цикл while. Синтаксис схожий на попередній:

```
while (condition) {}
```

Умова буде працювати вперше при введенні, та кожного разу, коли викликається цикл. Якщо умова поверне *false*, цикл не працюватиме. Якщо необхідно, щоб цикл завжди виконував принаймні **одну дію**, можна використовувати **do-while**:

```
do {
} while (condition);
```

! Не забуваємо крапку з комою в кінці.

FOREACH. Інша версія for, це foreach. Але в Java вирішили не додавати нове ключове слово each. Ключове слово, яке використовується все ще for, але коли необхідно виконати дії над елементами масиву, робиться так:

```
int[] arr = {2, 0, 1, 3};
for (int el: arr) {
     System.out.println(el);}
```

Це була коротка версія, еквівалентна наступному запису:

```
int[] arr = {1, 9, 9, 5};
for (int i = 0; i < arr.length; i++) {
     int el = arr [i];
     System.out.println(el);}
```

Зауважимо, що, якщо треба використовувати індекс елемента в циклі, то необхідно використовувати більш довгу версію запису, та не можна використовувати foreach.

break and continue. Ці два ключові слова допомагають керувати циклом. Оператор **break** зупиняє цикл і переходить до оператора, наступного за ним:

```
int i;
for (i = 0; i < 5; i++) {
     if (i >= 2) {
         break;
     }
     System.out.println("Yuhu");}
System.out.println(i);
// Output:
// Yuhu
// Yuhu
// 2
```

Оператор **continue** зупинить поточну ітерацію та переміститься до наступної. Зауважимо, що у циклі for дію у третій секції буде виконано.

```
int i;
for (i = 0; i < 5; i++) {
     if (i >= 3) {
         break;
     }
     System.out.println("Yuhu");
     if (i >= 1) {
         continue;
     }
     System.out.println("Tata");
}
System.out.println(i);
// Output
// Yuhu
// Tata
// Yuhu
// Yuhu
// 3
```

Завдання до лабораторної роботи №2:

Завдання 1. Обрати **два** пункти на вибір:

a) Ввести n рядків з консолі, знайти найкоротший рядок. Вивести цей рядок і його довжину, записати результат у файл.

b) Ввести n рядків з консолі. Впорядкувати і вивести рядки в порядку зростання їх довжин, а також вивести значення їх довжин, записати результат у файл.

c) Введіть n рядків з консолі. Вивести у консоль ті рядки, довжина яких менше середньої, а також вивести значення їх довжин, записати результат у файл.

d) У кожному слові тексту (використовувати читання з файлу), кожну k-ту букву замінити заданим користувачем символом. Якщо k більше довжини слова, коригування не виконувати.

e) У тексті кожну літеру замінити її номером у алфавіті (використовувати читання з файлу). В одному рядку друкувати текст з двома пробілами між літерами, у наступному рядку внизу під кожною буквою друкувати її номер.

f) З невеликого тексту (використовувати читання з файлу), видалити всі символи, крім пробілів, які не є літерами. Між усіма літерами залишити один пробіл.

g) З тексту видалити всі слова заданої довжини, що починаються на приголосну букву. Результат записати до файлу.

h) У тексті знайти всі пари слів, з яких одне є відображенням іншого (наприклад: сир-рис). Результати вивести у консоль та записати у файл.

i) Знайти і вивести у консоль, скільки разів повторюється в тексті кожне слово (використовувати читання з файлу).

j) Знайти, яких літер, голосних або приголосних, більше в тексті (використовувати читання з файлу).

Завдання 2. Заповніть масив випадковими числами та виведіть максимальне, мінімальне та середнє значення. Для генерації випадкового числа використовуйте метод Math.random(), який повертає значення у проміжку [0, 1].

Завдання 3. Реалізуйте алгоритм сортування бульбашкою для сортування масиву, отриманого у завданні 2.

Лабораторна робота №3
СПАДКУВАННЯ ТА ПОЛІМОРФІЗМ

Мета роботи. Здобути практичні навички створення ієрархій успадкування класів. Навчитися проектувати класи, використовуючи механізм успадкування. Мати уявлення про поліморфізм, переваги його використання при проектування класів.

ТЕОРЕТИЧНІ ВІДОМОСТІ

1. Загальні відомості про реалізацію механізму наслідування в Java.

Спадкування — один з трьох найважливіших механізмів об'єктно орієнтованого програмування (поряд з інкапсуляцією та поліморфізмом), що дозволяє описати новий клас на основі вже існуючого (батьківського), при цьому властивості і функціональність батьківського класу запозичуються новим класом. Іншими словами, клас-спадкоємець реалізує специфікацію вже існуючого класу (базового класу). Це дозволяє звертатись до об'єктів класу-спадкоємця так само, як до об'єктів базового класу. Клас, від якого відбулося спадкування, називається базовим чи батьківським (Base class). Класи, які усспадковані — називаються нащадками, спадкоємцями або похідними класами (Derived class).

Деякі мови використовують абстрактні класи. *Анотація* — це клас, що містить хоча б один абстрактний метод, він описаний у програмі, має поля, методи і не може використовуватись для безпосереднього створення об'єкта. Тобто від абстрактного класу можна тільки наслідувати. Об'єкти створюються тільки на основі похідних класів, успадкованих вид абстрактного класу. Наприклад, абстрактним класом може бути і базовий клас «співробітник вузу», від якого успадковуються класи «аспірант», «професор» та ін. Оскільки похідні класи мають спільні поля і функції (наприклад, поле «рік народження»), то ці члени класу можуть бути описані у базовому класі. У програмі створюються об'єкти на основі класів «аспірант», «професор», але немає сенсу створювати об'єкт на основі класу «співробітник вузу».

У мові Java підтримується тільки просте спадкування: будь-який підклас є похідним тільки від одного безпосереднього суперкласу. При цьому будь-який клас може успадковуватися від кількох інтерфейсів.

Спадкування інтерфейсів реалізує деяку заміну множинного наслідування, коли замість того щоб один клас мав кілька безпосередніх суперкласів, цей клас успадковує кілька інтерфейсів. **Інтерфейс** являє собою клас, у якому всі властивості — константі (тобто статичні — *static* і незмінні — *final*), а всі методи абстрактні, тобто інтерфейс дозволяє визначити певний шаблон класу: опис властивостей і методів без їх реалізації.

Мова Java дозволяє кілька рівнів спадкування, що визначаються безпосереднім суперкласом і опосередкованим суперкласом. Спадкування

можна використовувати для створення ієрархії класів.

При створенні підкласу на основі одного або кількох суперкласів можливі наступні способи зміни поведінки і структури класу:

- розширення суперкласу шляхом додавання нових даних і методів;
- заміна методів суперкласу шляхом їх перевизначення;
- злиття методів з суперкласів викликом однойменних методів з відповідних суперкласів.

Приклад успадкування:

```
public class Person {// неявне успадкування від класу Object
    public static final String GENDER_MALE = "MALE";
    public static final String GENDER_FEMALE = "FEMALE";

    public Person() {
    // Робити більше нічого...
    }

    private String name;
    private int age;
    private int height;
    private int weight;
    private String eyeColor;
    private String gender;
 }
```

Усі класи в Java неявно успадковують властивості і поведінки класу *Object*, тобтовін є предком всіх класів, які є в *JDK* (*Java Development Kit* — комплект розробки на мові Java), і які будуть створюватися користувачем. Наприклад: клас *Person* у прикладі вище неявно успадковує властивості *Object*. Так як це передбачається для кожного класу, не потрібно вводити *extends Object* для кожного визначення класу. Таким чином, клас *Person* має доступ до представлення змінних і методів свого суперкласу (*Object*). У даному випадку *Person* може «бачити» і використовувати загальнодоступні методи змінні об'єкта *Object*, а також його захищені методи і змінні.

2. Визначення ієрархії класів.

Визначимо новий клас *Employee*, який успадковує властивості *Person*. Його визначення класу (або граф успадкування) буде виглядати наступним чином:

```
public class Employee extends Person {
private String taxpayerIdentificationNumber;
private String employeeNumber;
private int salary ;
// . . .}
```

Граф успадкування для *Employee* вказує на те, що *Employee* має доступ до всіх загальнодоступних і захищених змінних і методів *Person* (так як він безпосередньо розширює його), а також *Object* (так як він фактично розширює і цей клас, хоча і опосередковано).

Для поглиблення в ієрархію класів ще на один крок можна створити третій клас, який розширює *Employee*:

```
public class Manager extends Employee {
// . . .
}
```

Будь-який клас може мати не більше одного суперкласу, але будь-яку кількість підкласів. Це найважливіша особливість ієрархії успадкування мови Java яку треба пам'ятати.

Всі методи конструктора використовують уточнення при перевизначенні методів по схеми зчеплення конструкторів. Зокрема, виконання конструктора починається із звернення до конструктору суперкласу, яке може бути явним або неявним. Для явного звернення до конструктора використовується оператор *super*, що вказує на виклик суперкласу (наприклад, *super*() викликає конструктор суперкласу без аргументів). Якщо ж в тілі конструктора явне звернення відсутнє, компілятор автоматично поміщає у першій рядку конструктора звернення до методу *super*(). Тобто у конструкторах використовується уточнення при **перевизначенні** методів, а в звичайних методах використовується **заміщення:**

```
public class Person {
 private String name;
 public Person() {}
    public Person(String name) {
             this.name = name;
 }
}
public class Employee extends Person {
 public Employee(String name) {
 super(name);
 }
}
```

3. Ключове слово this.

Іноді в класі необхідно звернутися до поточного примірника класу, який на даний момент обробляється методом. Для такого звернення використовується ключове слово ***this***. Застосування цієї конструкції зручно у разі необхідності звернення до поля поточного об'єкта, ім'я якого збігається з

ім'ям змінної, описаної в даному блоці (див. приклад вище).

4. Перевизначення методів.

Якщо наприклад: у базовому і дочірньому класах є методи з однаковими іменами і однаковими параметрами. У такому випадку доречно говорити про *перевизначення* методів, тобто у дочірньому класі змінюється реалізація вже існуючого в базовому класі методу. Якщо позначити метод модифікатором *final*, то метод не може бути перевизначеним. Поля не можна перевизначити, їх можна лише сховати. Розглянемо перевизначення методів:

```
public class Person {
 private String FIO;
 private int age;
 public Person() {}
 public String getFIO(){
 return this.FIO;
}
 public final setAge(int age){
 this.age = age;
 }
}
public class Employee extends Person {
 public Employee(String name) {
 super();
}
public String getFIO(){
 return "My name is " + super.getFio();
 }
}
```

5. Спадкування і абстракція.

