МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

МЕТОДИЧНІ ВКАЗІВКИ
ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

з дисципліни «Крос-платформне програмування»
(для студентів напряму підготовки
121 «Інженерія програмного забезпечення»)

З А Т В Е Р Д Ж Е Н О
на засіданні кафедри
інформаційних технологій
та програмування

Протокол № 7 від 14.03.25

Київ 2025

УДК 004.41:004.43

Методичні вказівки до виконання лабораторних робіт з дисципліни «Крос-платформне програмування» (для студентів напряму підготовки 121 «Інженерія програмного забезпечення») (Електронне видання) / Уклад.: Дьомін М. К. - Київ: Вид–во СНУ ім. В. Даля, 2025. – 50 с.

Укладено на основі ОП підготовки бакалаврів напряму 121 «Інженерія програмного забезпечення» робочої навчальної програми з дисципліни «Крос-платформне програмування».

Укладач: М.К.Дьомін, доц., к.т.н.

Резензент: О.І. Захожай, доц., д.т.н.

Зміст

ВСТУП

Методичні вказівки до виконання лабораторних робіт з дисципліни «Крос-платформне програмування» призначені для вивчення студентами основних методів розробки крос-платформного програмного забезпечення з використанням мови програмування Java.

Методичні вказівки містять теоретичну частину, опис лабораторних робіт, завдання і методичні рекомендації щодо їх виконання а також матеріали методичного характеру, що забезпечують успішне опанування програмного матеріалу відповідно до вимог освітньої програми.

Метою цих методичних матеріалів є ознайомлення студентів з мовою програмування Java та з методами створення крос-платформного програмного забезпечення.

В результаті вивчення дисципліни студент має:

- знати: мову програмування Java та основні методи розробки крос-платформного програмного забезпечення з використанням цієї мови.
- вміти: розробляти крос-платформне програмне забезпечення у сучасних інструментальних середовищах.
- володіти: практичними прийомами програмування на мові Java та прийомами крос-платформного програмування.

Лабораторна робота №1. Основи мови програмування Java. Знайомство з середовищем розробки NetBeans.

Ціль: навчитися створенню найпростіших крос-платформних застосунків на мові програмування Java.

Теоретичний матеріал

На відміну від C++ Java програма компілюється не у нативні команди процесора, а у спеціальній байт-код, який виконується за допомогою віртуальної машини Java. Тому для запуску програм на мові Java на комп'ютері має бути встановлена JRE (Java Runtime Environment). JRE є реалізацію віртуальної машини Java, яка також містить базову бібліотеку класів. Для кожної конкретної платформи є власна версія JRE.

Для розробки програм на Java знадобиться не лише JRE, а спеціальний комплект для розробника JDK (JavaDevelopmentKit). JDK вже містить JRE, а також ряд додаткових програм та утиліт, зокрема, компілятор Java – javac.

Завантажити та встановити відповідну версію JDK можна з офіційного сайту Oracle: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Інтегроване середовище розробки (Integrated Development Environment - IDE) IDE NetBeans надає широкі можливості та спрощує розробку настільних, мобільних та веб-додатків Java. Це безкоштовне програмне забезпечення з відкритим вихідним кодом, яке має велику спільноту користувачів та розробників по всьому світу. Завантажити та IDE NetBeans можна за адресою: https://netbeans.org/downloads/.

Також популярними IDE для розробки на Java є Eclipse та IntelliJ IDEA. Для освоєння цього курсу рекомендуємо використання саме IDE NetBeans.

Після встановлення JDK та NetBeans перейдемо до створення нашої першої програми на Java. Для цього запускаємо NetBeans та обираємо з меню File -> New Project. У меню створення проєкту обираємо Java with Gradle -> Java Application (Рис. 1).

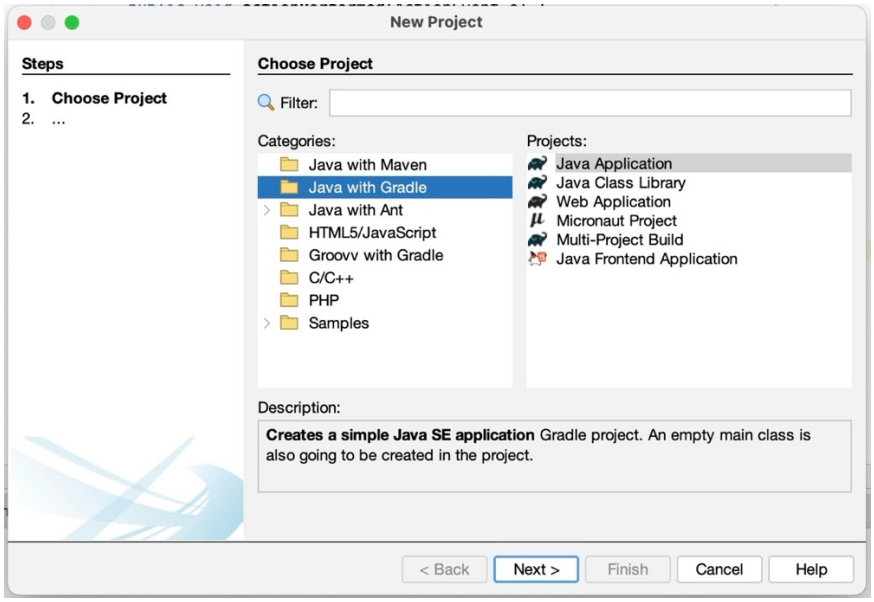


Рисунок 1. Створення нового проєкту у NetBeans.

Натискаємо на кнопку «Next» та вводимо ім’я проєкту, наприклад FirstJavaApplication. Після чого натискаємо на «Finish». Вікно проєкту представлено на рисунку 2.

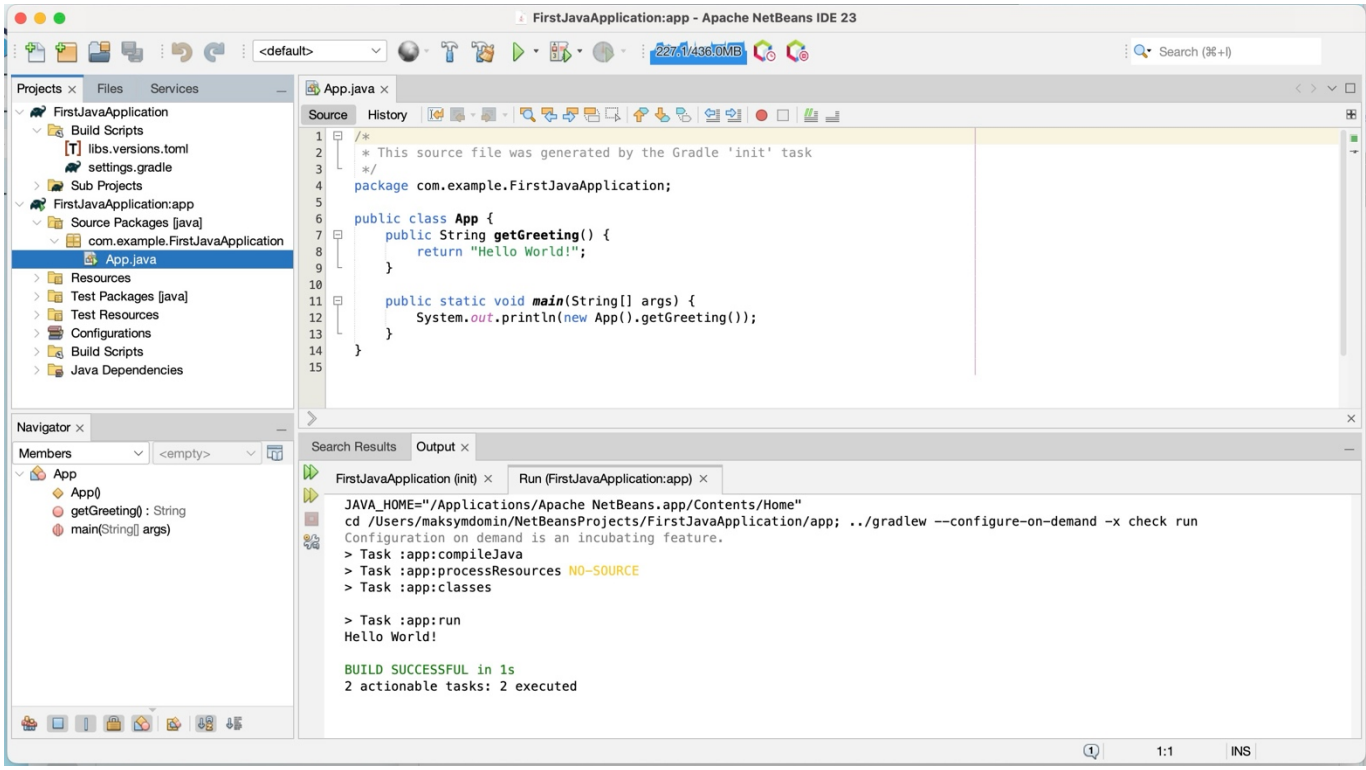


Рисунок 1. Вікно проєкту у NetBeans.

Зверніть увагу на те, де знаходяться вихідні файли програми. Зараз це єдиний файл App.java. Для компіляції та запуску програми використовуємо кнопку «Run Project» (зелена стрілка) на основній панелі інструментів.

