Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

ПРАКТИЧНА РОБОТА № 3

З дисципліни «Інженерія програмного забезпечення»

на тему «СТРУКТУРНІ ШАБЛОНИ ПРОЕКТУВАННЯ ПЗ. COMPOSITE, DECORATOR, PROXY, FLYWEIGHT, ADAPTER, BRIDGE, FACADE.»

Виконав:
студент групи ІО-42
Куліков М. М.
Залікова: 4214

Перевірив:
Ст. викладач кафедри ОТ
Васильєва М. Д.

Київ – 2025

Практична робота №3

Тема: «Структурні шаблони проєктування ПЗ. Composite, Ecorator, Proxy, Flyweight, Adapter, Bridge, Facade.».

Мета: Ознайомлення з видами шаблонів проектування ПЗ. Вивчення структурних шаблонів. Отримання базових навичок з застосування шаблонів Composite, Decorator, Proxy, Flyweight, Adapter, Bridge, Facade.

Виконання роботи:

Завдання 1

1. Ознайомитись з призначенням та видами шаблонів проектування ПЗ. Вивчити класифікацію шаблонів проектування ПЗ. Знати назви шаблонів, що відносяться до певного класу.
2. Вивчити структурні шаблони проектування ПЗ. Знати загальну характеристику структурних шаблонів та призначення кожного з них.
3. Детально вивчити структурні шаблони проектування Composite, Decorator, Proxy, Flyweight, Adapter, Bridge, Facade. Для кожного з них:
    - вивчити шаблон, його призначення, альтернативні назви, випадки, коли його застосування є доцільним та результати такого застосування;
    - знати особливості реалізації шаблону;
    - вільно володіти структурою шаблону, призначенням його класів та відносинами між ними;
    - вміти розпізнавати шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, що реалізують шаблон.
4. В підготованому проєкті створити програмний пакет work3. В пакеті розробити інтерфейси і класи, що реалізують завдання 1 та 2 (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.3). В класах, що розробляються, повністю реалізувати методи, пов'язані з функціюванням Шаблону. Методи, що реалізують бізнес-логіку, закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи. Приклад реалізації бізнес-

методу: void draw(int x, int y) { System.out.println(“Метод draw з параметрами x=”+x+” y=”+y); }

5. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

Визначимо варіант для індивідуальних завдань: 4214%12 = 2, 4214%11 = 1.

Варіант 2 до завдання 1:

Клас GameElement, який представляє елемент ігрового простору. Необхідно розширити функціональність так, щоб:

- Можна було створювати простий елемент (наприклад, одиночний об’єкт на рівні).

Кожний елемент має розмір (ширина та висота) у умовних одиницях.

- Можна було створювати композитний елемент (групу елементів), у якому:
  - Атрибути розміру та площа обчислюються динамічно на основі вкладених елементів.
  - Можливі вкладені композитні елементи (багаторівнева ієрархія).
- Можна було отримати площу елемента:
  - Для простого елемента – площа = ширина × висота.
  - Для композитного елемента – площа = сума площ усіх вкладених елементів або мінімальна обгортка, що охоплює всі піделементи (залежно від реалізації).

Вимоги

- Не змінювати існуючий клас GameElement. Реалізувати можливість гнучкого додавання нових типів елементів у майбутньому.
- Продемонструвати у методі main кілька варіантів роботи з елементами:
  - окремий простий елемент,
  - група з кількох простих елементів,
  - група з вкладеними групами (багаторівнева структура).

Код до завдання 1:

Вміст файлу main.java:

```
import Composite.*;
/**
 * Головний клас для демонстрації роботи патерну Composite
 * з ігровими елементами.
 */
public class Main {
    public static void main(String[] args) {
        // Один елемент
        System.out.println(">-- One Element:");
        GameElement tree = new SimpleElement("Tree", 2, 3);
        tree.print(" ");
        // Група з кількох простих елементів
        System.out.println(">-- Group of simple elements:");
        CompositeElement house = new CompositeElement("House");
        house.addChild(new SimpleElement("Wall", 5, 3));
        house.addChild(new SimpleElement("Door", 1, 2));
        house.addChild(new SimpleElement("Window", 1, 1));
        house.print(" ");

        // Багаторівнева структура
        System.out.println(">-- Many levels:");
        CompositeElement level = new CompositeElement("Game Level");
        level.addChild(tree);
        level.addChild(house);

        CompositeElement forest = new CompositeElement("Dark Forest");
        forest.addChild(new SimpleElement("Bush", 1, 1));
        forest.addChild(level);

        forest.print("");
    }
}
```