Якщо підклас *перевизначає* метод з суперкласу, цей метод, по суті, прихован, тому що виклик цього методу за допомогою посилання на підклас викликає версії методу підкласу, а не версію суперкласу. Це не означає, що метод суперкласу стає недоступним. Підклас може викликати метод суперкласу, додавши перед ім'ям методу ключове слово *super* (та на відміну від правил для конструкторів, це можна зробити в будь-якому рядку методу підкласу або навіть зовсім іншого методу). За замовчуванням Java-програма викликає метод підкласу, якщо він викликається за допомогою посилання на підклас (див. приклад вище).

У контексті ООП абстрагування означає узагальнення даних і поведінки до типу, більш високого за ієрархією спадкування, ніж поточний клас. При переміщенні змінних або методів з підкласу в суперклас кажуть, що ці члени абстрагуються. Основною причиною цього є можливість багаторазового використання загального коду шляхом просування його якомога вище за ієрархією. Коли загальний код зібраний в одному місці, полегшується його

обслуговування.

6. Абстрактні класи та методи.

Бувають випадки, коли потрібно створювати класи, які служать тільки як абстракції, і створювати їх примірники ніколи не доведеться. Такі класи називаються абстрактними класами. Бувають випадки, коли певні методи повинні бути реалізовані по-різному для кожного підкласу, що реалізується суперкласом. Це абстрактні методи. Деякі основні правила для абстрактних класів і методів:

- будь-який клас може бути оголошений як абстрактний;
- абстрактні класи не допускають створення своїх примірників;
- абстрактний метод не може містити тіла методу;
- клас, що містить абстрактний метод, повинен оголошуватися як абстрактний.

7. Використання абстрагування.

Наприклад: необхідно безпосередньо заборонити можливість створення екземпляра класу *Employee*. Для цього достатньо оголосити його за допомогою ключового слова *abstract*:

```
public abstract class Employee extends Person {
}
```

Тоді виконання наступного коду виклече помилку компіляції:

```
public void someMethodSomwhere () {
Employee p = new Employee (); // помилка компіляції !
}
```

Компілятор вкаже на те, що *Employee* — це абстрактний клас, і його копія не може бути створена.

8. Можливості абстрагування.

Наприклад: необхідний метод вивчення стану об'єкта *Employee* і перевірки його правомірності. Ця вимога може бути загальною для всіх об'єктів *Employee*, але між всіма потенційними підкласами поведінка буде істотно відрізнятися, що дає нульовий потенціал для повторного використання. У цьому випадку метод *validate*() оголошується абстрактним (змушуючи всі підкласи реалізовувати його):

```
public abstract class Employee extends Person {
public abstract boolean validate ();
}
```

Тепер кожний прямий підклас *Employee* (такий як *Manager*), повинен реалізувати метод *validate*(). При цьому, як тільки підклас реалізував метод *validate*(), жодному з його підкласів не доведеться цього робити:

```
public abstract class Employee extends Person {
public abstract boolean validate();
}
public class Manager extends Employee {
public  boolean  validate()  {//метод  для  цього  класу
обов'язковий.
        boolean flag = true;
// код що розраховує стан об'єкту
 return flag;
 }
}
public class Executive extends Manager {
//метод public boolean validate() можно пропустити
}
```

9. Поліморфізм.

Поліморфізм в мовах програмування — це можливість об'єктів з однаковою специфікацією мати різну реалізацію. Мова програмування підтримує поліморфізм, якщо класи з однаковою специфікацією можуть мати різну реалізацію. Наприклад, реалізація класу може бути змінена у процесі успадкування. Коротко зміст поліморфізму можна виразити фразою: «*один інтерфейс, безліч реалізацій*».

Для Java характерні кілька способів організації поліморфізму:

1) Поліморфізм інтерфейсів. Інтерфейси описують методи, які повинні бути реалізовані в класі, і типи параметрів, які повинен отримувати і повертати кожен член класу, але не містять певної реалізації методів, залишаючи це на реалізуючий інтерфейс класу (див. приклад нижче). У цьому полягає поліморфізм інтерфейсів. Кілька класів можуть реалізовувати один і той самий інтерфейс, в той же час один клас може реалізовувати один чи більше інтерфейсів. Інтерфейси знаходяться поза ієрархією успадкування класів, тому вони виключають визначення методу або набору методів з ієрархії успадкування.

```
interface Shape {
void draw();
void erase();
}
class Circle implements Shape {
public void draw() { System.out.println("Circle.draw()");}
public void erase() { System.out.println("Circle.erase()");}
}
```

```
class Square implements Shape {
    public void draw() {
System.out.println("Square.draw()");}
    public void erase() {
System.out.println("Square.erase()");}
}
class Triangle implements Shape {
    public void draw() {
System.out.println("Triangle.draw()");}
    public void erase() {
System.out.println("Triangle.erase()");}
}
public class Shapes {
    public static Shape randShape() {
    switch((int)(Math.random() * 3)) {
        default:
        case 0: return new Circle();
        case 1: return new Square();
        case 2: return new Triangle();
        }
    }
    public static void main(String[] args) {
        Shape[] s = new Shape[9];
        for(int i = 0; i < s.length; i++)
            s[i] = randShape();
        for(int i = 0; i < s.length; i++)
            s[i].draw();
    }
}
```

2) Поліморфізм успадкування. При спадкуванні клас отримує всі методи, властивості та події базового класу такими, якими вони реалізовані у базовому класі. При необхідності в успадкованих класах можна визначати додаткові члени або перевизначати члени, що дісталися вид базового класу, щоб реалізувати їх інакше (дис. приклад нижче).

Наслідуваний клас також може реалізовувати інтерфейсів. У даному випадку поліморфізм виявляється в тому, що функціонал базового класу присутній в успадкованих класах неявно. Функціонал може бути доповнений та перевизначений. А успадковані класи, що несуть в собі цей функціонал виступають в ролі багатьох форм.

```
class Shape {
void draw() {}
void erase() {}
}
class Circle extends Shape {
    void draw() { System.out.println("Circle.draw()");}
    void erase() { System.out.println("Circle.erase()");}
}
```

```
class Square extends Shape {
    void draw() { System.out.println("Square.draw()");}
    void erase() { System.out.println("Square.erase()");}
}
class Triangle extends Shape {
    void draw() { System.out.println("Triangle.draw()");}
    void erase() { System.out.println("Triangle.erase()");}
}
public class Shapes {
    public static Shape randShape() {
        switch((int)(Math.random() * 3)) {
            default:
            case 0: return new Circle();
            case 1: return new Square();
            case 2: return new Triangle();
            }
    }
public static void main(String[] args) {
    Shape[] s = new Shape[9];
    for(int i = 0; i < s.length; i++)
        s[i] = randShape();
    for(int i = 0; i < s.length; i++)
        s[i].draw();
    }
 }
}
```

3) Поліморфізм за допомогою абстрактних класів. Абстрактні класи підтримують як успадкування, так і можливості інтерфейсів. При побудові складної ієрархії, для забезпечення поліморфізму програмісти часто змушені вводити методи в класи верхнього рівня при тому, що ці методи ще не визначені.

Абстрактний клас — це клас, екземпляр якого неможливо створити; цей клас може лише служити базовим класом при спадкуванні. Не можна оголошувати абстрактні конструктори або абстрактні статичні методи. Деякі або всі члени цього класу можуть залишатися нереалізованими, їх реалізацію повинен забезпечити клас нащадок. Похідні класи, які не переважають усі абстрактні методи, повинні бути визначені як абстрактні. Породжуваний клас може реалізовувати також додаткові інтерфейсів.

4) Поліморфізм методів. Здатність класів підтримувати різні реалізації методів з однаковими іменами — один із способів реалізації поліморфізму. Різні реалізації методів з однаковими іменами називається *перевантаженням* методів. На практиці часто доводиться реалізовувати один і той же метод для різних типів даних. Право вибору специфічної версії методу надано компілятору.

Окремим варіантом поліморфізму методів є поліморфізм методів зі

змінним числом аргументів. Перевантаження методів тут передбачено неявно, тобто перевантажений метод може викликатися з різним числом аргументів, а в деяких випадках навіть без параметрів. Перевантаження методів, як правило, робиться для тісно пов'язаних за змістом операцій. Відповідальність за побудую перевантажених методів і виконання ними однорідних за змістом операцій лежить на розробнику.

```
    public class PrintStream extends FilterOutputStream
implements Appendable, Closeable {
    /*…*/
     public void print(boolean b) {
        write(b ? "true" : "false");
     }
     public void print(char c) {
        write(String.valueOf(c));
     }
     public void print(int i) {
        write(String.valueOf(i));
     }
     public void print(long l) {
        write(String.valueOf(l));
     }
     /*…*/
     public void write(int b) {
     }
    }
```

5) Поліморфізм через перевизначення методів. Якщо перевантажені методи з однаковими іменами знаходяться в одному класі, списки параметрів повинні відрізнятись. Але якщо метод підкласу збігається з методом суперкласу, то метод підкласу перевизначає метод суперкласу. Збігатися при цьому повинні імена методів та типи вхідних параметрів. В даному випадку перевизначення методів є основою концепції динамічного зв'язування (або пізнє зв'язування), що реалізує поліморфізм. Суть динамічної диспетчеризації методів полягає в тому, що рішення на виклик перевизначеного методу приймається під час виконання, а не під час компіляції. Однак *final* -методи не є перевизначеним, їх виклик може бути організований під час компіляції і називається раннім зв'язуванням.

Завдання до лабораторної роботи №3

Завдання 1. За варіантом обрати одне завдання, створити класи, в них передбачити різні члени класів і методи для роботи, описати відповідні класи:

1) Базовий клас — учень. Похідні — школяр і студент. Створити клас Конференція, який може містити обидва типи учнів. Передбачити метод

підрахунку учасників конференції окремо по школярах і по студентах (використовувати оператор *instanceof*).

2) Базовий клас — працівник. Похідні — працівник на почасову оплату та на окладі. Створити клас Підприємство, який може містити обидва види працівників. Передбачити метод підрахунку працівників окремо на почасову оплату і на окладі (використовувати оператор *instanceof*).

3) Базовий клас — комп'ютер. Похідні — ноутбук і смартфон. Створити клас Ремонт-Сервіс, який може містити обидва види об'єктів. Передбачити метод підрахунку окремо ремонтованих ноутбуків і смартфонів (використовувати оператор *instanceof*).

4) Базовий клас — друковані видання. Похідні — книги та журнали. Створити клас Книжковий Магазин, який може містити обидва види об'єктів. Передбачити метод підрахунку окремо книг, окремо журналів (використовувати оператор *instanceof*).

5) Базовий клас — приміщення. Похідні — квартира та офіс. Створити клас Будинок, який може містити обидва види об'єктів. Передбачити метод підрахунку окремо квартир , окремо офісів (використовувати оператор *instanceof*).

