Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №6

з дисципліни «Технології розробки програмного забезпечення»

Патерни програмування

Виконав:
студент групи ІА-34
Швець Роман Вадимович

Перевірив:
асистент кафедри ІСТ
Мягкий Михайло Юрійович

Київ - 2025

Тема: Патерни програмування

Мета: Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

4. **Графічний редактор** (proxy, prototype, decorator, bridge, flyweight, SOA)
Графічний редактор повинен вміти створювати / редагувати растрові (або векторні на розсуд студента) зображення в 2-3 основних популярних форматах (bmp, png, jpg), мати панель інструментів для створення графічних примітивів, вибору кольорів, нанесення тексту, додавання найпростіших візуальних ефектів (ч/б растр, інфрачервоний растр, 2-3 на вибір учня), роботи з шарами.

Теоретичні відомості

Шаблони проєктування (патерни) — це загальні рішення типових проблем, що виникають у процесі розроблення програмного забезпечення. Вони не є готовим кодом, а радше описують структуру взаємодії між класами та об’єктами, допомагаючи створювати гнучкі, масштабовані й легко підтримувані системи. Використання шаблонів дозволяє стандартизувати архітектурні рішення, полегшити командну розробку та уникнути повторення вже відомих помилок у проєктуванні.

Шаблон Abstract Factory (Абстрактна фабрика) використовується для створення сімейств пов’язаних об’єктів без необхідності вказувати їх конкретні класи. Замість створення об’єктів напряму, програма звертається до фабрики, яка повертає необхідні екземпляри. Це дозволяє легко змінювати ціле “сімейство” об’єктів, не змінюючи код клієнта. Такий підхід широко застосовується, коли потрібно підтримувати різні варіації продуктів — наприклад, різні стилі інтерфейсу користувача (темна або світла тема).

Шаблон Factory Method (Фабричний метод) також належить до порождаючих патернів і дозволяє делегувати створення об’єктів підкласам. Базовий клас визначає загальний інтерфейс для створення об’єкта, але рішення про те, який

конкретно тип створити, приймає підклас. Це робить систему розширюваною — при появі нових типів об’єктів не потрібно змінювати наявний код, достатньо створити новий підклас із власною реалізацією фабричного методу.

Шаблон Memento (Знімок) призначений для збереження і відновлення попереднього стану об’єкта без порушення принципу інкапсуляції. У ньому беруть участь три компоненти: Originator (об’єкт, стан якого зберігається), Memento (знімок стану) і Caretaker (керує збереженням та відновленням знімків). Цей шаблон часто використовується для реалізації функцій “Undo” або “Відміна дій” у програмах, наприклад, у текстових або графічних редакторах.

Шаблон Observer (Спостерігач) описує механізм “один до багатьох”, коли при зміні стану одного об’єкта всі залежні від нього об’єкти автоматично сповіщаються та оновлюються. Це дозволяє відокремити логіку сповіщення від логіки обробки подій. Класичним прикладом є підписка в інтерфейсі користувача — коли модель даних змінюється, усі пов’язані з нею елементи інтерфейсу (спостерігачі) автоматично оновлюють свій стан.

Шаблон Decorator (Декоратор) належить до структурних патернів і призначений для динамічного додавання нової поведінки об’єктам без зміни їхнього коду. Замість створення підкласів, нова функціональність “обгортає” базовий об’єкт, зберігаючи його інтерфейс. Це дозволяє комбінувати різні модифікації в гнучкий спосіб. У контексті графічного редактора декоратор може застосовувати різні ефекти до зображення — наприклад, фільтри, тіні чи кольорові корекції — послідовно або комбіновано.

Застосування цих шаблонів у проєктуванні програмних систем забезпечує розширюваність, спрощує тестування, дозволяє дотримуватись принципів SOLID та робить архітектуру більш зрозумілою і стійкою до змін.

