



**Звіт**  
**до лабораторної роботи № 4**

**з дисципліни: «Кросплатформні засоби програмування»**  
**на тему: «Спадкування та інтерфейси»**

**Варіант 11**

**Виконав:**

**ст.гр. КІ-36**

**Басько С.І.**

**Прийняв:**

**Іванов Ю.С.**

**Львів 2022**

# СПАДКУВАННЯ ТА ІНТЕРФЕЙСИ

**Мета:** ознайомитися з спадкуванням та інтерфейсами у мові Java.

## ТЕОРЕТИЧНІ ВІДОМОСТІ

### Спадкування

Спадкування в ООП призначене для розширення функціональності існуючих класів шляхом утворення нових класів на базі вже існуючих. У Java реалізована однокоренева архітектура класів згідно якої всі класи мають єдиного спільного предка (кореневий клас в ієрархії класів) – клас `Object`. Решта класів мови Java утворюються шляхом успадковування даного класу. Будь-яке спадкування у мові Java є відкритим, при цьому аналогів захищеному і приватному спадкуванню мови C++ не існує. На відміну від C++ у Java можливе спадкування лише одного базового класу (множинне спадкування відсутнє). Спадкування реалізується шляхом вказування ключового слова `class` після якого вказується назва підкласу, ключове слово `extends` та назва суперкласу, що розширюється у новому підкласі. Синтаксис реалізації спадкування:

```
class Підклас extends Суперклас {  
    Додаткові поля і методи  
}
```

### Поліморфізм

Механізм поліморфізму забезпечує можливість присвоєння об'єктним змінним суперкласу об'єктів похідних класів та звертання з-під цих змінних до перевизначених у підкласі членів суперкласу. У Java всі об'єктні змінні є поліморфними. Поліморфізм реалізується за допомогою механізму динамічного (пізнього) зв'язування, який полягає у тому, що вибір методу, який необхідно викликати, відбувається не на етапі компіляції, а під час виконання програми. Для глибшого розуміння поліморфізму розглянемо покроково виклик методу класу:

1. Компілятор визначає об'явлений тип об'єкту і ім'я методу та нумерує всі методи з однаковою назвою у класі та всі загальнодоступні методи з такою ж назвою у його суперкласах.
2. Компілятор визначає типи параметрів, що вказані при виклику методу. Якщо серед усіх методів з вказаним іменем є лише один метод, типи параметрів якого співпадають з вказаним, то відбувається його виклик. Цей процес називається дозволом перевантаження. Якщо компілятор не знаходить жодного методу з підходящим набором параметрів або в результаті перетворення типів виявлено кілька методів, що відповідають даному виклику, то видається повідомлення про помилку.
3. Якщо метод є приватним, статичним, фінальним або конструктором, то для нього застосовується механізм статичного зв'язування. Механізм статичного зв'язування передбачає визначення методу, який необхідно викликати, на етапі компіляції. В протилежному випадку метод, що необхідно викликати, визначається по фактичному типу неявного параметру (динамічне зв'язування).
4. Якщо для виклику методу використовується динамічне зв'язування, то віртуальна машина повинна викликати версію методу, що відповідає фактичному типу об'єкту на який посилається об'єктна змінна.

Оскільки на пошук необхідного методу потрібно багато часу, то віртуальна машина заздалегідь створює для кожного класу таблицю методів, в якій перелічуються сигнатури всіх методів і фактичні методи, що підлягають виклику. При виклику методу віртуальна машина просто переглядає таблицю методів. Якщо відбувається виклик методу з суперкласу за допомогою ключового слова `super`, то при виклику методу переглядається таблиця методів суперкласу неявного параметру.