6) Базовий клас — файл. Похідні — звуковий файл і відеофайл. Створити клас Каталог, який може містити обидва види об'єктів. Передбачити метод підрахунку окремозвукових та відео-файлів (використовувати оператор *instanceof*).

7) Базовий клас — літальний апарат. Похідні — літак і вертоліт. Створити клас Авіакомпанія, яка може містити обидва види об'єктів. Передбачити метод підрахунку окремо літаків, окремо вертольотів (використовувати оператор *instanceof*).

8) Базовий клас — змагання. Похідні — командні змагання та особисті. Створити клас Чемпіонат який може містити обидва види об'єктів. Передбачити метод підрахунку окремо командних змагань і особистих (використовувати оператор *instanceof*).

9) Базовий клас — меблі. Похідні — диван та шафа. Створити клас Кімната, який може містити обидва види об'єктів. Передбачити метод підрахунку окремо диванів, окремо шаф (використовувати оператор *instanceof*).

10) Базовий клас — зброя. Похідні — вогнепальна та холодна. Створити клас Збройна Палата, який може містити обидва види об'єктів. Передбачити метод підрахунку окремо вогнепальної і холодної зброї (використовувати оператор *instanceof*).

11) Базовий клас — оргтехніка. Похідні — принтер та сканер. Створити клас Офіс, який може містити обидва види об'єктів. Передбачити метод підрахунку окремо принтерів і сканерів (використовувати оператор *instanceof*).

12) Базовий клас — ЗМІ. Похідні — телеканал і газета. Створити клас

Холдинг, який може містити обидва види об'єктів. Передбачити метод підрахунку окремо телеканалів і газет (використовувати оператор *instanceof*).

Завдання 2. Обрати одне із завдань в залежності від парності варіанту:
(не парні варіанти)

Створити клас «Фільм» (MovieDVD) з наступними властивостями:
 назва фільму;
 режисер;
 рік випуску;
 жанр (1 — драма, 2 — комедія, 3 — серіал, 4 — бойовик)
 країна виробництва;
 основні актори;
 тривалість;
 базова вартість.
Клас повинен реалізовувати такі методи:
toString() — виведення повної інформації про фільм;
getPrice() — метод, що реалізує розрахунок ціни в залежності від жанру (для комедії — знижка 10%, для драми — 15%, для серіалу — 20%, для бойовика повна вартість).

(парні варіанти)
Реалізувати клас «Книга» (Book) з наступними властивостями:
 назва;
 автор;
 видавництво;
 рік видання;
 кількість сторінок;
 базова вартість.
Клас Book повинен реалізувати такі методи:
toString() — виведення повної інформації про книгу;
getPrice() — метод, що реалізує розрахунок ціни (для книг року видання до 2000 — 50% знижка, до 2020 року — знижка 30%, з 2023 року — націнка у 10%).

Лабораторна робота №4
КОЛЕКЦІЇ У JAVA

Мета роботи. Ознайомлення із основними структурами даних, такими як списки, множини та мапи. Важливо навчитися ефективно використовувати ці колекції для зберігання та обробки великих обсягів даних. Крім того, робота допомагає зрозуміти принципи роботи з ітераторами та методами сортування.

ТЕОРЕТИЧНІ ВІДОМОСТІ

При програмуванні операцій над групою однотипних об'єктів важливо вибирати найбільш ефективну структуру даних (клас) для зберігання цих об'єктів. У мові Java визначені спеціальні класи для зберігання однотипних об'єктів, які називаються колекціями, що визначають такі структури як перелік, безліч, черга. У лабораторної роботі розглядаються способи використання колекцій при розробці додатків.

Вибір певного класу для роботи з колекціями визначає набір методів, які будуть доступні для об'єкта цього класу. Наприклад, якщо використовується список (який визначає інтерфейс List), то є багатий вибір для його реалізації: ArrayList, LinkedList, Vector, Stack. Конкретний вибір реалізації списку впливає на ефективність маніпуляцій з об'єктами списку. Так, ArrayList зберігає елементи в вигляді масиву, а значить, доступ і заміна буде виконуватися відносно швидко. В той же час LinkedList зберігає елементи у вигляді зв'язного списку, що тягне за собою відносно повільний пошук елементів та швидку операцію додавання/видалення. Рекомендується ознайомитись з описами наступних колекцій по JSDK -документації:

- java.util.Collection ;
- java.util.ArrayList;
- java.util.HashMap;
- java.util.HashSet.

Важливо також знання основних класів для роботи з колекції. Особливу увага при вивченні колекцій заслуговують класи Comporator, Collections, Iterator.

Нижче наведено короткий розбір найбільш важливих класів при роботі з колекціями, а також рішення одного з індивідуальних завдань.

Інтерфейс Collection.

Інтерфейс містить набір загальних методів, які використовуються в більшості колекцій:

- **add**(Object item) — додає в колекцію новий елемент, якщо елементи колекції якимось чином упорядковані, новий елемент додається до кінця колекції;

- **clear**() — видаляє всі елементи колекції;
- **contains**(Object obj) — повертає true, якщо об'єкт obj міститься в колекції та false, якщо ні;
- **isEmpty**() — перевіряє true, якщо колекція порожня;
- **remove**(Object obj) — видаляє з колекції елемент obj, повертає false, якщо такого елемента в колекції не знайшлося;
- **size**() — повертає кількість елементів колекції.

Інтерфейс List.

Визначає впорядкований перелік елементів. Елементи списку пронумеровані, починаючи з нуля і до конкретного елементу можливо звернутися по індексу. Інтерфейс List є спадкоємцем інтерфейсу Collection, тому містить всі його методи та додає до них кілька своїх:

- **add**(int index, Object item) — вставляє елемент item у позицію index, причому список розширюється (всі елементи, починаючи з позиції index, збільшують свій індекс на 1);
- **get**(int index) — повертає об'єкт, що знаходиться в позиції index;
- **indexOf**(Object obj) — повертає індекс першої появи елемента obj в списку;
- **lastIndexOf**(Object obj) — повертає індекс останньої появи елемента obj в списку;
- **add**(int index, Object item) — замінює елемент, що знаходиться в позиції index об'єктом item;
- **subList**(int from, int to) — повертає новий список, що являє собою частину даного (починаючи з позиції from до позиції to-1 включно).

Інтерфейс Set.

Інтерфейс Set описує множину. Елементи множини не упорядковані, вона не може утримувати двох однакових елементів. Інтерфейс Set успадкований від інтерфейсу Collection, але ніяких нових методів не додає. Змінюється лише сенс методу **add** (Object item) — він не додає об'єкт item, якщо він вже присутній у множині.

Інтерфейс Queue.

Інтерфейс Queue описує чергу. Елементи можуть додаватися в чергу тільки з одного кінця, а витягуватись з іншого (аналогічно черзі в магазині). Інтерфейс Queue також успадкований від інтерфейсу Collection. Специфічні для черги методи:

- **poll**() — повертає перший елемент та видаляє його з черги;
- **peek**() — повертає перший елемент черги, не видаляючи його;
- **offer**(Object obj) — додає до кінця черги новий елемент і повертає true, якщо вставка вдалася.

Клас Vector.

Vector (Вектор) — набір упорядкованих елементів, до кожного з яких можна звернутися по індексу. По суті ця колекція представляє собою звичайний перелік. Клас Vector реалізує інтерфейс List, основні методи якого перелічено вище. До цих методів додається ще кілька:

- **firstElement()** — дозволяє звернутися до першого елементу вектору;
- **lastElement()** — дозволяє звернутися до його останнього елементу;
- **removeElementAt**(int pos) — видаляє елемент із заданої позиції;
- **removeRange**(int begin, int end) — видаляє кілька елементів, що йдуть підряд.

Всі ці операції також реалізуються комбінацією базових методів інтерфейсу List. Тому функціональність принципово не змінюється.

Клас ArrayList.

Клас ArrayList — аналог класу Vector. Він представляє собою перелік і може використовуватися в тих же ситуаціях. Основна відмінність в тому, що він не синхронізований і одночасна робота кількох паралельних процесів з об'єктом цього класу не рекомендується. У звичайних ситуаціях він працює швидше.

Клас Stack.

Stack — колекція, що поєднує елементи у стек. Стек працює за принципом LIFO (останнім прийшов — першим пішов). Елементи заносяться в стек «один на інший», причому можна дістати лише «верхній» елемент, тобто той, який був покладений у стек останнім. Для стеку характерні операції, реалізовані у наступних методах класу Stack:

- **push**(Object item) — поміщає елемент на вершину стеку;
- **pop**() — витягує зі стеку верхній елемент;
- **peek**() — повертає верхній елемент, не витягуючи його зі стеку;
- **empty**() — перевіряє, чи стек є порожній;
- **search**(Object item) — шукає «глибину» об'єкта в стеку. Верхній елемент має позицію 1, що знаходиться під ним — 2, і так далі. Якщо об'єкта в стеку немає, повертається -1.

Клас *Stack* є спадкоємцем класу Vector, тому має всі його методи (і реалізує інтерфейс List). Однак якщо у програмі потрібно моделювати саме стек, рекомендується використовувати тільки п'ять вище перелічених методів.

Інтерфейс Iterator.

Перевага використання масивів та колекцій полягає не тільки в тому, що є можливість помістити в них довільну кількість об'єктів і отримувати їх при необхідності, а й у тому, що ці об'єкти можна комплексно обробляти.

Наприклад, вивести на екран усі *елементи*, що містяться у списку *checkers*. При роботі з масивом використовується цикл:

```
for (int i = 1; i < array.length; i++){
// обробляємо елемент array[i]}
```

Працюючи зі списком, аналогічним чином використовується цикл, однак замість array[i] використовується array.get(i). Але з колекціями так робити не можна. Елементи колекцій не індексуються (наприклад, черга або множина). У разі індексованої колекції треба добре розуміти особливості її роботи: як визначити кількість елементів, як звернутися до елемента за індексом, чи може колекція бути розрідженою (тобто чи можуть існувати індекси, з якими не пов'язано жодних елементів та ін.).

Для навігації по колекціях в Java передбачено спеціальне архітектурне рішення, що отримало свою реалізацію в інтерфейсі Iterator. Ідея полягає в тому, що до колекції «прив'язується» об'єкт, єдине призначення якого — видати усі елементи цією колекції в деякому порядку, не розкриваючи її внутрішньої структури.

Інтерфейс **Iterator** має всього три методу:

- **next**() — повертає наступний у черзі елемент колекції, до якої «прив'язаний»ітератор (і робить його поточним). Порядок перебору визначає сам ітератор;
- **hasNext**() — повертає true, якщо перебір елементів ще не закінчено;
- **remove**() — видаляє поточний елемент.