Хід роботи

Для реалізації розширюваної системи застосування ефектів у графічному редакторі використано шаблон проєктування «Декоратор» (Decorator). Його основна ідея полягає в тому, що нову поведінку можна додавати до об’єкта динамічно, не змінюючи вихідний код класу. У проєкті це реалізовано через інтерфейс IImage, який визначає спільну поведінку для всіх зображень, базовий клас RealImage, що

представляє реальне зображення, та абстрактний клас ImageDecorator, який дозволяє «обгортати» зображення додатковими ефектами. Кожен конкретний декоратор (GrayscaleDecorator, SepiaDecorator, InvertDecorator) реалізує власну логіку обробки, забезпечуючи можливість комбінування кількох ефектів послідовно без зміни структури програми.

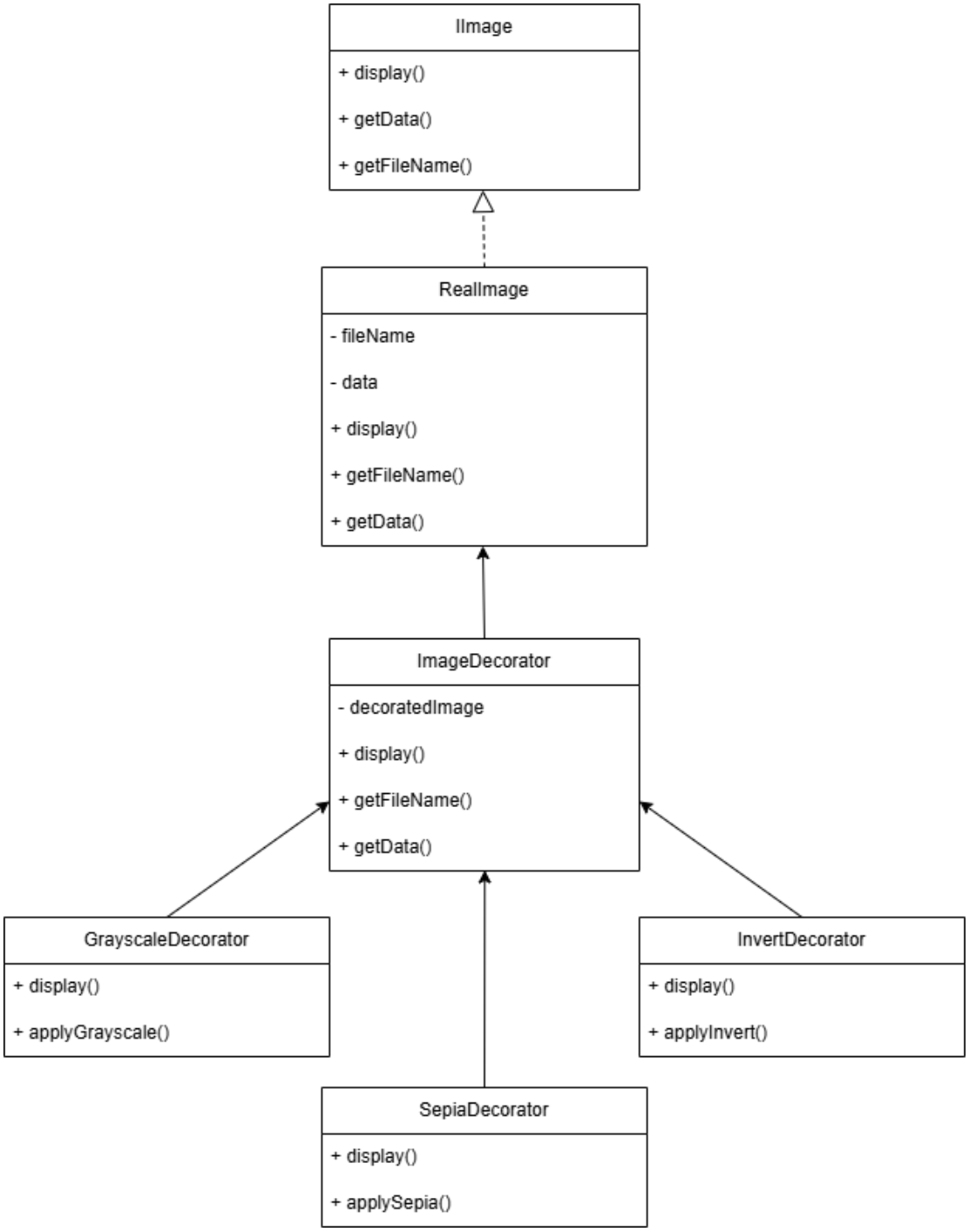


Рисунок 6.1 – Діаграма класів для Decorator

На поданій діаграмі класів відображено застосування шаблону «Декоратор» у структурі системи графічного редактора. Усі класи реалізують єдиний інтерфейс

IImage, що забезпечує взаємозамінність компонентів. Клас ImageDecorator виступає проміжною ланкою між базовим об’єктом (RealImage) і конкретними реалізаціями ефектів, дозволяючи динамічно додавати нові властивості. Такий підхід робить архітектуру програми гнучкою та модульною — нові ефекти можна додавати без зміни вже існуючих класів, а комбінація декораторів дозволяє створювати складні візуальні обробки в межах єдиної системи.

Вихідні коди класів системи

Лістинг 1 – Інтерфейс IImage

```
package com.mycompany.code;

public interface IImage {
    void display();
    byte[] getData();
    String getFileName();
}
```

Лістинг 2 – клас RealImage

```
public class RealImage implements IImage {

    private String fileName;
    private byte[] data;

    public RealImage(String fileName, byte[] data) {
        this.fileName = fileName;
        this.data = data;
        System.out.println("Завантажено зображення: " + fileName);
    }

    @Override
    public void display() {
        System.out.println("Відображення оригінального зображення: " + fileName);