Вміст файлу GameElement.java:

```
package Composite;

/**
*Абстрактний клас Компонент, який використовують інші об'єкти
*призначений для визначення загальних методів для роботи з об'єктами
 **/
abstract public class GameElement {
    /** Ім'я елемента, яке використовується для ідентифікації та друку */
    protected String name;
    /**
     * Конструктор компонента.
     * @param name ім'я елемента
     */
    public GameElement(String name) { this.name = name; }
    /**
     * Повертає ім'я елемента.
     * @return ім'я елемента
     */
    public String getName() { return name; }
```

```
    /**
     * Обчислює площу елемента.
     * Для простих елементів це width*height,
     * для композитів — сума площ усіх дочірніх елементів.
     * @return площа елемента
     */
    public abstract double getArea();

    /**
     * Друкує інформацію про елемент із заданим відступом.
     * Для композитів виконується рекурсивний виклик для дочірніх елементів.
     * @param indent рядок відступу для форматованого друку
     */
    public abstract void print(String indent);
}
```

Вміст файлу CompositeElement.java:

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

/**
 * Клас, який є композитним елементом
 * Може містити у собі інші GameElement (або інші композити)
 * Використовується для побудови ієрархічних структур сцени
 */
public class CompositeElement extends GameElement {
    /** Список дочірніх елементів */
    private List<GameElement> children =  new ArrayList<GameElement>();
    /**
     * Конструктор.
     * @param name ім'я композитного елемента
     */
    public CompositeElement(String name) { super(name); }
    /**
     * Додає дочірній елемент до композиту.
     * @param child елемент для додавання
     */
    public void addChild(GameElement child) { children.add(child); }
    /**
     * Обчислює загальну площу композитного елемента.
     * Сума площ усіх дочірніх елементів.
     * @return загальна площа
     */
    @Override
    public double getArea() {
        double total = 0;
        for (GameElement el : children) {
            total += el.getArea();
        }
        return total;
    }

    /**
     * Друкує композит і його дочірні елементи.
     * @param indent рядок відступу для форматованого друку
     */
    @Override
    public void print(String indent) {
```

```
        System.out.println("+ " + name + " (total area=" + getArea() + ")");
        for (GameElement el : children) {
            el.print(indent + "  "); }    } }
```

Вміст файлу SimpleElement.java:

```
package Composite;
/**
 * Клас "Листок", який немає підлеглих
 * Використовується для простих об'єктів у системі
 */
public class SimpleElement extends GameElement {
    private double width;
    private double height;

    /**
     * Конструктор.
     * @param name ім'я елемента
     * @param width ширина елемента
     * @param height висота елемента
     */
    public SimpleElement(String name, double width, double height) {
        super(name);
        this.width = width;
        this.height = height;
    }

    /**
     * Обчислює площу елемента.
     * @return площа (width * height)
     */
    @Override
    public double getArea() { return width * height;  }

    /**
     * Друкує інформацію про елемент.
     * Формат: "- name (width x height, area=...)".
     * @param indent рядок відступу для форматованого друку
     */
    @Override
    public void print(String indent) {
        System.out.println(indent + "- " + name + " (" + width + "x" + height +
                "=" + getArea() + ")");
    }
}
```

Результат виконання програми:

```
>-- One Element:
 - Tree (2.0x3.0=6.0)
>-- Group of simple elements:
+ House (total area=18.0)
   - Wall (5.0x3.0=15.0)
   - Door (1.0x2.0=2.0)
   - Window (1.0x1.0=1.0)
>-- Many levels:
+ Dark Forest (total area=25.0)
  - Bush (1.0x1.0=1.0)
+ Game Level (total area=24.0)
    - Tree (2.0x3.0=6.0)
+ House (total area=18.0)
      - Wall (5.0x3.0=15.0)
      - Door (1.0x2.0=2.0)
      - Window (1.0x1.0=1.0)
```

Рисунок 1 – результат виконання програми згідно першого завдання

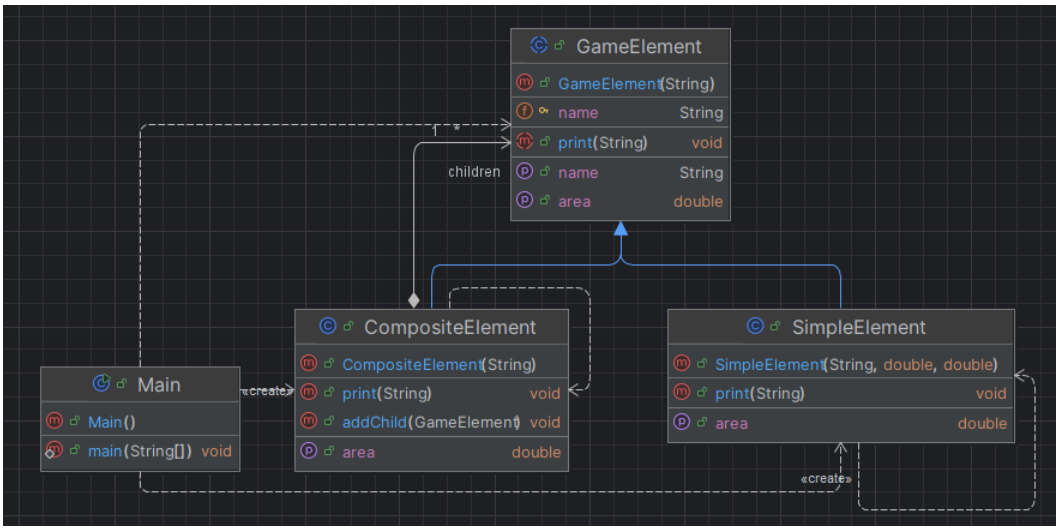

Рисунок 2 – Побудована в IntelliJ IDEA UML діаграма для першого завдання

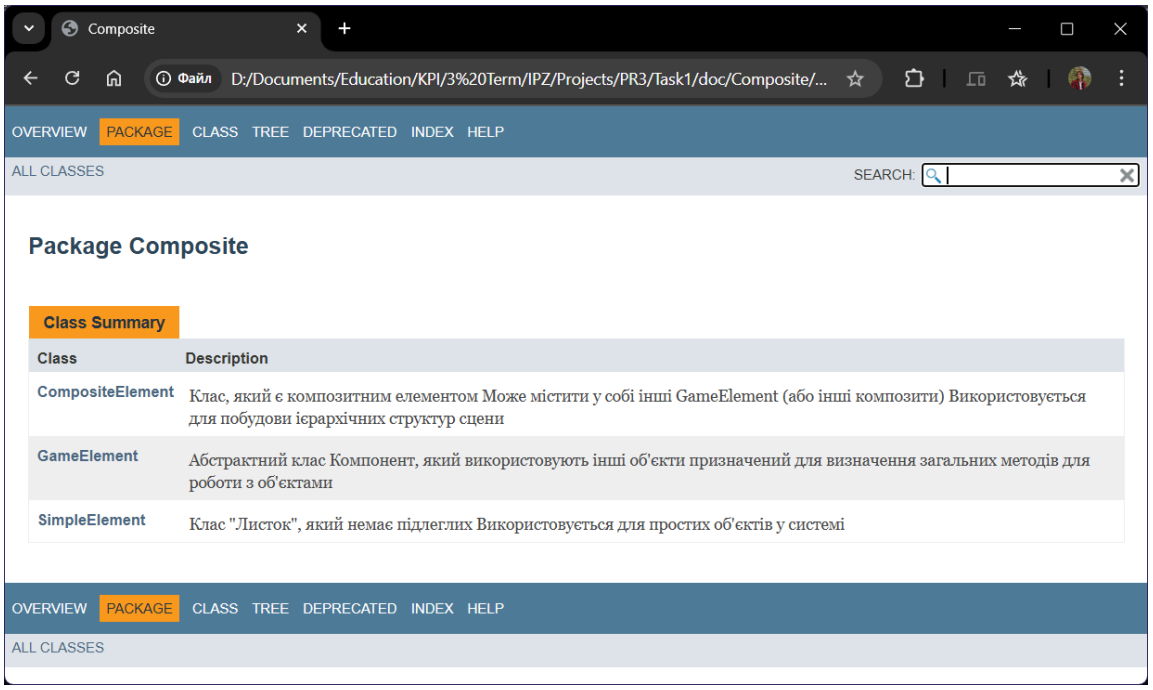

Рисунок 3 – Створена документація у форматі веб сторінки

Варіант 1 до завдання 2:

Визначити специфікації класів, які подають графічні об'єкти у редакторі растрової графіки – примітиви (точка) та їх композиції (прямокутне зображення). Забезпечити ефективне використання пам'яті при роботі з великою кількістю графічних об'єктів. Реалізувати метод рисування графічного об'єкту.

Код до завдання 2:

Вміст файлу GraphicObject.java:

```
package Flyweight;
/**
 * Інтерфейс для графічних об'єктів.
 * Визначає метод для відображення об'єкта на екрані.
 */
public interface GraphicObject {
    /**
     * Малює графічний об'єкт у заданих координатах.
     * @param x координата по горизонталі
     * @param y координата по вертикалі
     */
    void draw(int x, int y);
}
```

Вміст файлу Coords.java:

```
package Flyweight;

/**
 * Клас для збереження координат пікселя
 * Використовується для відображення пікселів у сцені
 */
public class Coords {
    /** Координата x */
    int x;
    /** Коодината y */
    int y;

    /**
     * Конструктор.
     * @param x координата по горизонталі
     * @param y координата по вертикалі
     */
    public Coords(int x, int y) {
        this.x = x;
        this.y = y;   } }
```

Вміст файлу Pixel.java:

```
package Flyweight;
/**
 * Клас, що представляє піксель.
 * Реалізує патерн Flyweight — зберігає стан спільного використання (color),
 * а координати передаються при малюванні.
```

```
 */
public class Pixel implements GraphicObject {
    /** Колір пікселя (спільний стан) */
    private final String color;
    /**
     * Конструктор.
     * @param color колір пікселя
     */
    public Pixel(String color) { this.color = color; }

    /**
     * Малює піксель у вказаних координатах.
     * @param x координата по горизонталі
     * @param y координата по вертикалі
     */
    @Override
    public void draw(int x, int y) { System.out.println("Pixel at (" + x + "," + y + ") with " + color + "
color"); }}
```

Вміст файлу PixelFactory.java:

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

/**
 * Фабрика пікселів.
 * Реалізує керування об'єктами Pixel для Flyweight-патерну.
 * Створює новий піксель лише якщо ще не існує пікселя з таким кольором.
 */
public class PixelFactory {
    /** Збереження вже створених пікселів */
    private static Map<String, Pixel> pixels = new HashMap<>();

    /**
     * Повертає піксель з заданим кольором.
     * Створює новий, якщо такого ще не існує.
     * @param color колір пікселя
     * @return об'єкт Pixel
     */
    public static Pixel getPixel(String color) {
        Pixel pixel = pixels.get(color);
        if (pixel == null) {
            pixel = new Pixel(color);
            pixels.put(color, pixel);
            System.out.println("Created new pixel with " + color + " color");
        }
        return pixel;
    }
}
```

Вміст файлу RectangleImage.java:

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

/**
 * Клас для формування зображення з пікселів.
 * Зберігає координати та посилання на пікселі (Flyweight).
 */
public class RectangleImage {
    private List<Coords> coordsList = new ArrayList<>();
```

```
    private List<Pixel> pixels = new ArrayList<Pixel>();

    /**
     * Додає піксель із заданими координатами та кольором.
     * @param x координата по горизонталі
     * @param y координата по вертикалі
     * @param color колір пікселя
     */
    public void addPixel(int x, int y, String color) {
        Pixel NewPixel = PixelFactory.getPixel(color);
        pixels.add(NewPixel);
        coordsList.add(new Coords(x, y));
    }

    /**
     * Малює усі пікселі.
     * Викликає draw() для кожного Pixel із відповідними координатами.
     */
    public void draw() {
        for (int i = 0; i < pixels.size(); i++) {
            pixels.get(i).draw(coordsList.get(i).x, coordsList.get(i).y);
        }
    }
}
```

Вміст файлу Main.java:

```
import Flyweight.*;
/**
 * Демонстраційний клас для показу роботи Flyweight-патерну.
 * Створює RectangleImage і додає пікселі з різними кольорами.
 * Використовує PixelFactory для економії пам'яті.
 */
public class Main {
    public static void main(String[] args) {
        RectangleImage img =  new RectangleImage();
        img.addPixel(0, 0, "Black");
        img.addPixel(1, 0, "Yellow");
        img.addPixel(1, 0, "Black");
        img.addPixel(1, 1, "Yellow");
        img.draw();
    }
}
```

Результат виконання програми:

```
Created new pixel with Black color
Created new pixel with Yellow color
Pixel at (0,0) with Black color
Pixel at (1,0) with Yellow color
Pixel at (1,0) with Black color
Pixel at (1,1) with Yellow color
```

Рисунок 4 – результат виконання програми згідно другого завдання

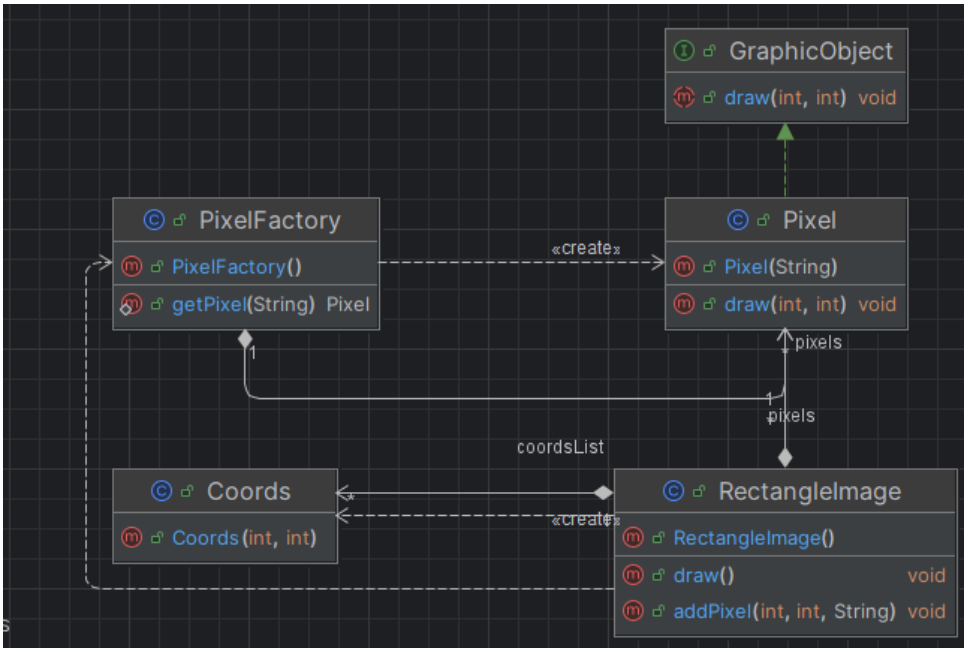

Рисунок 5 – Побудована в IntelliJ IDEA UML діаграма для другого завдання

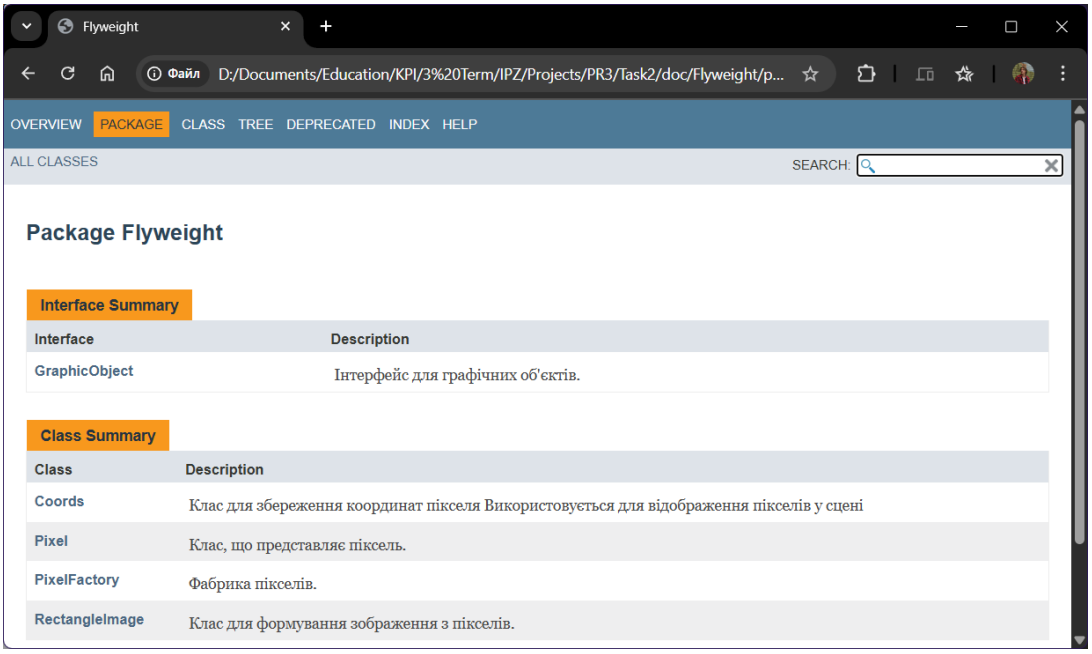

Рисунок 6 – Створена документація у форматі веб сторінки

Висновки:

Отже, під час виконання практичної роботи було створено дві програми, що допомогли закріпити знання з патернів програмування, зокрема Composite та Flyweight. Використання цих патернів дійсно полегшує роботу, особливо коли потрібно додавати нові функції без редагування початкового коду. Під час виконання практичної роботи проблем не виникло. Увесь код міститься на даному GitHub репозиторії: https://github.com/BronievM/KPI-IPZ-2025/tree/main/PR3