Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Звіт

з лабораторної роботи № 3 з дисципліни
«Мова програмування Java»

«ООР»

Варіант 3.1

Виконала ЗПІ-зп41 Федоренко О. Л.

Перевірила Орленко С. П.

Київ 2025

1. Завдання (3.1)

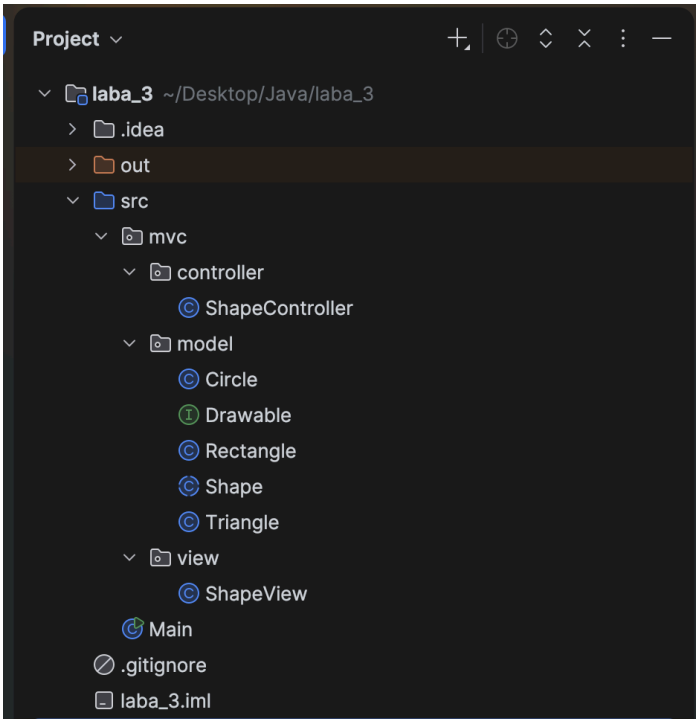
Напишіть консольний додаток, використовуючи архітектурний шаблон MVC, який:

- описує інтерфейс **Drawable** з методом побудови фігури *draw()*;
- описує абстрактний клас **Shape**, який реалізує інтерфейс **Drawable** і містить поле ***shapeColor*** типу **String** для кольору фігури і конструктор для його ініціалізації, абстрактний метод обчислення площі фігури *calcArea()* і перевизначений метод *toString()*;
- описує класи Rectangle, Triangle, Circle, які успадковуються від класу Shape і реалізують метод calcArea (), а також перевизначають метод toString ();
- створює набір даних типу **Shape** (масив розмірністю не менш 10 елементів);
- обробляє масив:
  - відображає набір даних;
  - обчислює сумарну площу всіх фігур набору даних;
  - обчислює сумарну площу фігур заданого виду;
  - впорядковує набір даних щодо збільшення площі фігур, використовуючи об'єкт інтерфейсу **Comparator**;
  - впорядковує набір даних за кольором фігур, використовуючи об'єкт інтерфейсу **Comparator**.

Значення для ініціалізації об'єктів вибираються з заздалегідь підготовлених даних (обраних випадковим чином або по порядку проходження).

2. Лістинг програмного коду

2.1. Структура проєкту



2.2. Модель даних (Model)

- Interface Drawable

```
package mvc.model;

public interface Drawable {
    void draw();
}
```

- Абстрактний клас Shape

```
package mvc.model;

public abstract class Shape implements Drawable {
    private String shapeColor;

    public Shape(String shapeColor) {
        this.shapeColor = shapeColor;
    }

    public String getShapeColor() {
        return shapeColor;
    }

    public abstract double calcArea();
```

```
    @Override
    public String toString() {
        return String.format("Color: %s, Area: %.2f",
shapeColor, calcArea());
    }

    @Override
    public void draw() {
        System.out.println(this.toString());
    }
}
```

- Клас Circle

```
package mvc.model;

public class Circle extends Shape {
    private double radius;

    public Circle(String color, double radius) {
        super(color);
        this.radius = radius;
    }

    @Override
    public double calcArea() { return Math.PI * radius *
radius; }
}
```

- Клас Rectangle

```
package mvc.model;

public class Rectangle extends Shape {
    private double width, height;

    public Rectangle(String color, double width, double
height) {
        super(color);
        this.width = width;
        this.height = height;
    }

    @Override
    public double calcArea() { return width * height; }
}
```

- Клас Triangle

```
package mvc.model;

public class Triangle extends Shape {
    private double base, height;

    public Triangle(String color, double base, double height)
{
        super(color);
        this.base = base;
        this.height = height;
    }

    @Override
    public double calcArea() { return 0.5 * base * height; }
}
```

2.3. Компонент відображення (View)

- Клас ShapeView – відповідає за виведення результатів у консоль

```
package mvc.view;

import mvc.model.Shape;

public class ShapeView {
    public void printMessage(String message) {
        System.out.println(message);
    }

    public void printShapes(Shape[] shapes) {
        for (Shape shape : shapes) {
            shape.draw();
        }
    }
}
```

2.3. Компонент керування (Controller)