    }

    @Override
    public byte[] getData() {
        return data;
    }

    @Override
    public String getFileName() {
        return fileName;
    }
}
```

Лістинг 3 – клас ImageDecorator

```
public abstract class ImageDecorator implements IImage {
    protected IImage decoratedImage;

    public ImageDecorator(IImage decoratedImage) {
        this.decoratedImage = decoratedImage;
    }
```

```
    @Override
    public void display() {
        decoratedImage.display();
    }

    @Override
    public byte[] getData() {
        return decoratedImage.getData();
    }

    @Override
    public String getFileName() {
        return decoratedImage.getFileName();
    }
}
```

Лістинг 4 – GrayscaleDecorator

```
public class GrayscaleDecorator extends ImageDecorator {

    public GrayscaleDecorator(IImage decoratedImage) {
        super(decoratedImage);
    }

    @Override
    public void display() {
        decoratedImage.display();
        applyGrayscale();
    }

    private void applyGrayscale() {
        System.out.println("Застосовано ефект: чорно-білий (Grayscale)");
        // Тут можна додати логіку зміни байтів зображення
    }
}
```

Лістинг 5 – SepiaDecorator

```
public class SepiaDecorator extends ImageDecorator {

    public SepiaDecorator(IImage decoratedImage) {
        super(decoratedImage);
    }

    @Override
    public void display() {
        decoratedImage.display();
        applySepia();
    }

    private void applySepia() {
        System.out.println("Застосовано ефект: сепія (Sepia)");
    }
}
```

Лістинг 6 – InvertDecorator

```
public class InvertDecorator extends ImageDecorator {

    public InvertDecorator(IImage decoratedImage) {
        super(decoratedImage);
    }
```

```
    @Override
    public void display() {
        decoratedImage.display();
        applyInvert();
    }

    private void applyInvert() {
        System.out.println("Застосовано ефект: негатив (Invert)");
    }
}
```