Зверніть увагу, що поруч з цією кнопкою (справа) знаходиться кнопка для відлагодження програми «Debug Project».

Класи у Java групуються по пакетам. У нашому прикладі вихідний код застосунку містить єдиний пакет com.example.FirstJavaApplication. Зазвичай компанії використовують своє інвертоване ім'я інтернет-домена як префікс імені пакета, наприклад, com.example. FirstJavaApplication для названого пакету означає им’я пакету FirstJavaApplication для компанії, що має домен example.com. Це ім’я можна змінити при створенні проекту або при створенні додаткового пакету. Для виконання лабораторних рекомендовано використовувати імена пакетів у вигляді: ua.edu.snu.<student_sirname>.lab_<lab_number>. Наприклад, ua.edu.snu.petrenko.lab_1.

Розглянемо структуру найпростішої програми на Java. На початку файлу знаходиться секція з оголошенням пакета, якому належить поточний клас.

Далі може йти секція із підключенням зовнішніх пакетів за допомогою директиви import, після якої йдуть назви пакетів, що підключаються та класів з цих пакетів.

Мова Java має C-подібний синтаксис, тобто кожен рядок завершується крапкою з комою, а кожен блок коду міститься у фігурних дужках.

Далі йде визначення класу програми. Класи оголошуються наступним способом: спочатку йде модифікатор доступу, для класу з нашого прикладу - це public, який вказує, що цей клас буде доступним усім, тобто він може містити код, що може бути запущеній з командного рядка. Далі йде ключове слово class, а потім назва класу, і далі блок самого класу у фігурних дужках.

Клас може містити різні змінні та методи. У нашому прикладі ми маємо один метод main. Метод main є вхідний точкою програми, з нього починається все виконання. Він обов’язково повинний бути у програмі, тобто один з класів програми має обов’язково містити статичний метод main. У якості аргументів він приймає параметри командного рядка.

Перейдемо до основ мови програмування Java. Ми не будемо розглядати деякі особливості мови Java, які не відрізняються від C++. Такі елементи мови можуть бути зрозумілі з прикладів без додаткових пояснень.

Отже почнемо з типів даних у Java. У Java існують два основні групи типів даних, які відрізняються за способом зберігання та роботи з ними: **value-based** типи (типи на основі значень або примітивні типи) та **reference-based** типи (типи на основі посилань).

До типів на основі посилань відносяться:

- Цілі числа: byte, short, int, long
- Числа з плаваючою точкою: float, double
- Символи: char
- Логічний тип: boolean

При роботі з такими типами ми працюємо безпосередньо зі значенням, і кожна змінна такого типу зберігає окреме значення у своїй власній комірці пам’яті.

Характеристики примітивних типів:

- Зберігають значення напряму, тобто кожна змінна зберігає своє власне значення.
- Використовують стек пам’яті для зберігання змінних.
- Копіювання змінної створює копію значення, а не посилання на те ж саме значення.
- Під час передачі в методи примітивні типи копіюються за значенням, тобто зміни всередині методу не впливають на оригінальні значення.

Приклад:

```
int a = 10;
int b = a; // b отримує копію значення a
b = 20;
System.out.println(a); // Виведе 10, оскільки a не змінилося
```

Reference-based типи або посилальні типи — це всі об’єктні типи у Java, такі як класи, масиви та інтерфейси. На відміну від примітивних типів, змінні таких типів зберігають не саме значення, а посилання на об’єкт у пам’яті (зазвичай у кучі).

Основні посилальні типи в Java:

- Класи (наприклад, String, Integer, Object, користувацькі класи)
- Масиви (наприклад, int[], String[])
- Інтерфейси

Характеристики посилальних типів:

- Змінні зберігають посилання на об’єкт у кучі, а не саме значення.
- Копіювання змінної не створює новий об’єкт, а копіює посилання на той самий об’єкт.
- При передачі посилальних типів у методи передається посилання на об’єкт. Зміни, зроблені з об’єктом всередині методу, впливають на оригінальний об’єкт.

Приклад

```
class Car {
    String model;
}

Car car1 = new Car();
car1.model = "Tesla";

Car car2 = car1; // car2 отримує посилання на той же об'єкт, що і car1
```

```
car2.model = "BMW";

// Виведе "BMW", оскільки car1 і car2 посилаються на той же об'єкт
System.out.println(car1.model);
```

Наведемо приклад програми, яка обчислює значення функції sin(x) у точках
0, 0.2, 0.4, … 4 та виводить його, якщо воно більше ніж 0.1:

```
package com.example.FirstLabExample;

public class App {

    public static void main(String[] args) {
        for (double x = 0.0; x <= 4; x += 0.2) {
            double y = Math.sin(x);
            if (y > 0.1) {
                System.out.println("Value of sin(x) at " + x +
                                   " is " + y);
            }
        }
    }
}
```

Завдання

Розрахувати масив значень двох математичної функції для визначеного діапазону аргументів та визначити кількість значень функцій, що перевищують апріорно задане порогове значення. Першу функцію потрібно взяти з таблиці варіантів, другу функцію, діапазон значень та порогове значення визначити самостійно.

Варіанти завдань

№	Функція
1.	$f(x) = \frac{x^2 \sin(x)}{4}$
2.	$f(x) = (1 - \cos(x^2 - 2))$
3.	$f(x) = 12x^3 - 4x^2 + x$
4.	$f(x) = 2x^2 + 6$
5.	$f(x) = \cos(x) - sin^2(x)$
6.	$f(x) = \frac{x^2 - 1}{x + 1}$
7.	$f(x) = \frac{\mathrm{tg}(x)}{x}$

8.	$f(x) = \frac{x^2}{sin^2(x)}$
9.	$f(x) = \frac{x^3 \cos(x)}{5x} - 4$
10	$f(x) = \frac{1}{x^2} \sin(x)$

Лабораторна робота №2. Класи. Створення застосунків з графічним інтерфейсом користувача.

Ціль: освоєння базових принципів об’єктно-орієнтованого програмування у Java, вивчення основ створення Java-програм з графічним інтерфейсом користувача.

Теоретичний матеріал

Клас у Java — це шаблон для створення об’єктів. Він визначає:

- Поля (атрибути або властивості): відображають стан об’єктів.
- Методи: визначають поведінку об’єктів.
- Конструктори: Ініціалізують об’єкти під час їх створення.

Приклад об’явлення класу:

```
package com.example.vehicles

public class Car {
    String model;
    int year;

    Car(String model, int year) {
        this.model = model;
        this.year = year;
    }

    void start() {
        System.out.println(model + " is starting");
    }
}
```

Ім’я файлу Java має збігатися з назвою класу. Це стосується лише тих класів, що можуть бути використані в інших частинах програми. Такі класи перед об’явленням мають ключове слово public.

Члени класу можуть бути звичайними або статичними. Статичні члени класу в Java - це змінні та методи, які належать не конкретному об’єкту, а самому класу. Це означає, що вони спільні для всіх об’єктів цього класу і можуть використовуватися без створення екземпляра класу. Ключове слово static використовується для позначення таких членів.

Модифікатори доступу повинні вказуватись перед кожним членом класу. Модифікатори доступу бувають наступні:

- **public (публічний).** Елемент (клас, метод або змінна) з модифікатором public доступний з будь-якого місця в програмі, тобто з будь-якого класу чи пакета.
- **private (приватний).** Елемент з модифікатором private доступний тільки в межах класу, в якому він оголошений. Клас не може бути оголошений як private. Це можливо тільки для внутрішніх (вкладених) класів.
- **protected (захищений).** Елемент з модифікатором protected доступний: в межах одного пакета, у підкласах, навіть якщо вони знаходяться в іншому пакеті.
- **default (без модифікатора).** Якщо модифікатор не вказано, то доступ до елемента вважається default, або "пакетним" (іноді називають package-private). Елемент з default доступний тільки в межах того ж пакета.

Успадкування (англ. Inheritance) — це один із ключових принципів об'єктно-орієнтованого програмування (ООП), який дозволяє одному класу успадковувати властивості та поведінку іншого класу. Це означає, що новий клас (нащадок або підклас) може використовувати всі змінні та методи існуючого класу (батьківського або суперкласу), а також додавати або змінювати свою поведінку.

Основні поняття успадкування в Java:

- Батьківський клас (суперклас) — це існуючий клас, який передає свої змінні та методи підкласам. Батьківський клас може бути лише один, а також він повинний дозволяти успадкування.
- Підклас (клас-нащадок) — це новий клас, який успадковує змінні та методи батьківського класу.
- Перевизначення методів (overriding): підклас може перевизначити метод батьківського класу, змінюючи його поведінку. Для цього використовується ключове слово override.

Наведемо приклад:

```
class Vehicle {
    Vehicle() {
        System.out.println("Vehicle created");
    }
    void honk() {
        System.out.println("Beep beep!");
    }
}

class Car extends Vehicle {
    Car() {
        super(); // Викликає конструктор батьківського класу
        System.out.println("Car created");
    }
    @Override
    void honk() {
```

```
        System.out.println("Car beep!");
    }
}
public class Main {
    public static void main(String[] args) {
        Vehicle myVehicle = new Car();
        myVehicle.honk(); // Виведе: Car beep!
    }
}
```

Виклик конструктору батьківського класу відбувається через ключове слово super: Ключове слово super використовується для виклику конструктора або методів батьківського класу з підкласу.