- Клас ShapeController – містить бізнес-логіку: створення масиву, сортування та обчислення.

```
package mvc.controller;

import mvc.model.*;
import mvc.view.ShapeView;
import java.util.Arrays;
import java.util.Comparator;
```

```
public class ShapeController {
    private Shape[] model;
    private ShapeView view;

    public ShapeController() {
        this.view = new ShapeView();
        this.model = new Shape[]{
                new Rectangle("Red", 5, 4),
                new Circle("Blue", 3),
                new Triangle("Green", 4, 6),
                new Rectangle("Yellow", 2, 8),
                new Circle("White", 5),
                new Triangle("Black", 3, 3),
                new Rectangle("Red", 10, 2),
                new Circle("Green", 2.5),
                new Triangle("Blue", 7, 2),
                new Rectangle("Purple", 4, 4)
        };
    }

    public void run() {
        view.printMessage("--- Список фігур ---");
        view.printShapes(model);

        view.printMessage(String.format("\nСумарна площа:
%.2f", sumArea(model)));

        view.printMessage("\n--- Сортування за площею ---");
        Arrays.sort(model,
Comparator.comparingDouble(Shape::calcArea));
        view.printShapes(model);

        view.printMessage("\n--- Сортування за кольором ---");
        Arrays.sort(model,
Comparator.comparing(Shape::getShapeColor));
        view.printShapes(model);
    }

    private double sumArea(Shape[] shapes) {
        double sum = 0;
        for (Shape s : shapes) sum += s.calcArea();
        return sum;
    }
}
```

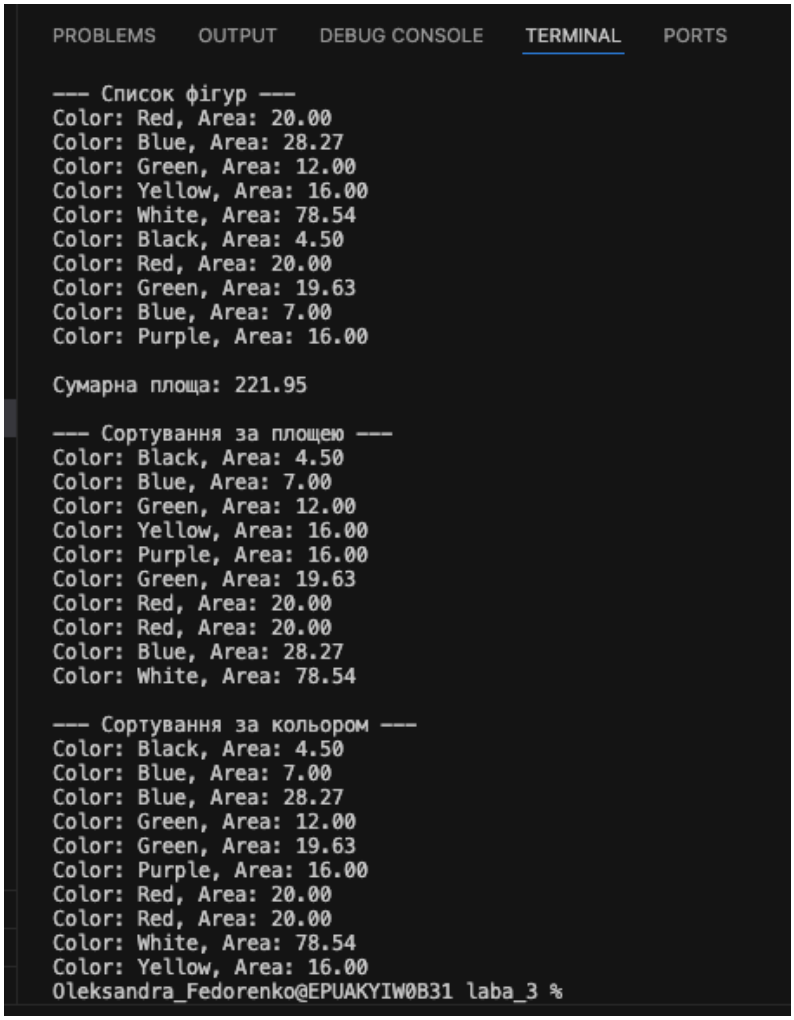
2.4. Точка входу

- *Головний клас Main*

```
import mvc.controller.ShapeController;

public class Main {
    public static void main(String[] args) {
        ShapeController controller = new ShapeController();
        controller.run();
    }
}
```

3. Скріншоти виконання програми



4. Висновки

У ході виконання лабораторної роботи було розроблено консольний додаток на мові Java, що демонструє роботу з об’єктно-орієнтованою моделлю даних. Під час роботи було опановано

архітектурний шаблон **MVC (Model-View-Controller)**, що дозволило відокремити бізнес-логіку програми від механізмів виведення даних. Реалізовано ієрархію класів фігур на основі абстрактного класу та інтерфейсу . Практично застосовано механізми сортування об'єктів за допомогою інтерфейсу Comparator для впорядкування набору даних за площею та кольором фігур. ShapeDrawable