Лістинг 7 – Демонстрація використання

```
public class EditorUI {

    public static void main(String[] args) {
        IImage image = new RealImage("photo.svg", new byte[0]);

        System.out.println("\n--- Без ефектів ---");
        image.display();

        System.out.println("\n--- З ефектом Grayscale ---");
        IImage grayscale = new GrayscaleDecorator(image);
        grayscale.display();

        System.out.println("\n--- З ефектом Grayscale + Sepia ---");
        IImage combo = new SepiaDecorator(new GrayscaleDecorator(image));
        combo.display();
    }
}
```

У поданих лістингах реалізовано застосування шаблону проєктування «Декоратор» (Decorator) у системі графічного редактора. Базовий інтерфейс IImage визначає спільні методи для всіх об’єктів-зображень. Клас RealImage є основною реалізацією інтерфейсу та відповідає за зберігання й відображення даних зображення. Абстрактний клас ImageDecorator виступає проміжною ланкою, що містить посилання на об’єкт типу IImage і делегує йому виклики методів, створюючи можливість розширення поведінки. Конкретні декоратори (GrayscaleDecorator, SepiaDecorator, InvertDecorator) перевизначають метод display() і додають власну логіку обробки, наприклад, застосування ефектів до зображення. Така реалізація дозволяє накладати кілька ефектів послідовно, комбінувати їх та легко додавати нові, не змінюючи вихідні класи. Завдяки цьому код залишається модульним, гнучким і зручним для подальшого розширення.

Висновок:

У ході виконання лабораторної роботи було реалізовано шаблон проєктування Decorator (Декоратор) у контексті системи графічного редактора.

Цей шаблон дозволив динамічно розширювати функціональність об’єктів без зміни їхньої базової структури, що особливо важливо для реалізації гнучких систем обробки зображень. У проєкті створено ієрархію класів, де інтерфейс IImage визначає спільну поведінку, клас RealImage виступає базовим компонентом, а ImageDecorator та його нащадки (GrayscaleDecorator, SepiaDecorator, InvertDecorator) забезпечують можливість послідовного накладання ефектів. Такий підхід спростив підтримку та розширення системи, дозволив реалізувати комбінацію різних фільтрів без дублювання коду.