Клас може містити абстрактні методи, у такому разі і сам клас буде абстрактним. Абстрактні методи, це метод, який додається у базовий клас, але не може мати реалізацію у ньому. Виходить, що дочірні класи відповідають за реалізацію абстрактних методів. Абстрактні класи потрібні для того, щоб забезпечити базову поведінку або структуру для підкласів, але дозволити кожному підкласу реалізувати власну специфічну поведінку.

Особливості абстрактних класів:

- Неможливо створити об’єкт абстрактного класу. Абстрактний клас може бути лише успадкованим іншими класами.
- Абстрактний клас може містити як абстрактні методи (без реалізації), так і звичайні методи з реалізацією.
- Ключове слово abstract використовується для оголошення абстрактного класу і абстрактних методів.
- Підклас, що успадковує абстрактний клас, повинен реалізувати всі абстрактні методи цього класу, якщо сам не є абстрактним.
- Абстрактний клас може містити поля та конструктори.
- Абстрактний клас може використовувати успадкування.

Приклад:

```
abstract class Animal {
    // Абстрактний метод (без реалізації)
    abstract void makeSound();

    // Звичайний метод
    void sleep() {
        System.out.println("Sleeping...");
    }
}
class Dog extends Animal {
    // Реалізація абстрактного методу
    @Override
    void makeSound() {
        System.out.println("Bark");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Animal dog = new Dog();
        dog.makeSound(); // Виведе "Bark"
        dog.sleep(); // Виведе "Sleeping..."
    }
}
```

Окрім абстрактних класів Java містить інтерфейси. Інтерфейс — це колекція абстрактних методів, які клас повинен реалізувати. Інтерфейси використовуються для того, щоб забезпечити контракт для класів, які цей інтерфейс реалізовують. Всі методи інтерфейсу за замовчуванням є абстрактними і публічними, але починаючи з Java 8, інтерфейси можуть містити статичні методи та методи з реалізацією за замовченням.

Особливості інтерфейсів:

- Клас, що реалізує інтерфейс, повинен реалізувати всі його методи (якщо він не абстрактний).
- Інтерфейси можуть містити:
    - Абстрактні методи (без реалізації).
    - Методи з реалізацією за замовченням методи.
    - Статичні методи.
    - Приватні методи (для внутрішнього використання) — з’явилися в Java 9.
- Множинна реалізація: клас може реалізовувати декілька інтерфейсів (на відміну від класів, де можливе лише одне успадкування).
- Інтерфейси використовуються для поліморфізму та для визначення того, які методи повинні бути реалізовані класом.
-

Приклад використання інтерфейсів:

```
interface Animal {
    // Абстрактний метод
    void makeSound();

    // Дефолтний метод
    default void sleep() {
        System.out.println("Sleeping...");
    }
}

class Dog implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Bark");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal dog = new Dog();
        dog.makeSound(); // Виведе "Bark"
```

```
        dog.sleep(); // Виведе "Sleeping..."
    }
}
```

Розглянемо основні класів, що дозволяють створювати програми з графічним інтерфейсом користувача. Ці класи містяться у бібліотеці Swing, яка є частиною JDK. Почнемо з класів JFrame та JPanel.

JFrame — це контейнер верхнього рівня, який представляє вікно з основними елементами, такими як рядок заголовка, кнопки.

JPanel — це загальний легкий контейнер для організації компонентів інтерфейсу користувача.

JFrame використовує JPanel як панель вмісту для зберігання компонентів. JFrame Може містити декілька панелей JPanel.

Наведемо приклад використання класів JFrame та JPanel:

```
import javax.swing.*;
import java.awt.*;

public class FrameExample {
    public static void main(String[] args) {
        // Створення вікна
        JFrame frame = new JFrame("Green JPanel Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);  // Встановити розмір вікна
        // Створення JPanel
        JPanel panel = new JPanel();
        // Встановити фоновий колір JPanel на зелений
        panel.setBackground(Color.GREEN);
        // Додати панель до вікна
        frame.add(panel);
        // Зробити вікно видимим
        frame.setVisible(true);
    }
}
```

JLabel — це компонент, який використовується для відображення текстового рядка або зображення. Мітки часто використовуються для опису інших компонентів (наприклад, полів введення чи кнопок) або для відображення статичної інформації.

Основні методи класу JLabel :

- setText(String text): встановити текст для мітки.
- setIcon(Icon icon): встановити зображення для мітки.
- setHorizontalAlignment(int alignment): встановити вирівнювання тексту.
- setForeground(Color color): встановити колір тексту.

JButton — це компонент, який користувачі можуть натиснути, щоб виконати дію. Це ключовий інтерактивний елемент у Swing. Кнопки можуть відображати як текст, так і піктограми, і вони зазвичай пов’язуються з

слухачами подій (наприклад, ActionListener), щоб визначити дії, які відбуваються під час натискання кнопки.

Основні властивості та методи:

- setText(String text): встановити текст для кнопки.
- setIcon(Icon icon): встановити зображення для кнопки.
- addActionListener(ActionListener listener): зареєструвати слухача подій для відповіді на натискання кнопки.

Більш детально розглянемо, яким чином відбувається обробка подій у бібліотеці Swing. Спочатку ми повинні створити об’єкт слухача подій. Слухачі подій є невід'ємною частиною керованої подіями моделі програмування, яка використовується для обробки взаємодії в графічних інтерфейсах користувача.

Компоненти, які можуть генерувати події, відомі як джерела подій (кнопки, вікна, текстові поля). Коли користувач взаємодіє з цими компонентами, генерується подія.

Кожна подія в Java представлена екземпляром об’єкта події, який містить деталі про подію.

Слухачі подій — це об’єкти, які «слухають» певний тип події та визначають, як реагувати, коли ця подія відбувається. Слухачі повинні реалізувати спеціальні інтерфейси із пакетів java.awt.event або javax.swing.event.

Щоб обробити подію, слухач події реєструється в джерелі події. Зазвичай це робиться за допомогою таких методів, як addActionListener(), addMouseListener() тощо.

Розглянемо, як це працює на прикладі:

```
import javax.swing.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        // Код, що виконується, коли натиснуто кнопку
        System.out.println("Button clicked!");
    }
}
public class EventListenerExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Event Listener Example");
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton button = new JButton("Click Me");
        // Додати слухача подій для кнопки
        button.addActionListener(new MyActionListener());
        frame.add(button);
        frame.setVisible(true);
```

```
    }
}
```

Відзначимо, що клас MyActionListener потрібний лише для створення одного об’єкту. У таких випадках можна використовувати так звані анонімні внутрішні класи. Перепишемо наш приклад з використанням анонімних класів.

```
import javax.swing.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

class MyActionListener implements ActionListener {}
public class EventListenerExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Event Listener Example");
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton button = new JButton("Click Me");
        // Додати слухача подій для кнопки
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Код, що виконується, коли натиснуто кнопку
                System.out.println("Button clicked!");
            }
        });
        frame.add(button);
        frame.setVisible(true);
    }
}
```

Рядок, у якому додається слухач подій, виглядає ніби то у ньому створюється об’єкт інтерфейсу ActionListener. Але ми пам’ятаємо, що створювати об’єкти інтерфейсів не можна. Насправді тут створюється анонімний внутрішній клас, який реалізує інтерфейс ActionListener. Він анонімний, бо його ім’я насправді не важливо. Все що нам потрібно — це створити один єдиній об’єкт цього класу і саме це і відбувається. Цей клас є внутрішнім, бо він створюється всередині іншого класу, а саме EventListenerExample. Внутрішні класи мають доступ до полів та методів зовнішніх класів.

Менеджери макетів автоматично впорядковують компоненти на основі попередньо визначених правил, спрощуючи дизайн GUI. Розробнику не потрібно вручну вказувати точні координати x і y для кожного компонента. Менеджери макетів обробляють зміну розміру та впорядкування компонентів, коли розмір вікна змінюється. Основні менеджери макетів, що можуть знадобитися при виконанні роботи: FlowLayout, BorderLayout, GridLayout.

FlowLayout:

- Компоненти розташовані горизонтально, один за одним (як слова в реченні), і переходять на наступний рядок, коли закінчується місце.

- Добре підходить для простих макетів з невеликою кількістю компонентів.

BorderLayout:

- Розділяє контейнер на п’ять областей: Північ, Південь, Схід, Захід і Центр.
- Компоненти, додані до цих регіонів, змінюють розмір так, щоб відповідати розміру регіону, центр займає простір, що залишився.

GridLayout:

- Розміщує компоненти в сітці з рядками та стовпцями.
- Розміри всіх компонентів сітки змінюються, щоб рівномірно відповідати клітинкам.

Наведемо декілька прикладів використання менеджерів компоновки. Почнемо з FlowLayout:

```
import javax.swing.*;
import java.awt.*;

public class LayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Layout Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 200);

        // Використання FlowLayout
        JPanel panel = new JPanel(new FlowLayout());

        panel.add(new JButton("Button 1"));
        panel.add(new JButton("Button 2"));
        panel.add(new JButton("Button 3"));

        frame.add(panel);
        frame.setVisible(true);
    }
}
```

Даля наведемо приклад використання BorderLayout.

```
import javax.swing.*;
import java.awt.*;

public class BorderLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("BorderLayout Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300); // Set the size of the frame
        // Використовуємо BorderLayout
        frame.setLayout(new BorderLayout());
        JButton northButton = new JButton("North");
        JButton southButton = new JButton("South");
        JButton eastButton = new JButton("East");
        JButton westButton = new JButton("West");
        JButton centerButton = new JButton("Center");
```

```
        frame.add(northButton, BorderLayout.NORTH);
        frame.add(southButton, BorderLayout.SOUTH);
        frame.add(eastButton, BorderLayout.EAST);
        frame.add(westButton, BorderLayout.WEST);
        frame.add(centerButton, BorderLayout.CENTER);

        frame.setVisible(true);
    }
}
```

Елемент вибору варіантів — це компонент, який дозволяє користувачам зробити один вибір із групи параметрів. Перемикачі представлені класом JRadioButton, і вони використовуються, коли потрібно обрати один параметр із попередньо визначеного набору. Перемикачі працюють у групах, тобто коли вибрано один перемикач у групі, вибір з усіх інших у групі автоматично знімається. Групування відбувається за допомогою класу ButtonGroup.