Інтерфейс **Collection** крім розглянутих раніше методів, ще має метод **iterator**(), який повертає ітератор для цієї колекції, готовий до її обходу. За допомогою ітератора можна обробити всі елементи будь-який колекції наступним способом:

```
Iterator iter = coll.iterator();
// coll – колекція
while (iter.hasNext()) {
// обробляємо об'єкт, повертається методом iter.next()}
```

Для колекцій, елементи яких проіндексовані, визначено більш функціональний ітератор, що дозволяє рухатися як у прямому, так і у зворотному напрямі, а також додавати в колекцію елементи. Такий ітератор має інтрефейс **ListIterator**, успадкований від інтерфейсу Iterator, та доповнює його наступними методами:

- **previous**() — повертає попередній елемент (і робить його поточним);
- **hasPrevious**() — повертає true, якщо попередній елемент існує (тобто поточний елемент не є першим елементом для цього ітератора);

- **add**(Object item) — додає новий елемент перед поточним елементом;
- **set**(Object item) — замінює поточний елемент;
- **nextIndex**() та **previousIndex**() — служать для отримання індексів наступного і попереднього елементів відповідно.

В інтерфейсі List визначено метод listIterator(), який повертає ітератор ListIterator для обходу даного списку.

Інтерфейс Map.

Інтерфейс Map з пакету java.util описує колекцію, що складається з пар «ключ–значення», які широко використовуються для зберігання налаштувань в файлах конфігурації. Кожен ключ має лише одне значення, що відповідає математичному поняттю однозначної функції чи відображення (Map). Інтерфейс Map містить наступні методи:

- boolean **containsKey** (Object key) — перевіряє наявність ключа key;
- boolean **containsValue** (Object value) — перевіряє наявність значення value;
- **Set entry Set**() — представляє колекцію у вигляді множини, кожен елемент якого — пара з цього відображення, з якою можна працювати методами вкладеного інтерфейсу Map.Entry;
- Object **get** (Object key) — повертає значення, що відповідає ключу key;
- **Set key Set**() — представляє ключі колекції у вигляді множини;
- Object **put** (Object key, Object value) — додає пару «key–value», якщо такої пари не було, і замінює значення ключа key, якщо такий ключ вже є в колекції;
- void **putAll** (Map m) — додає до колекції всі пари з відображення m;
- **collection values** () — представляє всі значення у вигляді колекції.

У інтерфейс Map вкладений інтерфейс Map.Entry, що містить методи роботи з окремою парою.

Вкладений інтерфейс Map.Entry. Цей інтерфейс описує методи роботи з парами, отриманими методом entrySet() з об'єкта типу Map. Методи getKey() і getValue() дозволяють отримати ключ і значення пари, метод setValue (Object value) змінює значення в цій парі.

Приклад програми.

Обчислити скільки раз кожна літера зустрічається в текст.

```
import java.util.HashMap;
import java.util.*;
public class Main {
 public static void main(String[] args) {
    String txt = " лабораторна робота " ;
    HashMap<Character, Integer> map = new
```

```
        HashMap<Character, Integer>(40);
        for (int i = 0; i < txt.length(); ++i) {
        char c = txt.charAt(i);
        //перевіряємо є чи символ буквою
          if (Character.isLetter(c)){
            if (map.containsKey(c)) { map.put(c, map.get(c) + 1);
                    }
                else { map.put(c, 1);
                }
            }
        }
        //висновок на екран літер з частотою їх появи
        for (Entry<Character, Integer> entry : map.entrySet()) {
            System.out.println("літера: "+entry.getKey()+" кіл –
в: "+entry.getValue());
        }
      }}
```

Завдання до лабораторної роботи №4

Завдання 1. (Обрати 1 варіант):

1) Ввести рядки з файлу, записати їх в стек. Вивести рядки з файлу в зворотному порядку.

2) Ввести число, занести його цифри в стек. Вивести число, у якого цифри йдуть у зворотному порядку.

3) Скласти два багаточлена заданого ступеня, якщо коефіцієнти багаточленів зберігаються в об'єкті HashMap.

4) Створити стек з елементів каталогу. Не використовуючи допоміжних об'єктів, переставити негативні елементи даного списку в кінець, а позитивні — на початок цього списку.

5) Задати два стека, поміняти інформацію місцями.

6) Помножити два багаточлени заданого ступеня, якщо коефіцієнти багаточленів зберігаються у списку.

Завдання 2. (Обрати 1 варіант):

1) Організувати обчислення у вигляді стеку. Виконати попарне підсумовування довільного кінцевого ряду чисел наступним чином: на першому етапі підсумовуються попарно поруч стоячі числа; на другому етапі підсумовуються результати першого етапу, і так далі, доки не залишиться одне число.

2) Визначити клас Stack. Оголосити об'єкт класу. Ввести послідовність символів і вивести її в зворотному порядку.

3) Визначити клас Set на основі множини цілих чисел, n = розмір. Створити методи для визначення перетину і об'єднання множин.

4) Програма отримує N параметрів (аргументи командного рядка). Ці Параметри — елементи вектора. Будується масив типу double, а на базі цього масиву — об'єкт класу DoubleVector. Далі програма виводить в консоль значення елементів вектору в вигляді: Вектор: 2.3 5.0 7.3.

5) Списки I та U містять результати N вимірювань струму і напруги на невідомому опорі R. Знайти число R методом найменших квадратів.

Завдання 3. (Обрати 1 варіант).

Дано файл, що містить відомості про іграшки. Вказується 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:

Варіант 1. Вивести перелік іграшок, ціна яких не перевищує вказану і які підходять дітям 5 років у порядку зростання ціни.

Варіант 2. Вивести перелік конструкторів у порядку зростання ціни. (Ціну виводити за зразком …грн…коп).

Варіант 3. Вивести перелік найбільш коштовних іграшок (ціна яких відрізняється від ціни найкоштовнішої іграшки не більш ніж на 10 грн.). Відсортувати у порядку спадання ціни.

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

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 №5
ОБРОБКА РЯДІВ. ВИКОРИСТАННЯ РЕГУЛЯРНИХ ВИРАЗІВ У JAVA-ДОДАТКАХ

Мета роботи. Вивчення методів маніпуляції та аналізу текстових даних за допомогою регулярних виразів. Студенти повинні освоїти синтаксис і практичне використання регулярних виразів для пошуку, валідації та заміни тексту. Це також допомагає навчитися оптимізувати роботу з рядками для вирішення різних завдань у Java-додатках..

ТЕОРЕТИЧНІ ВІДОМОСТІ

Регулярні вирази – ця система обробки тексту, заснована на спеціальній системі запису зразків для пошуку. Зразок (*pattern*), що задає правило пошуку, також іноді називають «шаблоном», «маскою». Нині регулярні вирази використовуються багатьма текстовими редакторами та утилітами для пошуку та зміни тексту на основі обраних правил. Мова програмування Java також підтримує регулярні вирази для роботи зі рядками.

Основними класами для роботи з регулярні вирази є клас java.util.regex.Pattern і клас java.util.regex.Matcher.

Клас **java.util.regex.Pattern** застосовується для визначення регулярних виразів, для якого шукається відповідність в рядку, файлі або другому об'єкті що представляє собою деяку послідовність символів. Для визначення шаблону застосовуються спеціальні синтаксичні конструкції. Більше інформації можна отримати за допомогою класу **java.util.regex.Matcher.** Далі наведено основні логічні конструкції для шаблон. Якщо у рядку, що перевіряється на відповідність, необхідно, щоб у будь-якій позиції був один із символів деякого символьного набору, то такий набір (клас символів) можна оголосити, використовуючи одну з конструкцій, представлених у табл.1.

Таблиця 1 - Способи визначення класів символів.

[abc]	a, b або c
[^abc]	символ, крім a, b і c
[az]	символ між a і z
[ad[mp]]	або між a і d, або між m і p
[ez&&[dem]]	e або m (кон'юнкція)

Крім стандартних класів символів існують зумовлені класи символів (Табл. 2)

Таблиця 2 - Додаткові способи визначення класів символів.

.	будь-який символ
\d	[0-9]
\D	[^0-9]
\s	[\t\n\x0B\f\r]
\S	[^ \s]
\w	[a-zA-Z_0-9]
\W	[^\w]
\p{javaLowerCase}	теж , що і Character.isLowerCase()
\p{javaUpperCase}	теж, що і Character.isUpperCase()

При створенні регулярного виразу можуть використовуватись логічні операції (Табл.3).

Таблиця 3 - Способи завдання логічних операцій.

XY	Після X слід Y
X|Y	X або Y
(X)	X

Дужки, крім їх логічного призначення, також використовуються для виділення груп. Для визначення регулярних виразів недостатньо одних класів символів, тому для шаблонів часто потрібно вказати кількість повторень. Для цього існують Квантифікатори (табл. 4).

Таблиця 4 – Квантифікатори.

X?	X один раз або ні разу
X*	X нуль або більше разів
X+	X один або більше разів
X{n}	X n раз
X{n,}	X n або більше разів
X{n,m}	X від n до m

Існує ще два типи квантифікаторів, які утворені ***додаванням*** суфікса **?** (слабкий або неповний збіг) чи + («жадібне» або власне збіг) ***до*** вище перелічених квантифікаторів. Неповний збіг відповідає вибору з найменш можливою кількістю символів, а власне – з максимально можливим.

Клас **Pattern.**

Клас Pattern використовується для простої обробки рядків. Для більше складної обробки рядків використовується клас Matcher, що розглядається нижче.

У класі Pattern оголошені наступні методи:

- compile(String regex) - повертає Pattern, який відповідає regex;
- matcher(CharSequence input) - повертає Matcher, за допомогою якого можна знаходити відповідності у рядку input;
- matches(String regex, CharSequence input) - перевіряє на відповідність рядки input шаблону regex;
- pattern() - повертає рядок, відповідний шаблону;
- split(CharSequence input) – розбиває рядок input, враховуючи, що роздільником є шаблон;
- split(CharSequence input, int limit) - розбиває рядок input на не більше чим limit частин.

За допомогою методу **matches**() класу Pattern можна перевіряти на відповідність шаблону цілий рядок, але якщо необхідно знайти відповідності всередині рядка, наприклад, *визначати ділянки,* які відповідають шаблону, то клас Pattern **не може** бути використаний. Для таких операцій необхідно використовувати клас Matcher.

Клас **Matcher.**

Початковий стан об'єкта типу Matcher не визначено. Спроба викликати йогоабо метод класу для отримання інформації про знайдену відповідність призведе до вийнятку IlegallegalException. Для того щоб розпочати роботу з об'єктом Matcher потрібно викликати один з його методів:

- matches() - перевіряє, чи відповідає весь рядок шаблону;
- lookingAt() – намагається знайти послідовність символів, що починається з початку рядка і відповідає шаблону;
- find() або find(int start) - намагається знайти послідовність символів, відповідних шаблону, в будь-якому місці рядки. Параметр start вказує на початкову позицію пошуку.