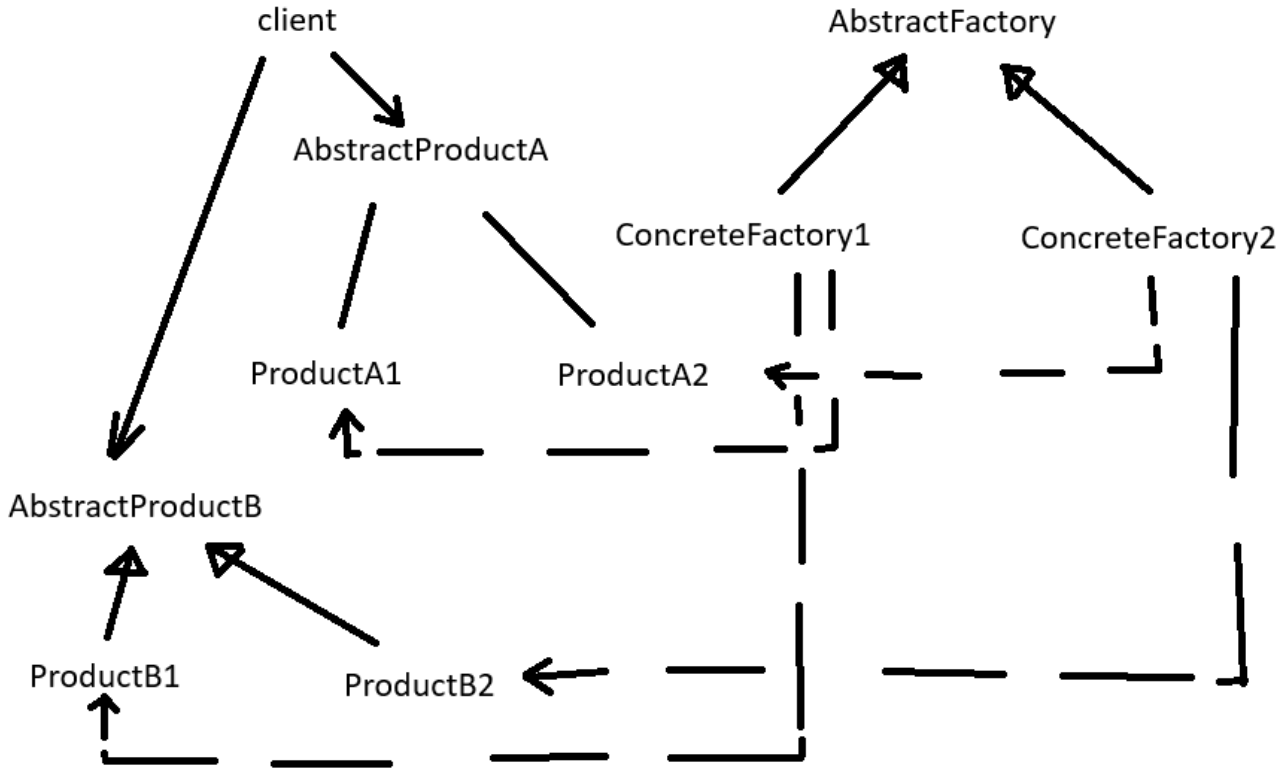
Використання патерну Decorator зробило архітектуру програми більш гнучкою, масштабованою та придатною для подальшого розвитку, зберігаючи при цьому принципи інкапсуляції й повторного використання коду.

Відповіді на контрольні запитання:

1. Яке призначення шаблону «Абстрактна фабрика»?

Забезпечує створення групи взаємопов’язаних об’єктів без вказівки їх конкретних класів, дозволяючи легко змінювати варіації продуктів у системі.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія?

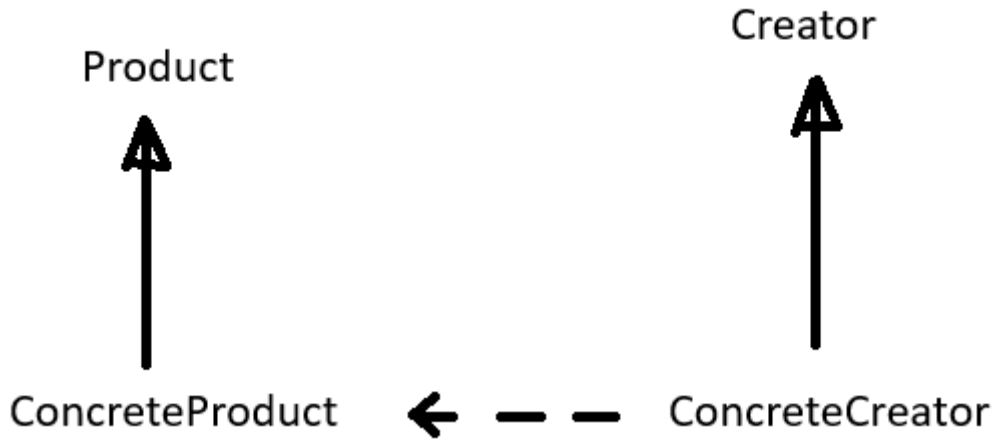
Основні класи: AbstractFactory, ConcreteFactory, AbstractProduct, ConcreteProduct, Client.

Клієнт звертається до фабрики через абстрактний інтерфейс, а фабрика створює відповідні конкретні продукти.

4. Яке призначення шаблону «Фабричний метод»?

Дозволяє визначити інтерфейс для створення об’єктів, делегуючи вибір конкретного класу підкласам.