Основні методи класу JRadioButton:

- isSelected(): повертає true, якщо вибрано перемикач, і false в іншому випадку.
- setSelected(boolean): вибирає перемикач програмно.
- addActionListener(ActionListener listener): зареєструвати слухач подій для відповіді на натискання кнопки.

Приклад використання елементів вибору:

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class RadioButtonExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JRadioButton Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 200);
        // Створити елементи вибору
        JRadioButton option1 = new JRadioButton("Option 1");
        JRadioButton option2 = new JRadioButton("Option 2");
        JRadioButton option3 = new JRadioButton("Option 3");
        // Створити ButtonGroup для групування елементів вибору
        ButtonGroup group = new ButtonGroup();
        group.add(option1);
        group.add(option2);
        group.add(option3);
        // Створити мітку, що відображатиме обрану опцію
        JLabel label = new JLabel("Select an option:");
        // Додати слухачі подій для елементів
        option1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                label.setText("You selected Option 1");
            }
        });
```

```
        option2.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                label.setText("You selected Option 2");
            }
        });

        option3.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                label.setText("You selected Option 3");
            }
        });

        JPanel panel = new JPanel();
        panel.add(option1);
        panel.add(option2);
        panel.add(option3);
        panel.add(label);

        frame.add(panel);

        frame.setVisible(true);
    }
}
```

Текстове поле (JTextField) — це компонент, який може відображати один рядок тексту, що можна редагувати. Є можливість встановити початковий текст, отримати введені користувачем дані та реагувати на дії користувача (наприклад, натискання Enter).

Основні методи класу JTextField:

- setText(String text): встановити початковий текст у текстовому полі.
- getText(): повертає поточний текст із текстового поля.
- setHorizontalAlignment(int alignment): встановити горизонтальне вирівнювання тексту.
- addActionListener(ActionListener listener): реагує на натискання клавіші Enter, коли користувач вводить текст у текстовому полі.
- Document getDocument(): повертає модель, що асоційована з текстовим полем

Приклад використання текстового поля:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
 public class JTextFieldButtonExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTextField and JButton Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        frame.setSize(400, 200);
        frame.setLayout(new FlowLayout());

        // довжина - 20 символів
        JTextField textField = new JTextField(20);
        JButton button = new JButton("Click Me");
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Отримати текст з поля вводу та надрукувати його
                String inputText = textField.getText();
                System.out.println("Text entered: " + inputText);
            }
        });
        frame.add(textField);
        frame.add(button);
        frame.setVisible(true);
    }
}
```

Завдання 1. Варіанти

№	Завдання
1.	Розробити клас для роботи із простими дробами з наступною функціональністю: Реалізувати конструктори • () - за замовчуванням • (int num, int denom) - явне завдання чисельника й знаменника • (int value) - створення дроби з цілого числа • (double value, int digits) - створення дробу із числа з плаваючою крапкою, другий параметр - кількість чисел після коми, які треба враховувати. Реалізувати операції додавання, віднімання, множення, ділення. Після кожної зміни дроби робити автоматично операцію спрощення. Реалізувати операції перетворення типів: • toString() • toDouble() Реалізувати методи доступу до чисельника й знаменника: • getNumerator() • getDenominator()
2.	Розробити клас для роботи із прямокутниками з наступною функціональністю: Реалізувати конструктори ○ () - за замовчуванням

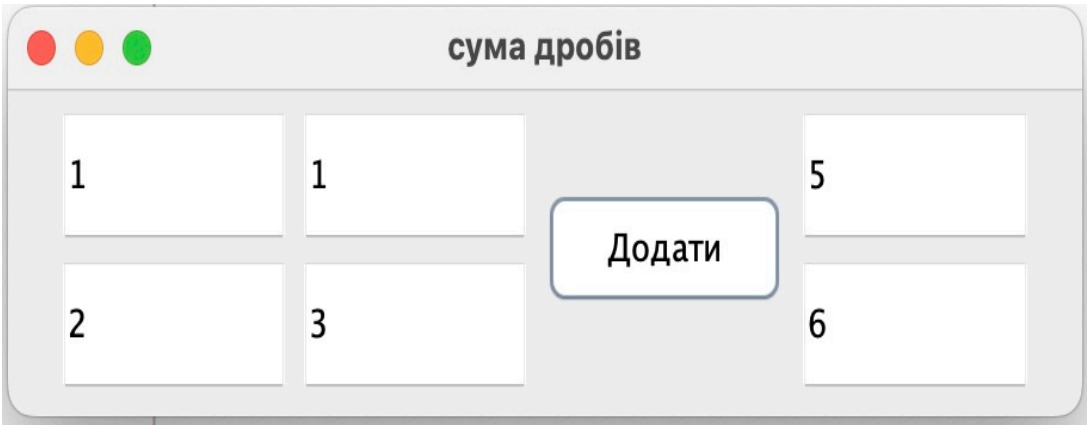
	○ (double x, double y, double width, double height) - завдання позиції та довжини з шириною ○ (double x, double y, double size) - завдання позиції та однакової довжини з шириною для створення квадрату ○ (double x1, double y1, double x2, double y2) - створення прямокутника за допомогою завдання координатів верхнього лівого та нижнього правого кута. Реалізувати операції обчислення перетину двох прямокутників, обчислення площі та обчислення найменшого прямокутника, що містить два інших прямокутника (так званий bounding box), . Реалізувати операцію перетворення типів: toString() Реалізувати методи доступу до координат кутів, до довжини та ширини.
3.	Розробити клас для роботи із векторами з наступною функціональністю: Реалізувати конструктори • () - за замовчуванням • (double x, double y) - завдання координат вектору Реалізувати операції обчислення суми векторів, обчислення векторного та скалярного добутку. Реалізувати операцію перетворення типів: toString() Реалізувати методи доступу до компонент вектору, та метод обчислення до довжини вектору, метод нормування вектору.
4.	Розробити клас для роботи із комплексними числами з наступною функціональністю: Реалізувати конструктори: • () - за замовчуванням • (double re, double im) - завдання дійсної та уявної частини Реалізувати операції обчислення суми, різниці та добутку та ділення. Реалізувати операцію перетворення типів: toString() Реалізувати методи доступу до дійсної та уявної частин та операції зведення у квадрат. Примітка: Будь-яке комплексне число може бути зображено формальною сумою x + *i*y, де x та y -дійсні числа, а *i* - це уявна одиниця. Уявна одиниця має наступну властивість: $i^2 = -1$. Арифметичні операції виконуються за наступними правилами: $z_1+z_2 = (a+bi) + (c+di) = (a+c) + (b+d)i$

	z_1-z_2 = (a+bi) - (c+di) = (a-c) + (b-d)i z_1z_2 = (a+bi) (c+d i) = (ac-bd) + (ad+bc)i $\frac{z_1}{z_2} = \frac{a+bi}{c+di} = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}i$
5.	Розробити клас для роботи із матрицями розміру 2x2 з наступною функціональністю: Реалізувати конструктори • () - за замовчуванням (створює одиничну матрицю) • (double a11, double a12, double a21, double a22) - завдання компонент матриці Реалізувати операції додавання, віднімання та перемноження матриць. Реалізувати операцію перетворення типів: toString() Реалізувати методи доступу до компонент матриці, та методи обчислення транспонованої та оберненої матриці, а також метод обчислення визначника матриці.

Завдання 2

Розробити графічний додаток, що дозволяє виконувати всі операції з об’єктами класу, що розроблений при виконанні завдання 1.

Нижче на малюнку представлений приклад додатка який виконує додавання дробів.



Лабораторна робота №3. Малювання простих зображень.

Ціль: вивчення методики створення графіки стандартними засобами мови Java.

Теоретичний матеріал

При створенні графічного інтерфейсу користувача у цій роботі необхідно буде використовувати список, що розкривається. Список, що розкриваються — це компонент, який надає користувачеві розкривний список елементів на вибір. Він широко використовується в додатках, щоб дозволити користувачам вибрати один елемент зі списку або за бажанням ввести спеціальне значення залежно від конфігурації.

JComboBox відображає список елементів, які можна розгорнути, натиснувши стрілку вниз. Користувачі можуть вибрати один пункт зі списку.

За замовчуванням JComboBox не можна редагувати (користувач може лише вибирати елементи зі списку). Однак можна зробити цей список редагованим, дозволяючи користувачам вводити власні дані на додаток до вибору з доступних параметрів.

JComboBox підтримується моделлю даних (ComboBoxModel), яка дозволяє маніпулювати елементами в списку.