Іноді необхідно скинути стан об'єкта класу Matcher у вихідний. Для цього застосовується метод **reset()** або **reset(CharSequence input)**, котрий також встановлює нову послідовність символів для пошуку.

Для заміни всіх послідовностей символів, що задовольняють шаблону,

на заданий рядок, можна застосувати метод **replaceAll(String replacement).**

Для того щоб обмежити пошук межами вхідних послідовностей застосовується метод region(int start, int end), а отримання значення цих меж можна за допомогою - regionEnd() та regionStart().

З регіонами (областями), пов'язано кілька методів:

- useAnchoringBounds(boolean b) – якщо встановлений у true, то початок і кінець регіону відповідає символу **^ і $** відповідно;
- hasAnchoringBounds() - перевіряє закріпленість кордонів.

У регулярному виразі для більшої зручності обробки вхідних послідовностей застосовуються групи, які допомагають виділити частини знайденої підпослідовності. У шаблоні вони позначаються дужками "**(**" та "**)**". Номери груп починаються з одиниці. Нульова група збігається зі всією знайденою підпослідовністю.

Далі наведено методи для вилучення інформації про групи:

- end() - повертає індекс останнього символу підпослідовності, що задовольняє шаблону;
- end(int group) - повертає індекс останнього символу зазначеної групи;
- group() – повертає всю підпослідовність, що задовольняє шаблону;
- group(int group) - повертає конкретну групу;
- groupCount() - повертає кількість груп;
- start() – повертає індекс першого символу підпослідовності,що задовольняє шаблону;
- start(int group) - повертає індекс першого символу зазначеної групи;
- hitEnd() – повертає істину, якщо було досягнуто кінця вхідної послідовності.

Наступний **приклад** показує використання можливостей класів Pattern та Matcher, для пошуку, розбору та розбиття рядків:

```
import java.util.regex.*;

public class DemoRegular {

    public static void main(String[] args) {
    // перевірка на відповідність рядка шаблону
    Pattern p1 = Pattern.compile("a*y");
    Matcher m1 = p1.matcher( "aaay" );
    boolean b = m1.matches();
    System.out.println(b);
    // пошук та вибір підрядка, заданого у шаблоні
    String regex = "(\\w+)@(\\w+\\.)(\\w+)(\\.\\w+)*" ;
```

```
        String s = "адреса ел. пошти: mymail@tut.ua та
rom@bsu.ua";
        Pattern p2 = Pattern. compile (regex);
        Matcher m2 = p2.matcher(s);
        while (m2.find()) {
        System.out.println("e-mail: " + m2.group());
        }
        // розбиття рядка на підрядки за допомогою шаблону в
якості роздільника
        Pattern p3 = Pattern.compile("\\d+\\s?");
        String[] words = p3.split("java5tiger 77 java6mustang");
        for (String word : words)
        System.out.println(word);
        }
    }
```

У результаті буде виведено:

```
true
e-mail: mymail@tut.bye-mail: rom@bsu.by java
tiger java mustang
```

Наступний **приклад** демонструє можливості використання груп, а також власних і неповних квантифікаторів:

```
import java.util.regex.*;

public class Groups {
    public static void main(String[] args) {
    String input = "abdcxyz";
    myMatches("([a-z]*)([a-z]+)", input);
    myMatches("([a-z]?)([a-z]+)", input);
    myMatches("([a-z]+)([a-z]*)", input);
    myMatches("([a-z]?)([a-z]?)", input);
    }

    public static void myMatches(String regex,
    String input) {
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(input);

    if(matcher.matches()) {
    System.out.println("First group: "
    + matcher.group(1));
    System.out.println("Second group: "
    + matcher.group(2));
    }
    else
        System.out.println("nothing");
```

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

Результат роботи програми:

```
First group: abdcxySecond group: z
First group: a Second group: bdcxyz
First group: abdcxyzSecond group:
nothing
```

У першому випадку до першої групи (First group) відносяться всі можливі символи, але при цьому залишається мінімальна кількість символів для другої групи (Second group). У другому випадку для першою групи вибирається найменша кількість символів, тому що використовується *слабкий* збіг. У третьому випадку першій групі буде відповідати весь рядок, а для другого не залишається жодного символу, оскільки друга група використовує *слабкий* збіг. У четвертому випадку рядок не відповідає регулярному виразу, тому для двох груп вибирається найменша кількість символів.

У класі Matcher оголошені два корисних методи для заміни знайдених підпослідовностей у вхідний рядок.

Matcher appendReplacement(StringBuffer sb, String replacement) - метод читає символи з вхідних рядків і додає їх до sb. Читання зупиняється на start() - 1 позиції попереднього збігу, після чого відбувається додавання в sb рядка replacement. При наступному виклику цього методу, додавання символів, починається з символу з індексом end() попереднього збігу.

Завдання до лабораторної роботи №5

Завдання 1. (Обрати два пункти):

a) Написати регулярний вираз, що визначає чи є цей рядок рядком «abcdefghijklmnopqrstuv18340» чи ні.

– приклад правильних виразів:
abcdefghijklmnopqrstuv18340.

– приклад неправильних виразів:
abcdefghijklmnoasdfasdpqrstuv18340.

b) Написати регулярний вираз, що визначає чи є цей рядок GUID з дужками або без них. Де GUID це рядок, що складається з 8, 4, 4, 4, 12 шістнадцяткових цифр розділених тире.

– приклад правильних виразів:
e02fd0e4-00fd-090A-ca30-0d00a0038ba0.

– приклад неправильних виразів:
e02fd0e400fd090Aca300d00a0038ba0.

c) Написати регулярний вираз, який визначає чи є заданий рядок правильною MAC-адресою.

– приклад правильних виразів: aE:dC:cA:56:76:54.
– приклад неправильних виразів: 01:23:45:67:89:Az.

d) Написати регулярний вираз, що визначає чи є цей рядок валідною URL-адресою. У цій задачі правильним URL вважаються адреси http і https, явне вказівка протоколу також може бути відсутня. Враховуються тільки адреси, які складаються з символів, тобто IP-адреси як URL не перевіряються. Допускаються піддомени, вказівка порту доступу через двокрапку, GET запити з передачею параметрів, доступ до підпапок на домені, допускається наявність якоря через ґрати. Однолітерні домени вважаються забороненими. Заборонені спецсимволи, наприклад «-» на початку та наприкінці імені домену. Заборонено символ «_» та пробіл в імені домену. При складанні регулярного виразу орієнтуйтесь на список правильних та неправильних виразів заданих нижче:

– приклад правильних виразів:
http://www.example.com, http://example.com.
– приклад неправильних виразів:
Just Text, http://a.com.

e) Написати регулярний вираз, що визначає чи є цей рядок шістнадцятковим ідентифікатором кольору HTML. Де #FFFFFF для білого, #000000 для чорного, #FF0000 для червоного і тому подібне.

– приклад правильних виразів: #FFFFFF, #FF3421, # 00ff00.
– приклад неправильних виразів: 232323, f#fddee, #fd2.

f) Написати регулярний вираз, що визначає чи є цей рядок датою в форматі dd/mm/yyyy. Починаючи з 1600 року до 9999 року.

- приклад правильних виразів: 29/02/2000, 30/04/2023, 01/01/2003.
- приклад неправильних виразів: 29/02/2001, 30-04-2023, 1/1/1899.

g) Написати регулярний вираз, що визначає чи є цей рядок валідною E-mail адресою згідно RFC під номером 2822:

– приклад правильних виразів:
user@example.com, root@localhost
– приклад неправильних виразів:
bug@@@com.com, @val.ua, Just Text2.

h) Скласти регулярне вираз, що визначає, чи заданий рядок є IP адресою, записаною у десятковому вигляді.

– приклад правильних виразів: 127.0.0.1, 255.255.255.0.
– приклад неправильних виразів: 1300.6.7.8, abc.def.gha.bcd.

i) Перевірити, чи надійно складено пароль. Пароль вважається надійним, якщо він складається з 8 або більше символів. Де символом може бути англійська літера, цифраі знак підкреслення. Пароль повинен містити хоча б одну велику літеру, одну маленьку букву і одну цифру.

– приклад правильних виразів: C00l_Pass, SupperPas1.

– приклад неправильних виразів: Cool_pass, C00l.

j) Перевірити чи є заданий рядок шестизначним числом, записаним удесятковій системі числення без нулів в старших розрядах.

– приклад правильних виразів: 123456, 234567.

– приклад неправильних виразів: 1234567, 12345.

k) Є текст зі списками цін. Вилучити з нього ціни в USD, UAH, EU.

– приклад правильних виразів: 23.78 USD.

– приклад неправильних виразів: 22 UDD, 0.002 USD.

l) Перевірити чи існують в тексті цифри, за якими стоїть "+".

– приклад правильних виразів: (3 + 5) - 9 × 4.

– приклад неправильних виразів: 2 * 9 - 6 × 5.

m) Створити запит для виводу тільки правильно написаних виразів зідужками (кількість відкритих та закритих дужок має бути однаково).

– приклад правильних виразів: (3 + 5) - 9 × 4.

– приклад неправильних виразів: ((3 + 5) - 9 × 4.

Завдання 2. (Оберіть одне завдання).

Розробити мовою програмування Java метод, що перевіряє коректність поштового індексу з використанням регулярних виразів. Обрати 1 стиль формування індексу з Таблиці 5.

Таблиця 5 – Поштові індекси.

№	Країна	Алгоритм формування індексу
1	Сполучені Штати Америки	Числовий код з дев’ятьма цифрами. Після п’ятої опціонально тире
2	Нідерланди	Чотири цифри. Перша цифра не нуль. Після опціонального пропуску код з двох літер
3	Італія	П’ять цифр. Попереду опціонально “V-“ чи “I-“.
4	Індія	Шість цифр. Перша цифра не нуль. Опціонально пропуск після третьої
5	Польща	Дві цифри, дефіс, три цифри
6	Іспанія	П’ять цифр. Після третьої цифри опціональний пропуск. Попереду опціонально “I-“.
7	Великобританія	Від 5 до 8 цифр і літер, що розділені пропуском. Наприклад, AA9A 9AA
8	Латвія	Чотири цифри Попереду опціонально “V-“ чи “I-“.
9	Македонія	Чотири цифри. Перша від 1 до 7
10	Португалія	Чотири цифр, дефіс, три цифри, пропуск, назва регіону до 25 літер

Створити 10 випадкових поштових індексів, визначити кількість коректних і не коректних.