5. Нарисуйте структуру шаблону «Фабричний метод».



6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія?

Класи: Product, ConcreteProduct, Creator, ConcreteCreator.

Клієнт використовує метод створення з базового класу, не знаючи, який саме продукт створюється.

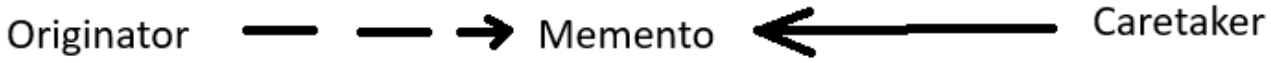
7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

«Фабричний метод» створює один тип продукту, а «Абстрактна фабрика» — цілу групу взаємопов’язаних продуктів.

8. Яке призначення шаблону «Знімок»?

Зберігає стан об’єкта для можливості його відновлення без порушення інкапсуляції.

9. Нарисуйте структуру шаблону «Знімок».



10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

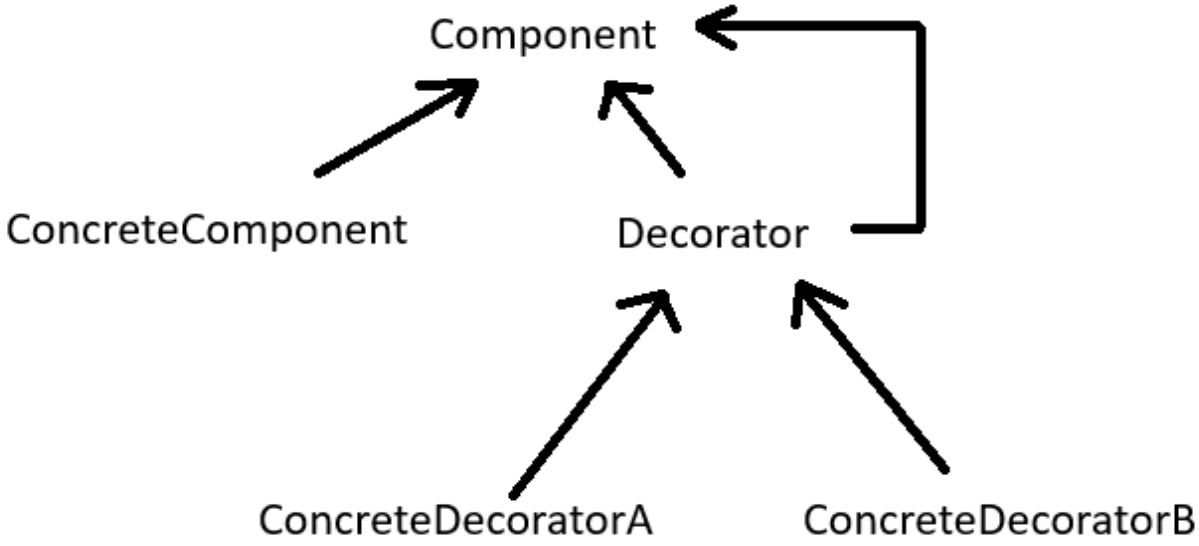
Класи: Originator (створює і відновлює стан), Memento (зберігає стан), Caretaker (керує знімками).

Originator створює об’єкт Memento, який зберігає його стан, а Caretaker вирішує, коли його відновити.

11. Яке призначення шаблону «Декоратор»?

Дозволяє динамічно додавати об’єктам нову функціональність без зміни їхнього коду.

12. Нарисуйте структуру шаблону «Декоратор».



13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?

Класи: Component, ConcreteComponent, Decorator, ConcreteDecorator.

Decorator містить посилання на Component і делегує йому виклики, додаючи додаткову поведінку.

14. Які є обмеження використання шаблону «Декоратор»?

Збільшує кількість дрібних класів у проєкті й ускладнює налагодження, особливо при вкладених декораторах.