Список, що розкриваються генерує події, коли користувач взаємодіє з ним (наприклад, вибирає елемент), що дозволяє розробникам реагувати на ввід користувача.

Також можна налаштувати спосіб відображення кожного елемента, встановивши спеціальний рендерер. Ця функція має назву — спеціальна візуалізація.

Основні операції:

- Конструювання:
  String[] items = { "Item 1", "Item 2", "Item 3" };
  JComboBox comboBox = new JComboBox(items);
- Додавання елементів:
  comboBox.addItem("Item 4");
- Додавання великої кількості елементів: потрібно створити DefaultComboBoxModel, заповнити її за допомогою addElement, а потім виконайте виклик методу setModel класу JComboBox.
- Видалення елементів:
  comboBox.removeItem("Item 2");
  comboBox.removeItemAt(1);

- Отримання індексу обраного елементу:
  int selectedIndex = comboBox.getSelectedIndex();

У Java Swing малювання стосується відображення графіки на компоненті, наприклад фігур, тексту, зображень або інших графічних елементів. Це досягається за допомогою об’єкта класу Graphics, який надає методи малювання на поверхні компонента. Якщо необхідно хочете налаштувати зовнішній вигляд компонента або створити власні графічні елементи, то необхідно реалізовувати ці функції згідно правилами, які будуть наведені нижче.

Найважливішим для малювання є метод paintComponent. Цей метод є частиною класу JComponent і відповідає за малювання компонента. Коли потрібно виконати нестандартне малювання в компоненті Swing, потрібно замістити метод paintComponent і використовувати об’єкт Graphics, переданій, як параметр. Цей метод викликається автоматично щоразу, коли компонент потрібно перемалювати (наприклад, під час зміни розміру вікна або явного виклику repaint()).

Графічний контекст — це абстрактний клас, наданий пакетом AWT (java.awt.Graphics). Він діє як контекст для всіх операцій малювання та надає різні методи малювання фігур, тексту, зображень та інших компонентів. У Swing об’єкт Graphics створюється бібліотекою та передається відповідним методам.

Під час виконання спеціального малювання в Swing або AWT метод paint(Graphics g) або paintComponent(Graphics g) надає об’єкт Graphics як аргумент. Однак цей об’єкт насправді є екземпляром Graphics2D (підклас Graphics), але сигнатура методу використовує більш загальний тип Graphics для зворотної сумісності.

Клас Graphics2D пропонує додаткові методи та функції, недоступні в Graphics. Наприклад:

- Трансформації: обертання, масштабування, перенесення або зсув фігур.
- Можливість визначення стилів ліній (наприклад, пунктирні лінії).
- Можливість контролювати якість візуалізації (наприклад, згладжування).
- Альфа-компонування: застосування ефектів прозорості та змішування.
- Розширені форми: використовуйте такі об’єкти фігур, як Rectangle2D, Ellipse2D і Path2D.

Ключові методи класу Graphics2D:

- drawLine(int x1, int y1, int x2, int y2).
- drawRect(int x, int y, int width, int height).
- fillRect(int x, int y, int width, int height).
- drawOval(int x, int y, int width, int height).

- fillOval(int x, int y, int width, int height).
- drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight).
- fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight).
- drawPolygon(int[] xPoints, int[] yPoints, int nPoints).
- fillPolygon(int[] xPoints, int[] yPoints, int nPoints)
- drawString(String str, int x, int y)
- setColor(Color c)

Найкращі практики, що мають застосовуватись при малюванні.

- Завжди викликайте super.paintComponent(g) на початку методу paintComponent, щоб переконатися, що фон компонента належним чином очищено.
- Використовуйте Graphics2D: якщо можливо, транслюйте об’єкт Graphics до Graphics2D для розширених функцій, таких як згладжування та трансформації.
- Уникайте інтенсивних обчислень: не виконуйте трудомісткі операції (наприклад, складні обчислення чи пошук даних) усередині paintComponent. Він має містити лише код малювання.
- Правильно запускайте перемальовування: використовуйте repaint(), щоб запускати повторне малювання, коли змінюється стан вашого компонента (наприклад, коли потрібно перемалювати фігуру).

Наведемо приклад реалізації малювання:

```
import javax.swing.*;
import java.awt.*;

public class Graphics2DExample extends JPanel {
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Перетворюємо Graphics на Graphics2D
        Graphics2D g2d = (Graphics2D) g;

        // Увімкнення згладжування (антиаліасінг)
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        // Малюємо заповнений прямокутник із закругленими кутами
        g2d.setColor(Color.BLUE);
        g2d.fillRoundRect(50, 50, 200, 100, 20, 20);

        // Малюємо повернутий текст
        g2d.setColor(Color.RED);
        g2d.rotate(Math.toRadians(45), 150, 150);
        g2d.drawString("Hello, Graphics2D!", 100, 150);
    }

    public static void main(String[] args) {
```

```
        JFrame frame = new JFrame("Graphics2D Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 400);
        frame.add(new Graphics2DExample());
        frame.setVisible(true);
    }
}
```

Завдання

Розробити графічний додаток, що виконує вибір фігури для малювання, вибір кольорів фігури, завдання наявності заливання, малювання фігури по натисканню на кнопку. Необхідно реалізувати малювання шести різних фігур. Для всіх варіантів - хрест, квадрат, коло й п'ятикутна зірка. У таблиці нижче наведені інші фігури по варіантах.

№	Завдання
1.	Ромб, серце
2.	Паралелограм, стрілка вправо
3.	Стрілка вниз, кільце
4.	Трапеція, правильний семикутник
5.	Фігура у вигляді букви «Н»
6.	Серце, паралелограм
7.	Правильний шестикутник, стрілка вліво
8.	Ромб, сонце (круг та промені у вигляді ліній)
9.	Серце, фігура у вигляді букви «П»
10.	Правильний восьмикутник, трапеція

Лабораторна робота №4. Колекції. Використання просунутих UI елементів бібліотеки Swing.

Ціль: вивчити методики створення складних графічних додатків, методи роботи з таймером, а також методи роботи з колекціями Java.

Теоретичний матеріал

У розробці програмного забезпечення керування даними має надважливе значення. Java надає обширну бібліотеку під назвою Java Collections Framework, яка дозволяє працювати з групами об'єктів ефективно та з використанням єдиного підходу до колекцій різних типів.

Java Collections Framework (JCF) – це бібліотека, яка є частиною JDK, що містить інтерфейси та класи, що їх реалізують, та надають можливість розробнику користуватися великою кількістю основних структур даних. JCF реалізує уніфіковану архітектуру для зберігання та керування групами об'єктів. JCF має вбудовану підтримку типових операцій, таких як пошук, сортування, вставка, видалення.

JCF заснований на 5 основних інтерфейсах: Collection, List, Set, Queue, Map. Інтерфейс Collection є батьківським для інтерфейсів List, Set, Queue.

Головні інтерфейси JCF:

- **Collection** – описує основні операції: додавання, видалення, отримання розміру.
- **List** – описує упорядковані колекції (також відомі як послідовності), може містити дублікати.
  Основні класи: ArrayList, LinkedList.
- **Set** - описує неупорядковані колекції, не може містити дублікати.
  Основні класи: HashSet, TreeSet.
- **Queue** – описує колекції типу FIFO (First-In-First-Out)
  Основні класи: LinkedList, PriorityQueue.
- **Map** - описує колекції, що зберігають пари типу ключ-значення.
  Основні класи: HashMap, TreeMap, LinkedHashMap.

При виконанні цієї роботи достатньо буде використання інтерфейсу List та його реалізації ArrayList.

Основні операції інтерфейсу Collection:

- **boolean add(E e);**
- **boolean addAll(Collection<? extends E> c)**

- **void clear()**
- **boolean contains(Object o)**
- **boolean containsAll(Collection<?> c)**
- **boolean equals(Object o)**
- **int size()**
- **boolean isEmpty()**
- **boolean removeAll(Collection<?> c)**
- **Iterator<E> iterator()**

Основні операції інтерфейсу List:

- **E get(int index)**
- **E set(int index, E element)**
- **void add(int index, E element)**
- **boolean addAll(int index, Collection<? extends E> c)**
- **E remove(int index)**
- **int indexOf(Object o)**
- **ListIterator<E> listIterator()**
- **ListIterator<E> listIterator(int index)**
- **List<E> subList(int fromIndex, int toIndex)**

Приклади використання колекцій:

```
// Робота зі списком:
List<String> list = new ArrayList<>();
list.add("Alice");
list.add("Bob");
list.add("Alice");
System.out.println(list);
// Робота з множиною:
Set<String> set = new HashSet<>();
set.add("Alice");
set.add("Bob");
set.add("Alice");
System.out.println(set);
// Робота з чергою:
Queue<String> queue = new LinkedList<>();
queue.add("First");
queue.add("Second");
queue.add("Third");
System.out.println(queue.poll()); // Виводить "First"
```

Розглянемо такі компоненти бібліотеки Swing які будуть необхідні привиконанні лабораторної роботи, а саме: слайдер, інкрементний регулятор меню, списки та таблиці.

Слайдер представлений класом JSlider. Це компонент для вибору значення з безперервного діапазону, пересуваючи ручку (регулятор). Зазвичай

він використовується для вибору таких значень, як гучність, яскравість або будь-якого іншого значення у визначеному діапазоні.