Приклад роботи з регулярними виразами

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class NameChecker {

 private final static Pattern =
Pattern.compile("(^\S+\b)\s(\b\S+$)");
 public static void main(String[] args) {
    String str1 = "Hello Helen"; // true
    String str2 = "Good day"; // false
    String str3 = "Good good!"; // false
    System.out.println(isValid(str1));
    System.out.println(isValid(str2));
    System.out.println(isValid(str3));

 }
 private static boolean isValid(String str) {
    final Matcher m = pattern.matcher(str);
    return m.matches() && m.groupCount() == 2 &&
m.group(1).length() == m.group(2).length();
 }
}
```

Лабораторна робота №6
API-ІНТЕРФЕЙС JAVAFX CANVAS

Мета роботи. Ознайомлення студентів з можливостями створення графічних інтерфейсів у Java. Студенти навчаться використовувати Canvas для малювання графіки, таких як лінії, фігури та зображення, в JavaFX-додатках. Це розвиває навички побудови інтерактивних і візуальних елементів у сучасних програмах.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Розглянемо API-інтерфейс **JavaFX** Canvas. Він визначається за класами **Canvas**, **CanvasBuilder** і **GraphicsContext javafx.scene.canvas**. Використання цього API включає в себе створення Canvas об'єкта, його отримання **GraphicsContext** і запуск операцій малювання для відображення користувацьких фігур на екран. Оскільки **Canvas** це **Node** підклас, його можна використовувати в графі сцени **JavaFX**.

Малювання основних фігур.

Намалюємо деякі основні форми (лінії, овали, закруглені прямокутники, дуги і багатокутники **GraphicsContext** класу).

Створіть новий проект JavaFx (Jdk1.8). Видаліть створені автоматично пакети. Додайте свій пакет і в ньому новий клас. Вставте в нього наступний код для малювання основних фігур на полотні.

```
package laba_6_1;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.scene.shape.ArcType;
import javafx.stage.Stage;

public class laba_6_1 extends Application {

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

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Малювання основних фігур");
        Group root = new Group();
        Canvas canvas = new Canvas(500, 300);
        GraphicsContext gc = canvas.getGraphicsContext2D();
```

```
            drawShapes(gc);
            root.getChildren().add(canvas);
            primaryStage.setScene(new Scene(root));
            primaryStage.show();
        }
        /**
         * Малює ряд базових фігур, gc GraphicsContext об'єкт для
малювання
         */
        private void drawShapes(GraphicsContext gc) {
            gc.setFill(Color.RED);
            gc.setStroke(Color.AQUAMARINE);
            gc.setLineWidth(5);


            gc.strokeLine(40, 10, 10, 40);
            gc.fillOval(10, 60, 30, 30);
            gc.strokeOval(60, 60, 30, 30);
            gc.fillRoundRect(110, 60, 30, 30, 10, 10);
            gc.strokeRoundRect(160, 60, 30, 30, 10, 10);
            gc.fillArc(10, 110, 30, 30, 45, 240, ArcType.OPEN);
            gc.fillArc(60, 110, 30, 30, 45, 240, ArcType.CHORD);
            gc.fillArc(110, 110, 30, 30, 45, 240, ArcType.ROUND);
            gc.strokeArc(10, 160, 30, 30, 45, 240, ArcType.OPEN);
            gc.strokeArc(60,    160,    30,    30,    45,    240,
ArcType.CHORD);
            gc.strokeArc(110,    160,    30,    30,    45,    240,
ArcType.ROUND);
            gc.fillPolygon(new double[]{10, 40, 10, 40},
                    new double[]{210, 210, 240, 240}, 4);
            gc.strokePolygon(new double[]{60, 90, 60, 90},
                    new double[]{210, 210, 240, 240}, 4);
            gc.strokePolyline(new double[]{110, 140, 110, 140},
                    new double[]{210, 210, 240, 240}, 4);
        }
    }
```

Екземпляр **Canvas** (полотно) створюється з шириною 500 і заввишки 300. Потім його отримують за допомогою виклику **canvas.getGraphicsContext2D().** Після цього, кілька основних операцій малювань здійснюються з допомогою методів **strokeLine, fillOval, strokeArc,** і **fillPolygon.**

ArcType.CHORD — тип замикання для дуги, шляхом малювання відрізка прямої лінії від початку сегменту дуги до кінцевого сегмента дуги.

ArcType.OPEN — тип замикання для дуги без сегментів, що з'єднують два кінця сегменту дуги.

ArcType.ROUND — тип замикання для дуги, закритої шляхом малювання відрізків прямих ліній від початку сегмента дуги до центру

повного еліпса і від цієї точки до кінця сегменту дуги.

Застосування градієнтів та тіней.

Наступний приклад тестує GraphicsContext метод, малюючи користувацьку фігуру, разом з деякими градієнтами та тінями (Рис. 7). Код для цього прикладу організований так, що кожна операція малювання виконується в своєму власному приватному методі. Це дозволяє вам тестувати різні функції, просто викликаючи (або коментуючи) цікаві методи. З точки зору вивчення Canvas API код, — це базові виклики об'єктів Canvas або GraphicsContext.

Створіть новий проект.

```
/*
 * Copyright (c) 2012 Oracle and/or its affiliates...
 */
package laba_6_2;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.effect.DropShadow;
import javafx.scene.paint.Color;
import javafx.scene.paint.CycleMethod;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.RadialGradient;
import javafx.scene.paint.Stop;
import javafx.stage.Stage;

public class laba_6_2 extends Application {

    private Canvas canvas = new Canvas(200, 200);
    private GraphicsContext gc =
canvas.getGraphicsContext2D();
    private Group root = new Group();

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

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Laba_6_2");
        moveCanvas(0,0);
        drawHeart();
        drawRadialGradient(Color.BLACK, Color.HOTPINK);
        drawLinearGradient(Color.AQUA, Color.DARKBLUE);
```

```
            drawDropShadow(Color.CHARTREUSE,        Color.YELLOW,
Color.ORANGE, Color.RED);
            root.getChildren().add(canvas);
            primaryStage.setScene(new Scene(root, 200, 200));
            primaryStage.show();
        }

        /**
          * Переміщуємо полотно на нове місце у сцені
         */
        private void moveCanvas(int x, int y) {
            canvas.setTranslateX(x);
            canvas.setTranslateY(y);
        }

        /**
         * Малюємо серце за допомогою кубічних кривих Безьє
         */
        private void drawHeart() {
            gc.beginPath();
            gc.moveTo(95,60);
            gc.bezierCurveTo(95,57,90,45,70,45);
            gc.bezierCurveTo(40,45,40,82.5,40,82.5);
            gc.bezierCurveTo(40,100,60,122,95,140);
            gc.bezierCurveTo(130,122,150,100,150,82.5);
            gc.bezierCurveTo(150,82.5,150,45,120,45);
            gc.bezierCurveTo(105,45,95,57,95,60);
            gc.closePath();
        }

        /**
         * Радіальний градієнт
         */
        private void drawRadialGradient(Color firstColor, Color
lastColor) {
            gc.setFill(new RadialGradient(0, 0, 0.5, 0.5, 0.1,
true,
                    CycleMethod.REFLECT,
                    new Stop(0.0, firstColor),
                    new Stop(1.0, lastColor)));
            gc.fill();
        }

        /**
         * Лінійний градієнт
         */
        private void drawLinearGradient(Color firstColor, Color
secondColor) {
            LinearGradient lg = new LinearGradient(0, 0, 1, 1,
true,
                    CycleMethod.REFLECT,
```

```
                    new Stop(0.0, firstColor),
                    new Stop(1.0, secondColor));
            gc.setStroke(lg);
            gc.setLineWidth(3);
            gc.stroke();
        }

        /**
         * Тінь із 4 кольорів
         */
        private void drawDropShadow(Color firstColor, Color
secondColor,
                Color thirdColor, Color fourthColor) {
            gc.applyEffect(new DropShadow(20, 20, 0,
firstColor));
            gc.applyEffect(new DropShadow(20, 0, 20,
secondColor));
            gc.applyEffect(new DropShadow(20, -20, 0,
thirdColor));
            gc.applyEffect(new DropShadow(20, 0, -20,
fourthColor));
        }
    }
```

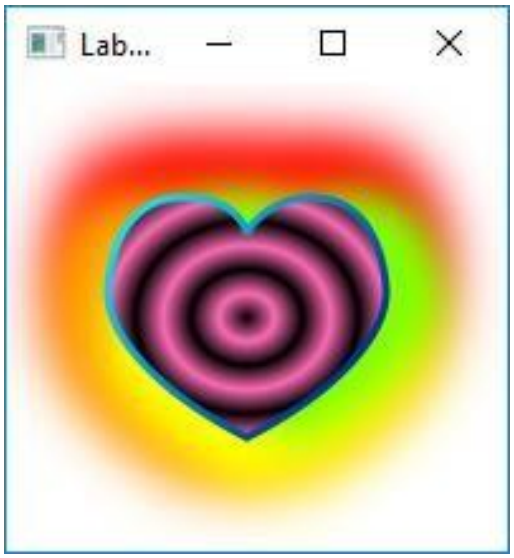

Рисунок 7 — Результат роботи програми.

- **Canvas** задається в координатах **(0,0)**. Можна передати інші значення в якості параметрів **setTranslateX** і **setTranslateY**, при цьому **Canvas** переміститься в задане місце.

Початок координат в — лівому верхньому кутку, Y вниз X праворуч.

- **RadialGradient** забезпечує круговий візерунок на задньому плані (всередині), **setFill** метод **GraphicsContext** приймає **RadialGradient** об'єкт в якості параметра.

- **LinearGradient** лінійний градієнт для кривих. Код встановлює штрих **GraphicsContext** для використання **LinearGradient**, а потім відображає шаблон за допомогою **gc.stroke**().
- **applyEffectна** різнокольорова тінь (Зовні). Створення **DropShadow** об'єкта із зазначеним кольором, котрий передається **applyEffect** методом **GraphicsContext** об'єкту.

Взаємодія з користувачем.

Створення найпростішого «малюнку», коли користувач затискаючи мишу - малює на полотні, при подвійному кліку сцена відчищається (Рис. 8).