## Хід роботи

### 5. Сенсорний екран

```
package vitkor.ki.objects;

import vitkor.ki.objects.enums.BacklightType;
import vitkor.ki.objects.enums.Color;
import vitkor.ki.objects.enums.MonitorStatus;
import vitkor.ki.objects.parts.Brightness;
import vitkor.ki.objects.parts.Image;
import vitkor.ki.objects.parts.Resolution;

import java.io.FileNotFoundException;
import java.io.PrintWriter;

public abstract class Display {

    private static final String OUTPUT_FILE = "OUTPUT.txt";

    private BacklightType backlightType;
    private Resolution displayResolution;
    private Double diagonal;
    private Color frameColor;

    private Brightness brightness;
    private MonitorStatus currentStatus = MonitorStatus.ON;

    private PrintWriter printWriter;

    public Display(Resolution displayResolution, Double diagonal,
                  BacklightType backlightType,
                  Brightness brightness) {
        this.displayResolution = displayResolution;
        this.diagonal = diagonal;
        this.backlightType = backlightType;
        this.frameColor = Color.BLACK;
        this.brightness = brightness;
    }

    public Display(Resolution displayResolution, Double diagonal,
                  BacklightType backlightType,
                  Brightness brightness,
                  Color frameColor) {
        this.backlightType = backlightType;
        this.displayResolution = displayResolution;
        this.diagonal = diagonal;
        this.frameColor = frameColor;
        this.brightness = brightness;
    }

    /**
     * Turning on monitor, status moved to ON
     */
    public void turnOnMonitor() {
        try {
            printWriter = new PrintWriter(OUTPUT_FILE);
            this.currentStatus = MonitorStatus.ON;
            printWriter.println("Monitor successfully turned ON");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    /**
```

```

    * Turning off monitor, status moved to OFF
    */
    public void turnOffMonitor() {
        if (MonitorStatus.ON.equals(currentStatus)) {
            printWriter.println("Turning off monitor");
            printWriter.flush();
            printWriter.close();
            currentStatus = MonitorStatus.OFF;
        }
    }

    /**
     * Increasing of brightness value
     *
     * @param value how much to increase
     */
    public void increaseBrightness(Integer value) {
        int newBrightnessValue = brightness.getCurrentValue() + value;
        int brightnessMaxValue = brightness.getMaxValue();
        brightness.setCurrentValue(Math.min(brightnessMaxValue,
newBrightnessValue));
        printWriter.println("Brightness was increased");
        printWriter.println("Current brightness is: " +
brightness.getCurrentValue());
    }

    /**
     * Decreasing of brightness value
     *
     * @param value how much to decrease
     */
    public void decreaseBrightness(Integer value) {
        int newBrightnessValue = brightness.getCurrentValue() - value;
        int brightnessMinValue = brightness.getMinValue();
        brightness.setCurrentValue(Math.max(brightnessMinValue,
newBrightnessValue));
        printWriter.println("Brightness was decreased");
        printWriter.println("Current brightness is: " +
brightness.getCurrentValue());
    }

    /**
     * Displaying of an image
     *
     * @param image - image with content
     */
    public void displayImage(Image image) {
        printWriter.println("Image displayed: " + image.getContent());
    }

    /**
     * Show monitor information
     */
    public void viewMonitorInformation() {
        if (MonitorStatus.ON.equals(currentStatus)) {
            printWriter.println("Display Resolution: " +
displayResolution.getWidth() + "x" + displayResolution.getHeight());
            printWriter.println("Display diagonal: " + diagonal + "\"");
            printWriter.println("Display type of backlight: " + backlightType);
        }
    }

    //GETTERS & SETTERS

    public Resolution getDisplayResolution() {
        return displayResolution;
    }

```

```

    }

    public void setDisplayResolution(Resolution displayResolution) {
        this.displayResolution = displayResolution;
    }

    public Double getDiagonal() {
        return diagonal;
    }

    public void setDiagonal(Double diagonal) {
        this.diagonal = diagonal;
    }

    public BacklightType getBacklightType() {
        return backlightType;
    }

    public void setBacklightType(BacklightType backlightType) {
        this.backlightType = backlightType;
    }

    public Color getFrameColor() {
        return frameColor;
    }

    public void setFrameColor(Color frameColor) {
        this.frameColor = frameColor;
    }

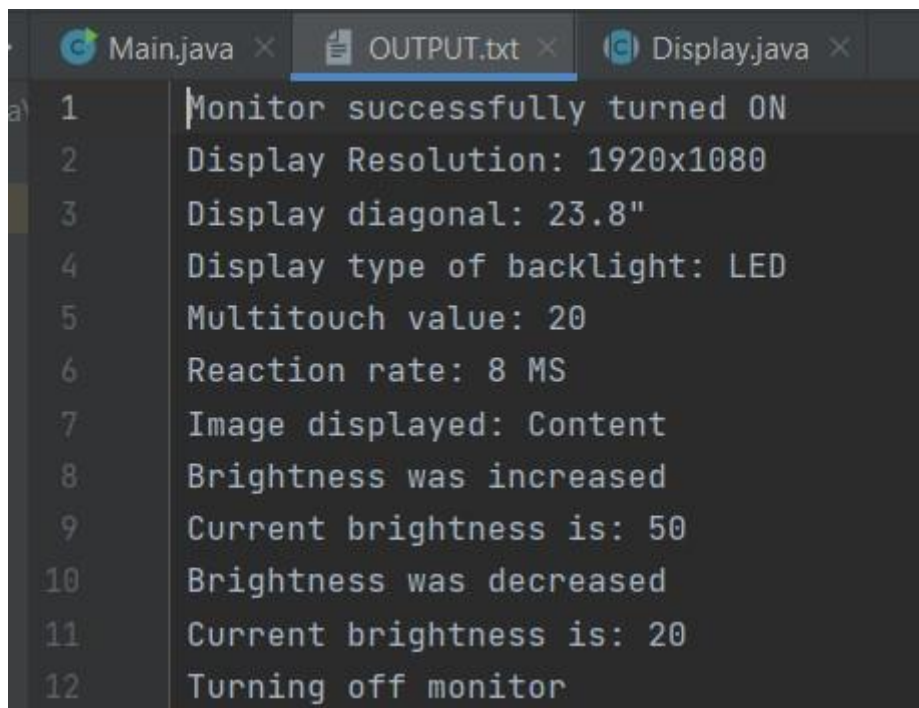
    public Brightness getBrightness() {
        return brightness;
    }

    public void setBrightness(Brightness brightness) {
        this.brightness = brightness;
    }

    protected PrintWriter getPrintWriter() {
        return printWriter;
    }

    public MonitorStatus getCurrentStatus() {
        return currentStatus;
    }
}

```



```
1 Monitor successfully turned ON
2 Display Resolution: 1920x1080
3 Display diagonal: 23.8"
4 Display type of backlight: LED
5 Multitouch value: 20
6 Reaction rate: 8 MS
7 Image displayed: Content
8 Brightness was increased
9 Current brightness is: 50
10 Brightness was decreased
11 Current brightness is: 20
12 Turning off monitor
```

**Висновок:** Я ознайомився з спадкуванням і інтерфейсами в Java.