Створення слайдеру:

```
// Вертикальний слайдер з діапазоном 0..100 та початковим значенням 50
JSlider slider = new JSlider(JSlider.VERTICAL, 0, 100, 50);
```

Налаштування слайдеру

```
slider.setMajorTickSpacing(15);// Великі позначки через кожні 15
одиниць
slider.setMinorTickSpacing(3);// Маленькі позначки через кожні 3
одиниці
slider.setPaintTicks(true);   // Відображення позначок
slider.setPaintLabels(true);  // Відображення числових міток
slider.setSnapToTicks(true);  // Прив’язати регулятор до позначок
```

Приклад використання слайдеру:

```
import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.*;
public class SliderExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JSlider Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 200);
        JSlider slider = new JSlider(0, 100, 50);
        slider.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                JSlider source = (JSlider) e.getSource();
                int value = source.getValue();
                System.out.println("Slider value: " + value);
            }
        });
        frame.add(slider);
        frame.setVisible(true);
    }
}
```

Інкрементний регулятор представлений класом JSpinner. Це компонент, який дозволяє користувачеві вибирати значення з послідовності або діапазону значень, наприклад чисел, дат або списків. Його часто використовують для вводу або безпосередньо значення, або для надання можливості збільшувати/зменшувати значення за допомогою кнопок зі стрілками.

Використання JSpinner для вводу значення з діапазону:

```
// JSpinner із початковим значенням 10, мінімальним 0, максимальним 100
// і кроком 5
JSpinner spinner = new JSpinner(new SpinnerNumberModel(10, 0, 100, 5));
```

Використання JSpinner для вводу дати:

```
JSpinner dateSpinner = new JSpinner(new SpinnerDateModel());
dateSpinner.setEditor(new                JSpinner.DateEditor(dateSpinner,
"dd/MM/yyyy"));
```

Використання Jspinner для вибору значення:

```
String[] colors = {"Red", "Green", "Blue", "Yellow"};
JSpinner listSpinner = new JSpinner(new SpinnerListModel(colors));
```

Приклад використання інкрементного регулятору:

```
import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.*;

public class SpinnerExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JSpinner Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 200);
        JSpinner numberSpinner = new JSpinner(
            new SpinnerNumberModel(10, 0, 100, 1)
        );
        String[] colors = {"Red", "Green", "Blue", "Yellow"};
        JSpinner listSpinner = new JSpinner(
            new SpinnerListModel(colors)
        );

        listSpinner.addChangeListener(e -> {
            System.out.println("List Spinner Value: " +
            listSpinner.getValue());
        });
        frame.setLayout(new FlowLayout());
        frame.add(numberSpinner);
        frame.add(listSpinner);
        frame.setVisible(true);
    }
}
```

Для створення меню Swing пропонує наступні класи:

- JMenuBar: контейнер для всіх меню (наприклад, панель у верхній частині вікна).
- JMenu: представляє безпосередньо меню (наприклад, «Файл», «Редагувати»).
- JMenuItem: представляє окремі пункти меню, які можна натиснути, всередині JMenu.
- Меню можуть містити підменю (JMenu всередині іншого JMenu).

Меню також може містити прапорці або перемикачі. Для їх створення використовуються класи:

- JCheckfioxMenuItem – клас, що представляє елемент меню з прапорцем опцій.
- JRadioButtonMenuItem - клас, що представляє елемент меню з перемикачем.

Додати логіку до елементу меню можна за допомогою методу addActionListener. Альтернативний варіант – використання класу AbstractAction. Один і той самий об'єкт типу Action може бути пов'язаний з різними UI елементами. Action має метод setEnabled().

Приклад:

```
public class OpenAction extends AbstractAction {
    public OpenAction() {
        super("Open");
        putValue(SHORT_DESCRIPTION, "Opens a file");
        putValue(MNEMONIC_KEY, (int) 'O');
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Open action triggered");
    }
}
```

Списки у Swing представлені класом JList. JList використовується для відображення списку елементів для користувача, дозволяючи йому вибрати один або більше елементів.

Основні особливості списків:

- Одиночний або множинний вибір: можна налаштувати JList, щоб дозволити одиночний вибір (по одному елементу за раз) або множинний (наприклад, утримуючи клавішу Ctrl, щоб вибрати кілька елементів).
- Спеціальна візуалізація: можна налаштувати спосіб відображення кожного елемента в списку, надавши спеціальний ListCellRenderer.
- Можливість прокручування: помістивши JList у JScrollPane, можна створювати списки, які можна прокручувати, для великих наборів даних.

Приклад використання списку:

```
import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import java.awt.*;

public class JListExample {
    public static void main(String[] args) {
        String[] data = {"Item 1", "Item 2", "Item 3",
                         "Item 4", "Item 5"};
        JList<String> list = new JList<>(data);
        list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
        // Обробка подій вибору елементу
        list.addListSelectionListener(new ListSelectionListener() {
            @Override
            public void valueChanged(ListSelectionEvent e) {
                if (!e.getValueIsAdjusting()) {
                    System.out.println("Selected: " +
                      list.getSelectedValue());
                }
            }
        });
        JScrollPane scrollPane = new JScrollPane(list);
        frame.add(scrollPane, BorderLayout.CENTER);
        frame.setVisible(true);
    }
}
```

Додаткові приклади використання списків наведені нижче.

Спеціальна візуалізація:

```
class MyCellRenderer extends DefaultListCellRenderer {
    @Override
    public Component getListCellRendererComponent(JList<?> list, Object
value, int index, boolean isSelected, boolean cellHasFocus) {
        JLabel label = (JLabel) super.getListCellRendererComponent(list,
value, index, isSelected, cellHasFocus);
        label.setIcon(new ImageIcon("path_to_icon.png"));
        label.setText("Item " + value);
        return label;
    }
}
list.setCellRenderer(new MyCellRenderer());
```

Додавання та видалення елементів:

```
DefaultListModel<String> model = new DefaultListModel<>();
model.addElement("New Item 1");
model.addElement("New Item 2");
model.removeElementAt(0);
```

Списки з множинним вибором:

```
list.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
List<String> selectedValues = list.getSelectedValuesList();
System.out.println("Selected items: " + selectedValues);
```

Клас JTable використовується для відображення та редагування табличних даних. Він дозволяє упорядковувати дані в рядки та стовпці, а також пропонує такі функції, як сортування, редагування та спеціальна візуалізація елементів.

Основні компоненти таблиці:

- Модель таблиці (JTableModel): TableModel визначає структуру та дані таблиці.

- Модель стовпців (JTableColumnModel): визначає поведінку та вигляд стовпців.
- Рендерер клітинок: керує відображенням окремих клітинок.
- Редактор клітинок: контролює спосіб редагування клітинок.

Приклад використання таблиці:

```
import javax.swing.*;

public class JTableExample {
    public static void main(String[] args) {
        String[][] data = {
                {"John", "Doe", "35"},
                {"Jane", "Doe", "30"},
                {"Jack", "Smith", "40"}
        };
        String[] columnNames = {"First Name", "Last Name", "Age"};
        JTable table = new JTable(data, columnNames);

        JScrollPane scrollPane = new JScrollPane(table);

        JFrame frame = new JFrame("JTable Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(scrollPane);
        frame.setSize(400, 200);
        frame.setVisible(true);
    }
}
```

Далі наведені приклади коду, що дозволяють використовувати деякі спеціальні функції таблиць.

Таблиці, що дозволяють редагування:

```
DefaultTableModel model = new DefaultTableModel(data, columnNames) {
    @Override
    public boolean isCellEditable(int row, int column) {
        return true;
    }
};
JTable table = new JTable(model);
```

Спеціальна візуалізація:

```
import javax.swing.table.DefaultTableCellRenderer;
import java.awt.Component;
class MyCellRenderer extends DefaultTableCellRenderer {
    @Override
    public Component getTableCellRendererComponent(JTable table, Object
value, boolean isSelected, boolean hasFocus, int row, int column) {
        Component c = super.getTableCellRendererComponent(table, value,
                                    isSelected, hasFocus, row, column);
        if (row %  2 == 0) {
            c.setBackground(Color.YELLOW);
        } else {
```

```
            c.setBackground(Color.WHITE);
        }
        return c;
    }
}
table.getColumnModel().getColumn(2).setCellRenderer(
  new MyCellRenderer()
);
```

Реакція на події:

```
table.getSelectionModel().addListSelectionListener(event -> {
    int selectedRow = table.getSelectedRow();
    if (selectedRow != -1) {
        System.out.println("Selected row: " + selectedRow);
    }
});
```

Сортування:

```
TableRowSorter<DefaultTableModel> sorter =
  new TableRowSorter<TableModel >(model);
table.setRowSorter(sorter);
```

Фільтрування:

```
RowFilter<DefaultTableModel, Object> filter =
RowFilter.regexFilter("Doe", 1);
sorter.setRowFilter(filter);
```

Спеціальне редагування:

```
TableColumn column = table.getColumnModel().getColumn(1);
JComboBox<String> comboBox = new JComboBox<>(new String[]{"Doe",
"Smith", "Johnson"});
column.setCellEditor(new DefaultCellEditor(comboBox));
```

Для одержання дати й часу використати клас GregorianCalendar.

```
GregorianCalendar calendar = new GregorianCalendar();
// встановити поточний час
calendar.setTimeInMillis(System.currentTimeMillis());
```

Щоб витягти хвилини використовуємо calendar.get(Calendar.MINUTE);

Інші константи:

години (24) - Calendar.HOUR;

години (12) - Calendar.HOUR_OF_DAY;
секунди - Calendar.SECOND.