```
package Laba_6_3;

import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class Laba_6_3 extends Application {

    public static void main(String[] args) {
        launch(args);
    }
     /** Скидає полотно у його первісний вигляд, заповнюючи
полотно*/
    private void reset(Canvas canvas, Color color) {
        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.setFill(color);
        gc.fillRect(0,         0,         canvas.getWidth(),
canvas.getHeight());
    }
     @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("PaintApp");
        Group root = new Group();
        // Задній фон
        Rectangle   rect   =   new   Rectangle(500,   500,
Color.WHITE);
        root.getChildren().add(rect);
        // Передній фон
       final Canvas canvas = new Canvas(500, 500);
       // canvas.setTranslateX(0);
       // canvas.setTranslateY(0);
```

```
            reset(canvas, Color.DARKBLUE);
            final GraphicsContext gc = canvas.getGraphicsContext2D();
            // Видалення переднього фону при натисканні ЛКМ та переміщенні по полотну
            canvas.addEventHandler(MouseEvent.MOUSE_DRAGGED, new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent e) {
                gc.clearRect(e.getX() - 1, e.getY() - 1, 2, 2);
                }
            });
            // Повернення переднього фону при подвійному кліку
            canvas.addEventHandler(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent t) {
                    if (t.getClickCount() >1) {
                        reset(canvas, Color.DARKBLUE);
                    }
                }
            });
            // Додавання полотна до сцени та відображення сцени
            root.getChildren().add(canvas);
            primaryStage.setScene(new Scene(root, 500, 500));
            primaryStage.show();
        }
    }
```

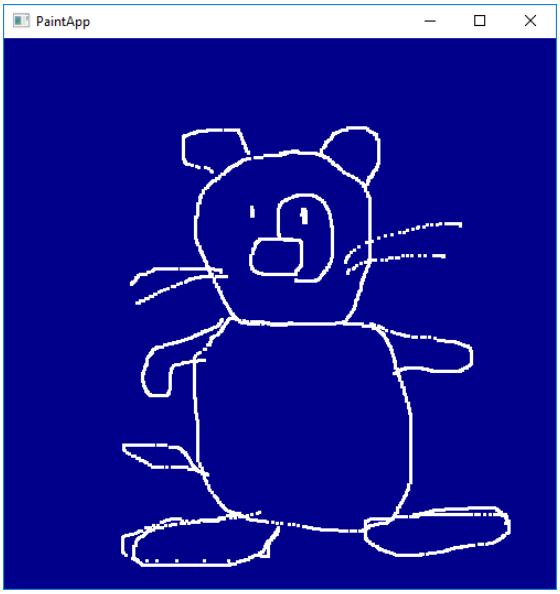

Рисунок 8 — Результат роботи програми.

- **reset** метод заповнює усе полотно синім кольором, **start** метод додає обробник подій **MouseEvent** об'єктів, коли користувач перетягує мишу. При кожному перетягуванні викликається **clearRect** метод **GraphicsContext**

об'єкта, передаючи поточні координати миші, а також розмір області, яку потрібно видалити. Коли це станеться, задній фон буде просвічуватися. Залишився код, що рахує кількість натискань і скидає синій фон у вихідний стан, якщо користувач двічі клацає мишею.

Завдання до лабораторної роботи №6

1. Намалювати і стилізувати свій особистий двовимірний малюнок на полотні.
2. Реалізувати видалення вашого малюнку при проведенні по ньому натиснутою мишею.
3. Повертати полотно у вихідний стан при потрійному кліку.
4. Додати можливість вибору розміру, форми та кольору пензля для малювання.

Лабораторна робота №7
РЕАЛІЗАЦІЯ ДИНАМІЧНОГО WEB-ПРОЄКТУ ЗАСОБАМИ СЕРВЛЕТІВ

Мета роботи. Навчитись створювати динамічні веб-додатки на Java з використанням сервлетів. Студенти опановують принципи обробки HTTP-запитів і відповідей для взаємодії з користувачем через браузер. Це дозволяє розуміти, як будуються серверні компоненти веб-додатків і забезпечується зв'язок між клієнтом і сервером.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Сервлет — це тип Java-класу, що виконується на сервері. Призначений для динамічного формування відповіді на запит, що надходить з мережі в основному по HTTP-протоколу. Сервлети розширюють функціональні можливості web серверу.

Обробка запиту статичної web-сторінки відбувається у наступній послідовності: у браузері користувач задає URL (Uniform Resource Locator — уніфікований вказівник інформаційного ресурсу), браузер формує запит за протоколом HTTP та направляє його вказаному в URL web-серверу. Сервер знаходить конкретний файл за запитом та повертає його браузеру у вигляді відповіді через протокол HTTP. HTTP-заголовок відповіді вказує на тип вмісту. Якщо тип MIME (Multipurpose Internet Mail Extensions) позначено як text/plain, то це звичайний текст у форматі ASCII, а якщо — text/html, то це HTML-документ.

Якщо вміст web-сторінки залежить від часу отримання запиту до неї, то її називають динамічною, а її вміст може змінюватись. Зазвичай, web-сторінка, що створюється як відповідь на запит до бази даних, є динамічною. Більшість сторінок, які отримуються на запити в Інтернеті є динамічними. Використання технології сервлетів для формування динамічних web-сторінок має ряд особливостей:

- сервлет виконується в адресному просторі web- серверу;
- обробка декількох запитів клієнта можуть виконуватись в одному процесі;
- сервлети не залежать від конкретної платформи, тому що розроблюються на мові Java;
- диспетчер безпеки Java на сервері створює ряд обмежень для захисту його ресурсів;
- для сервлету є доступними усі функціональні можливості бібліотек класів Java.

Сервлет може через механізми сокетів і віддаленого виклику методів (RMI) зв’язуватись з базою даних та іншим програмним забезпеченням.

Сервлети можуть бути вбудовані у різні сервери і виконуються у спеціальному середовищі виконання, який створює контейнер сервлетів або web-контейнер. Web-контейнер може бути компонентом-надбудовою (add-on) HTTP-сервера або окремим сервером як Tomcat, GlassFish, JBoss, WildFly та інші. Сервлети інсталюються до web-контейнерів як частину web-додатку, який також може вміщувати інші web-ресурси: HTML-сторінки, різні зображення та мультимедіа, JSP, XML-файли тощо. Після розгортання web-додатку у web-контейнері створюється та завантажується екземпляр класу сервлету на віртуальну машину Java (JVM) для обслуговування запитів, що надходять до серверу. Сервлет обслуговує кожний запит в окремому потоці. Тому, розробник сервлету вирішує як забезпечується синхронізація доступу до змінних екземпляру та змінних класу, визначає необхідність використання ресурсів баз даних та інше.

Контейнер сервлету (servlet container) забезпечує мережеві служби, за допомогою яких отримуються та декодуються запити, формуються та надсилаються відповіді. Усі контейнери сервлетів підтримують HTTP та HTTPS протоколи. Контейнер сервлету може бути розподіленим, тобто дозволяє запускати розподілені web-додатки на декількох віртуальних машинах Java. При цьому віртуальні машини можуть бути запущені, як на одному, так і на різних комп'ютерах. Контекстом сервлету (servlet context) є об'єкт, що вміщує представлення (вид) web-додатку, в якому запущено сервлет. Контекст використовується для того, щоб сервлет міг вести журнал подій, отримувати URL-посилання на ресурси, а також встановлювати та зберігати атрибути, які можуть використовувати інші сервлети у додатку. Відображення сервлету (servlet mapping) визначає зв'язок між структурою URL та сервлетом. Як правило використовується для відображення запитів до сервлетів.

Сервлети створюються як дочірні класи пакета javax.servlet. Це дозволяє через класи та інтерфейси пакета javax.servlet описувати та визначати контракти між класом сервлету та середовищем виконання. Середовищу виконання надається екземпляр класу сервлету за допомогою відповідного контейнеру. Повний опис пакету доступний за посиланням:

http://docs.oracle.com/javaee/7/api/javax/servlet/package-summary.html

Усі сервлети повинні реалізовувати інтерфейс javax.servlet.Servlet (http://docs.oracle.com/javaee/7/api/javax/servlet/Servlet.html) або безпосередньо, або розширюючи клас, який його реалізує, наприклад, HttpServlet (Рис. 9).

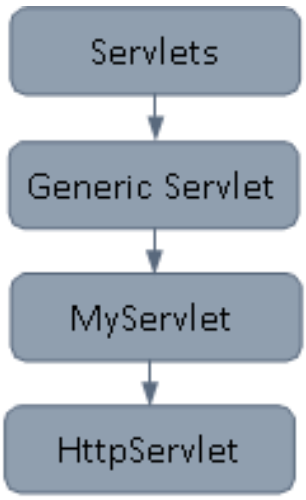

Рисунок 9 – Реалізація інтерфейсу.

Інтерфейс Servlet оголошує, але не реалізує методи, які керують сервлетом та його взаємодією з клієнтами. При написанні сервлету розробник повинен реалізовувати деякі або всі методи інтерфейсу сервлету.

Для забезпечення взаємодії з клієнтом використовуються два об'єкта:
- ServletRequest, який встановлює зв'язок від клієнта до серверу;
- ServletResponse, який встановлює зв'язок від сервлета до клієнту.

Обидва створюються на основі відповідних інтерфейсів пакета javax.servlet:
http://docs.oracle.com/javaee/7/api/javax/servlet/ServletRequest.html

Інтерфейс ServletRequest забезпечує сервлету доступ до:
- наданих клієнтом імен параметрів;
- протоколу, що використовується;
- імені віддаленого хосту, який виконав запит;
- імені сервера, який його отримав;
- вхідному потоку ServletInputStream.

Сервлети використовують вхідний потік для отримання даних від клієнтів, які використовують протоколи рівня додатків і такі методи як HTTP POST та PUT. Інтерфейси, які розширюють інтерфейс ServletRequest дозволяють сервлетам отримувати більш специфічні дані протоколу. Наприклад, інтерфейс HttpServletRequest має методи для отримання спеціальної інформації HTTP заголовків.

Інтерфейс ServletResponse надає сервлету методи, для відправлення повідомлень клієнту. Він дозволяє сервлету встановлювати довжину вмісту і тип MIME відповіді, а також встановлювати вихідний потік, ServletOutputStream та Writer, через який сервлет може відправляти дані

відповіді. Інтерфейси, що розширюють інтерфейс ServletResponse надають додаткові можливості. Наприклад, інтерфейс HttpServletResponse має методи, що дозволяють сервлету маніпулювати спеціальною інформацією HTTP заголовків.

Приклад реалізації Servlet

Застосуємо технологію сервлетів до створення динамічного web-проекту призначеного для перевірки логіну та паролю користувача.

1. Створюємо новий динамічний web-проект в IDE Eclipse і задаємо йому ім'я, наприклад, LoginServletApp. В навігаторі проектів Project Explorer розкриваємо Java Resources та на папці src клацаємо правою кнопкою миші. В меню обираємо New -> Servlet. У віконці Create Servlet, що відкрилось, вказуємо ім'я пакету ua.edu.znu.lab.servlet, ім'я нового класу LoginServlet та натискаємо Finish (Рис. 10).

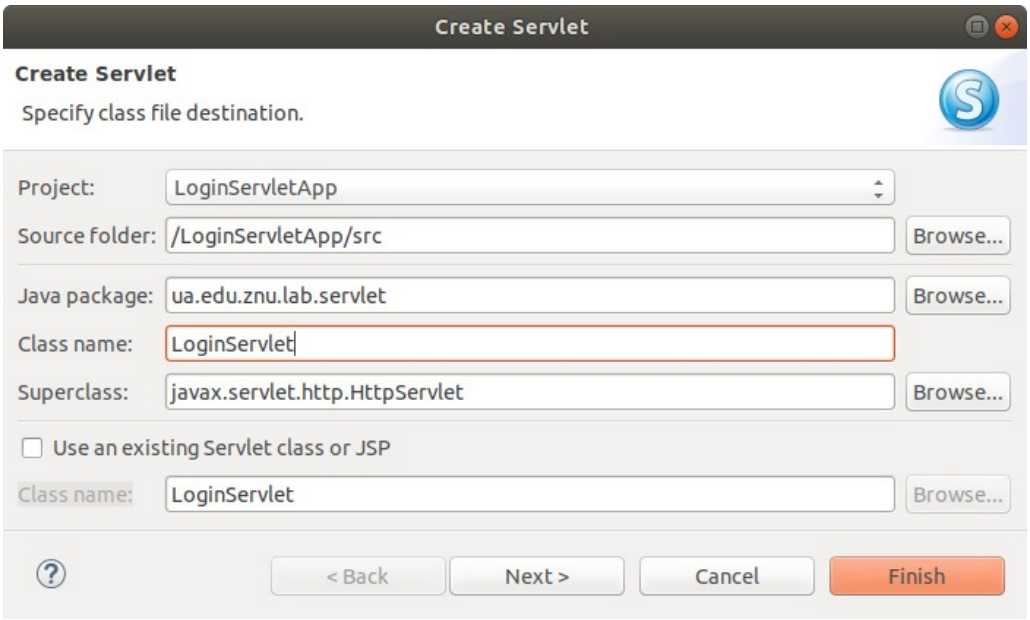

Рисунок 10 – Створення нового динамічного web-проекту.

2. Перед оголошенням класу є анотація @WebServlet("/LoginServlet"), яка вказує контейнеру сервлетів, що сервлет прив'язано до URL: /LoginServlet. У ранніх версіях Java EE, замість такої анотації, необхідно було в файлі web.xml робити дві записи: <servlet> та <servlet-mapping>.

Тепер замінюємо URL з /LoginServlet на /login, для чого змінюємо анотацію на @WebServlet("/login").

У створеному класі LoginServlet автоматично утворились конструктор LoginServlet() та два методи doGet та doPost. Ці методи є перевизначеними для базового класу: HttpServlet. Метод doGet використовується для створення відповіді на Get-запит, а метод doPost — відповідає на Post-запит. Логін-

форма буде створюватись у методі doGet, а обробка даних форми — в методі doPost. Але тому, що doPost може знадобитись для показу форми у випадку помилки введення даних, можна створити нову функцію createForm, яку можна викликати як з методу doPost, так і doGet.

3. Додаємо функцію createForm:

```
protected String createForm(String errMsg) {
  StringBuilder sb = new StringBuilder("<h2>Login</h2>");
  //перевірка відображення повідомлення про помилку
  if (errMsg != null) {
    sb.append("<span style='color: red;'>")
    .append(errMsg)
    .append("</span>");
  }
  //створення форми
  sb.append("<form method='post'>\n")
  .append("User Name: <input type='text'
name='userName'><br>\n")
  .append("Password: <input type='password'
name='password'><br>\n")
  .append("<button type='submit'
name='submit'>Submit</button>\n")
  .append("<button type='reset'>Reset</button>\n")
  .append("</form>");
  return sb.toString();
  }
```

4. Тепер змінюємо функцію doGet так, щоб з неї викликалась функція createForm, що повертає форму у відповідь:

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
  response.setContentType("text/html");
  response.getWriter().write(createForm(null));
}
```

Метод getWrite викликається на об’єкті response та записується вміст форми до нього, який отримується з виклику функції createForm. Коли форма показується вперше, то ніяких повідомлень про помилку ще не може бути, тому в якості аргументу функції createForm передається null.

Для того, щоб браузер вірно оброблював відповідь необхідно вказувати її тип: response.setContentType("text/html") інакше браузер може показувати текст відповіді і не інтерпретувати його, як html.

5. Для обробки вмісту форми, яка публікується після натискання

кнопки «Submit», змінюємо метод doPost:

```
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
       String userName = request.getParameter("userName");
       String password = request.getParameter("password");
       //Створення StringBuilder для рядка відповіді
       StringBuilder responseStr = new StringBuilder();
       if ("admin".equals(userName) && "admin".equals(password)) {
         responseStr.append("<h2>Welcome admin !</h2>")
         .append("You are successfully logged in");
       }
       else {
       //недійсні дані користувача
       responseStr.append(createForm("Invalid user id or
password.Please try again"));
       }
       response.setContentType("text/html");
       response.getWriter().write(responseStr.toString());
       }
```

Використання метода request.getParameter дозволяє отримати ім'я користувача та пароль від об'єкта запиту. Якщо логін та пароль дійсні, то до рядка відповіді буде додано привітальне повідомлення. Якщо логін та пароль введено не правильно, то відповіддю буде createForm з повідомленням про помилку.

На завершення, записується відповідь, яку сформовано в responseStr.

6. Для запуску на сервері необхідно клацнути правою кнопкою миші на файлі LoginServlet.java в Project Explorer та обрати опцію Run As | Run on Server. Попередньо проект не додавався до серверу додатків WildFly, тому Eclipse запитає про можливість конфігурації серверу запускати цей сервлет. Зараз слід натиснути кнопку Finish, сервлет повинен запуститись за адресою http://localhost:8080/LoginServletApp/login. Перевірте роботу сервлету.

Повний текст коду LoginServlet.java:

```
package ua.edu.znu.lab.servlet;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
* Servlet implementation class LoginServlet
*/
```

```
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
    public LoginServlet() {
      super();
    }
  protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
    response.setContentType("text/html");
    response.getWriter().write(createForm(null));
  }
  protected void doPost(HttpServletRequest request,
HttpServletResponse response)throws ServletException, IOException
{
    String userName = request.getParameter("userName");
    String password = request.getParameter("password");
    //Створення StringBuilder для рядка відповіді
    StringBuilder responseStr = new StringBuilder();
    if ("admin".equals(userName) &&
"admin".equals(password)) {
      responseStr.append("<h2>Welcome admin
!</h2>").append("You are successfully logged in");
    }
    else {
    //недійсні дані користувача
    responseStr.append(createForm("Invalid user id or
password! Please try again"));
    }
    response.setContentType("text/html");
    response.getWriter().write(responseStr.toString());
    }
  protected String createForm(String errMsg) {
    StringBuilder sb = new StringBuilder("<h2>Login</h2>");
    //перевірка відображення повідомлення про помилку
    if (errMsg != null) {
      sb.append("<span style='color: red;'>")
      .append(errMsg)
      .append("</span>");
    }
    //створення форми
    sb.append("<form method='post'>\n")
    .append("User Name: <input type='text' name ='userName'>
<br>\n")
    .append("Password: <input type='password'
name='password'> <br>\n")
    .append("<button type='submit' name='submit'>
Submit</button>\n").append("<button type='reset'>
Reset</button>\n").append("</form>");
    return sb.toString();
    }
```

```
}
```

Створення HTML форми або іншої сторінки у сервлеті є не дуже зручним, особливо при великих їх кількостях. Для цього краще використовувати JSP або HTML. Однак сервлети достатньо добре використовуються для реалізації контролерів в конфігурації Model-View-Controller (MVC), для обробки запитів, які генерують не текстову відповідь або для створення веб-сервісів або кінцевих точок веб-сокетів.

Завдання до лабораторної роботи №7

1. Виконайте пункти практичної частини лабораторної роботи (1-6).

2. Додайте у проект новий клас, що буде відповідати за утримання логінів та паролів, та створіть в ньому необхідні методи. Внесіть зміни у файл LoginServlet.java так, щоб логін та пароль користувача брались для перевірки саме з нового класу.

3. Внесіть зміни у код сторінки так, щоб кількість невдалих спроб пройти верифікацію обмежувалось трьома, а спроба оновити сторінку з формою після цього мало затримку 1 хвилину.

СПИСОК РЕКОМЕНДОВАНИЇ ЛІТЕРАТУРИ:

1. Gosling J. The Java Language Specification – Java SE 8 Edition / James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley. – 500 Oracle Parkway, Redwood City, California 94065, U.S.A. – 2015. – 788 p.
2. Олецький О. В. Перші кроки в JAVA. Навчальний посібник для студентів, які навчаються за спеціальностями “Інформатика”, “Програмна інженерія”, “Комп’ютерні науки”, “Прикладна математика”, Київ – 2017. — 144 с.
3 Bloch J. Effective Java: 3rd Edition, Addison Wesley. — 2017. — 412 p.
4. Horstmann C. S. Core Java Volume I – Fundamentals: 11th Edition, Prentice Hall. — 2018. — 889 p.
5. Grinev S. Mastering JavaFX 10. Build advanced and visually stunning Java applications. Packt Publishing. — 2018. — 268p.
6. Кадомський К.К., Ніколюк П.К. Java. Теорія і практика: навчальний посібник для студентів природничих спеціальностей університетів / Кадомський К.К., Ніколюк П.К. – Вінниця: Донну, 2019. – 197 с.
7. Васильєв О. Програмування мовою Java. Навчальна книга-Богдан, Тернопіль. — 2020. — 696 с.
8. Мартін Р. Чистий кодер. Видавництво: Фабула. — 2023. — 256 с. ISBN: 978-617-522-082-5
9. https://en.wikibooks.org/wiki/Java_Programming (Освоюємо Java – Вікіпідручник)
10. https://w3schoolsua.github.io/java/index.html#gsc.tab=0 (Java Tutorial)
11. http://javaland.com.ua (Програмування на Java)
12. https://riptutorial.com/ebook/javafx (Learning javafx eBook)

Навчальне видання

Методичні вказівки
до виконання лабораторних завдань
з навчальної дисципліни «Програмування на Java»
для студентів денної та заочної форм навчання
за спеціальностями «122 Комп’ютерні науки», та
«113 Прикладна математика»

Укладач:

ШАПОВАЛОВА Марія Ігорівна

Відповідальний за випуск доц. Водка О.О.

Роботу до видання рекомендував доц. Федоров В.О.

В авторській редакції

План 2024 р., поз. 893

Підпісано до видання 24.10.2024

Гарнітура Times New Roman. Обсяг - 3,3 др. арк.

Електронна версія