Щоб виконувати певні дії через фіксовані проміжки часу використовуються таймери.

Створення таймера, що буде викликати ваш код кожну секунду:

Timer timer = new Timer(1000, listener);

Запуск таймера: timer.start();

listener - об'єкт класу, що реалізує інтерфейс ActionListener, у методі actionPerformed завдаються дії, що виконуються по таймеру.

Завдання

Створити програму, що:

1. Міститиме JTable, самостійно обрати 3-4 назви для стовбців, таким чином, щоб рядок таблиці містив дані про деякий об’єкт, наприклад, книгу у бібліотеці чи товар у магазині.
2. Організувати додавання даних у таблицю. Для воду даних обов’язково використовувати UI-компонент зі стовбця (2) таблиці 1, відповідно до вашого варіанту.
3. Міститиме мітку, що відображатиме поточний час, мітка повинна оновлюватися кожну секунду. При додаванні елемента в одній з комірок (у відповідному стовбці) зберігатиметься поточний час, як час додавання елементу.
4. Міститиме JList справа від таблиці, та кнопки між ними, що дозволятимуть переміщувати рядки між таблицею та списком або копіювати рядки з таблиці до списку.
5. Для парних варіантів: при натисканні на кнопку “>” обраний у таблиці рядок переміщуватиметься до списку. При натисканні на кнопку “<” обраний у списку рядок переміщуватиметься назад то таблиці.
Для непарних варіантів: при натисканні на кнопку “>” обраний у таблиці рядок копіюватиметься до списку. При натисканні на кнопку “<” обраний у списку рядок видалятиметься зі списку.
6. У списку повинна відображатися інформація про переміщений або доданий об’єкт, що зазначена у стовбці (2) таблиці 1 відповідно до варіанта.
7. При обрані елемента у списку реалізувати логіку зі стовбця (3) таблиці 1.

Варіанти завдання

№ варіанта	(1)	(2)	(3)
1.	JComboBox (з можливістю додавання власного варіанта)	Всі поля об’єкта у форматі: <Ім’я стовбця1>: <Значення поля1>; Ім’я стовбця2>: <Значення поля2>; …. Ім’я стовбцяN>: <Значення поляN>	Для обраного елементу відповідний йому рядок у таблиці змінює колір тексту на зелений
2.	JComboBox (без можливості додавання	Об’єкт доданий о: <Час додавання об’єкта>	Фон кожного рядка таблиці, який був доданий після обраного елементу,

	власного варіанта)		змінюється на зелений.
3.	JSpinner (значення з діапазону)	Значення одного з поля об’єкту на ваш вибір, за виключенням дати додавання	Для обраного елементу відповідний йому рядок у таблиці змінює колір фону на жовтий
4.	JSlider	Значення одного з поля об’єкту на ваш вибір, за виключенням дати додавання, та дату додавання об’єкту	Колір тексту кожного рядка таблиці, який був доданий до обраного елементу, змінюється на червоний.
5.	JCheckbox	Всі поля об’єкта у форматі: <Ім’я стовбця1>: <Значення поля1>; Ім’я стовбця2>: <Значення поля2>; …. Ім’я стовбцяN>: <Значення поляN>	Для всіх рядків таблиці, у яких значення хоча б одного поля (комірки) співпадає зі значенням відповідного поля обраного об’єкта, колір тексту змінюється на зелений
6.	JRadioButton	Об’єкт доданий о: <Час додавання об’єкта>	Для всіх рядків таблиці, у яких значення обраного вами поля (комірки) співпадає зі значенням відповідного поля обраного об’єкта, колір фону змінюється на жовтий
7.	JSpinner (вибір варіантів)	Значення одного з поля об’єкту на ваш вибір, за виключенням дати додавання	Для обраного елементу відповідний йому рядок у таблиці змінює колір тексту на зелений
8.	JSpinner (дата)	Значення одного з поля об’єкту на ваш вибір, за виключенням дати додавання, та дату додавання об’єкту	Фон кожного рядка таблиці, який був доданий після обраного елементу, змінюється на зелений.
9.	JComboBox	Всі поля об’єкта у форматі: <Ім’я	Для обраного елементу відповідний

	(з можливістю додавання власного варіанта)	стовбця1>: <Значення поля1>; Ім’я стовбця2>: <Значення поля2>; …. Ім’я стовбцяN>: <Значення поляN>	йому рядок у таблиці змінює колір фону на жовтий
10.	JComboBox (без можливості додавання власного варіанта)	Об’єкт доданий о: <Час додавання об’єкта>	Колір тексту кожного рядка таблиці, який був доданий до обраного елементу, змінюється на червоний.
11.	JSpinner (значення з діапазону)	Значення одного з поля об’єкту на ваш вибір, за виключенням дати додавання	Для всіх рядків таблиці, у яких значення хоча б одного поля (комірки) співпадає зі значенням відповідного поля обраного об’єкта, колір тексту змінюється на зелений
12.	JSlider	Значення одного з поля об’єкту на ваш вибір, за виключенням дати додавання, та дату додавання об’єкту	Для всіх рядків таблиці, у яких значення обраного вами поля (комірки) співпадає зі значенням відповідного поля обраного об’єкта, колір фону змінюється на жовтий
13.	JCheckbox	Всі поля об’єкта у форматі: <Ім’я стовбця1>: <Значення поля1>; Ім’я стовбця2>: <Значення поля2>; …. Ім’я стовбцяN>: <Значення поляN>	Для обраного елементу відповідний йому рядок у таблиці змінює колір тексту на зелений
14.	JRadioButton	Об’єкт доданий о: <Час додавання об’єкта>	Фон кожного рядка таблиці, який був доданий після обраного елементу, змінюється на зелений.

15.	JSpinner (вибір варіантів)	Значення одного з поля об’єкту на ваш вибір, за виключенням дати додавання	Для обраного елементу відповідний йому рядок у таблиці змінює колір фону на жовтий
16.	JSpinner (дата)	Значення одного з поля об’єкту на ваш вибір, за виключенням дати додавання, та дату додавання об’єкту	Колір тексту кожного рядка таблиці, який був доданий до обраного елементу, змінюється на червоний.

Лабораторна робота №5. Потоки вводу-виводу.

Ціль: вивчити методики роботи з потоками вводу-виводу у Java на прикладі зберігання інформації про об’єкти у файлі.

Теоретичний матеріал

Потоки є основною абстракцією, яку Java використовує для обробки операцій вводу/виводу. Java надає уніфікований API для вводу та виводу, незалежно від того, надходять дані з файлів, клавіатури, мережі чи інших джерел. Потоки є односпрямованими, тобто дані надходять лише в одному напрямку: або в програму (вхідний потік), або з програми (вихідний потік).

Потоки Java поділяються на два основні типи:

- Потоки байтів: робота з двійковими даними (таких як зображення, аудіо та інші нетекстові дані).
- Потоки символів: обробляйте текстові дані, піклуючись про кодування та декодування символів.

Потоки байтів представлені абстрактними класами InputStream та OutputStream. Основні класи: FileInputStream, FileOutputStream, BufferedInputStream, BufferedOutputStream.

Потоки символів представлені абстрактними класами Reader та Writer. Основні класи: FileReader, FileWriter, BufferedReader, BufferedWriter.

Потоки байтів використовуються для читання та запису двійкових даних. Це може включати будь-який тип даних, який природно не читається людиною, наприклад зображення, аудіофайли, відеофайли або будь-які спеціальні двійкові формати.

Кожен байт розглядається як одиниця даних, що не потребує додаткових перетворень, і кодування або декодування не застосовується.

Переваги байтових потоків:

- Гнучкість: вони можуть обробляти будь-які типи даних.
- Ефективність для двійкових даних.

InputStream — це абстрактний клас для читання байтів. Він надає методи для читання по одному байту з джерела (наприклад, файлу, мережі або масиву).

Основні методи класу InputStream :

- read(): читає один байт і повертає його як int. Повертає -1, коли досягнуто кінця потоку.

- read(byte[] b): читає байти в масив і повертає кількість прочитаних байтів.
- close(): закриває потік, щоб звільнити системні ресурси.

OutputStream — це абстрактний клас для запису байтів. Він надає методи для запису по одному байту до пункту призначення (наприклад, файлу, мережі чи масиву).

Основні методи класу OutputStream :

- write(int b): Записує один байт.
- write(byte[] b): записує масив байтів.
- close(): закриває потік, забезпечуючи запис усіх даних і звільнення системних ресурсів.

Перейдемо до класів, що реалізують байтові потоки, та які можна використовувати безпосередньо.

FileInputStream та FileOutputStream використовується для читання та запису файлів у двійковому форматі.

ByteArrayInputStream та ByteArrayOutputStream використовується для читання та запису даних у пам’яті з/до байтового масиву.

Класи BufferedInputStream та BufferedOutputStream:

- додають буферизацію до вхідних і вихідних потоків, підвищуючи продуктивність за рахунок зменшення кількості разів, коли дані зчитуються або записуються в базове джерело даних.
- класи зберігають дані в буфері, що дозволяє читати або записувати великі фрагменти даних одночасно.

Приклад використання класів FileInputStream та FileOutputStream:

```
FileOutputStream outputStream = new FileOutputStream("output.bin");
outputStream.write(65);  // 'A' у кодуванні ASCII
outputStream.close();

FileInputStream inputStream = new FileInputStream("output.bin");
int byteData = inputStream.read();
while (byteData != -1) {
  // Зчитування та обробка даних побайтово
  System.out.println(byteData);
  byteData = inputStream.read();
}
inputStream.close();
```

Використання BufferedOutputStream:

```
import java.io.*;

public class BufferedOutputExample {
    public static void main(String[] args) {
```

```
        try {
            FileOutputStream fileOutput =
              new FileOutputStream("output.txt");
            BufferedOutputStream bufferedOutput =
              new BufferedOutputStream(fileOutput);

            String data = "Buffered output example.";
            bufferedOutput.write(data.getBytes());
            bufferedOutput.flush();
            bufferedOutput.close();
            fileOutput.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Використання BufferedInputStream:

```
import java.io.*;

public class BufferedInputExample {
    public static void main(String[] args) {
        try {
            FileInputStream fileInput = new
FileInputStream("input.txt");
            BufferedInputStream bufferedInput =
              new BufferedInputStream(fileInput);
            int data;
            while ((data = bufferedInput.read()) != -1) {
                System.out.print((char) data);
            }
            bufferedInput.close();
            fileInput.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Використання DataOutputStream:

```
import java.io.*;

public class BufferedInputExample {
    public static void main(String[] args) {
        try {
            FileOutputStream outputStream =
              new FileOutputStream("output.bin");
            DataOutputStream dataStream =
              new DataOutputStream(outputStream);
            dataStream.writeUTF("This is a string");
            dataStream.flush();
            dataStream.close();
            outputStream.close();
```

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

При роботі з потоками можуть виникати виключні ситуації, які пов’язані з вводом/виводом. Розглянемо основні класи, що представляють виключні ситуації пов’язані з потоками.

IOException є суперкласом для всіх винятків, пов’язаних із вводом/виводом. Це сигналізує про те, що операція вводу/виводу сталася невдалою або була перервана. Ця виключна ситуація може статися під час читання, запису або закриття потоків.

FileNotFoundException — це підклас класу IOException, це виключення виникає, коли спроба відкрити файл, позначений вказаним шляхом, не вдається. Таке часто трапляється, якщо файл не існує або програма не має відповідних дозволів.

EOFException — це підклас класу IOException, цей виняток виникає, коли досягається неочікуваний кінець файлу під час спроби читання з потоку.

UnsupportedEncodingException — це також підклас класу IOException. Цей виняток виникає, коли певне кодування символів не підтримується.

Серіалізація — це процес перетворення об'єкта в потік байтів. Це дозволяє зберегти об’єкт у файл, надіслати його по мережі або перенести між різними віртуальними машинами Java (JVM). Після серіалізації об’єкт можна зберегти або передати, а потім реконструювати за допомогою десеріалізації, яка повертає процес назад, перетворюючи потік байтів назад на об’єкт.

У Java об’єкт серіалізується, якщо він реалізує інтерфейс Serializable, який є інтерфейсом маркером (це означає, що він не має методів для реалізації). Без реалізації цього інтерфейсу об’єкт не можна серіалізувати. Серіалізований об’єкт може бути десеріалізований у точну копію оригіналу.

Серіалізація виконується за допомогою класу ObjectOutputStream, який записує об’єкт у вихідний потік, зазвичай у файл або надсилається через мережу. Десеріалізація (зворотний процес) використовує клас ObjectInputStream для читання потоку байтів і реконструкції об'єкта.

Коли Java серіалізує об’єкт, вона автоматично серіалізує всі поля (як примітивні, так і посилальні типи), які є частиною об’єкта, якщо вони не позначені як тимчасові.

- Примітивні типи (наприклад, int, float, boolean) безпосередньо записуються в потік байтів.
- об’єктні типи серіалізуються рекурсивно, якщо вони реалізують Serializable. Якщо будь-яке поле об’єкта не можна серіалізувати, серіалізація не вдасться, викинувши NotSerializableException.

Приклад серіалізації:

```
import java.io.Serializable;
class Person implements Serializable {
    private String name;
    private int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

public class SerializationExample {
    public static void main(String[] args) {
        String filePath = "path/to/your/image.png";
        try {
            Person obj = new Person("John", 30);
            FileOutputStream fileOut =
              new FileOutputStream("object.ser");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(obj);
            out.close();
            fileOut.close();
        } catch (IOException e) {
            System.err.println("Error while writing to file ");
        }
    }
```

Приклад десеріалізації:

```
import java.io.Serializable;
class Person implements Serializable {
    private String name;
    private int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

public class SerializationExample {
    public static void main(String[] args) {
        String filePath = "path/to/your/image.png";
        try {
            FileInputStream fileIn = new FileInputStream("object.ser");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            Person obj = (Person) in.readObject();
            in.close();
            fileIn.close();

        } catch (IOException e) {
            System.err.println("Error while reading file ");
        }
        catch (ClassNotFoundException y) {
            System.err.println("Class was not found");
```

```
        }
    }
}
```

Під час десеріалізації віртуальна машина Java (JVM) повинна знайти клас об’єкта, який десеріалізується. Якщо визначення класу не знайдено, виникає виключення ClassNotFoundException. Це може трапитися, якщо:

- клас було видалено.
- клас недоступний у поточному середовищі, наприклад, під час десеріалізації об’єктів, надісланих з іншої JVM.

Статичні поля не серіалізуються бо серіалізація — це операція на рівні екземпляра.

Клас також може містити звичайні поля, які не будуть серіалізуватися. Це так звані тимчасові поля. Ключове слово **transient** використовується для позначення полів, які не слід серіалізувати. Коли поле оголошено як тимчасове, воно пропускається під час процесу серіалізації, тобто його значення не включається в потік байтів. Тимчасові поля потрібні щоб уникати при серіалізації зберігання конфіденційних даних (наприклад, паролі, криптографічні ключі), які не можна зберігати чи передавати або виключити непостійні поля, значення яких є тимчасовим або похідним від інших полів.

Приклад використання тимчасових полів:

```
import java.io.*;
import java.time.LocalDate;
import java.time.Period;

public class Person implements Serializable {
    private static final long serialVersionUID = 1L;
    private String name;
    private LocalDate birthDate;
    private transient int age;
    public Person(String name, LocalDate birthDate) {
        this.name = name;
        this.birthDate = birthDate;
        age = calculateAge();
    }
    private int calculateAge() {
        return Period.between(birthDate, LocalDate.now()).getYears();
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        if (age == 0) {
            age = calculateAge();
        }
        return age;
    }
}
```

Також є можливість використовувати спеціальну серіалізацію щоб мати детальний контроль над тим, як серіалізується та десеріалізується об’єкт. Це особливо корисно в сценаріях, коли потрібно змінити типову поведінку серіалізації, яку забезпечує вбудований механізм серіалізації Java. Спеціальна серіалізація може допомогти досягти різних цілей, наприклад, оптимізувати продуктивність, обробити конфіденційні дані або підтримувати зворотну сумісність.

Щоб реалізувати спеціальну серіалізацію, можна перевизначити два певних методи у своєму класі. Ці методи викликаються під час процесів серіалізації та десеріалізації, що дозволяє точно визначити, як стан об’єкта записується та зчитується з вихідного потоку.

Це методи:

- private void writeObject(ObjectOutputStream out) throws IOException
- private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException

Приклад реалізації спеціальної серіалізації:

```
import java.io.*;
import java.util.Base64;
public class User implements Serializable {
    private static final long serialVersionUID = 1L;
    private String username;
    private transient String password; // Original password (not
serialized)
    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }
    private String encryptPassword(String password) {
        return Base64.getEncoder().encodeToString(password.getBytes());
    }
    private String decryptPassword(String encryptedPassword) {
        return new
String(Base64.getDecoder().decode(encryptedPassword));
    }
    public String getUsername() {
        return username;
    }
    public String getPassword() {
        // Повертає дійсний пароль після десеріалізації
        return password;
    }
    private void writeObject(ObjectOutputStream out) throws IOException
{
        out.defaultWriteObject();

        String encryptedPassword = encryptPassword(password);
        // Записати закодований пароль
        out.writeObject(encryptedPassword);
    }
```

```
    private void readObject(ObjectInputStream in) throws IOException,
ClassNotFoundException {
        in.defaultReadObject();

        String encryptedPassword = (String) in.readObject();
        // Розшифрувати пароль
        this.password = decryptPassword(encryptedPassword);
    }
}
```

Завдання

Додати до програми, щ обула створенна під час виконання лабораторної роботи №4 можливість зберігати дані у файлі та зчитувати дані з файлу.

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ
ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ
з дисципліни «Крос-платформне програмування»
(для студентів напряму підготовки
121 «Інженерія програмного забезпечення»)

Укладач:
Дьомін Максим Костянтинович

Редактор *М.К.Дьомін*
Техн. редактор *М.К.Дьомін*
Оригінал - макет *М.К.Дьомін*

Підписано до друку _____
Формат $60\times84\,{}^{1}\!/_{16}$. Папір типограф. Гарнитура *Times*.
Друк офсетний. Ум. друк. арк.___. Обл.-вид.арк._____.
Тираж ___ прим. Вид. №_____. Замовл. №_____. Ціна договірна.

**Видавництво Східноукраїнського національного університету
імені Володимира Даля**

Свідоцтво про реєстрацію:
Адреса університета: вул. Іоанна Павла II, 17
м. Київ, 01042, Україна
e-mail: vidavnictvoSNU.ua@gmail.com.