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

Лабораторна робота №7

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 «Mediator», «Facade», «Bridge», «Template method» 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 Mediator (Посередник) належить до поведінкових патернів і призначений для зменшення зв’язності між множиною класів шляхом переміщення логіки їхньої взаємодії в один клас-посередник. Замість того щоб об’єкти спілкувалися один з одним безпосередньо, створюючи заплутану мережу залежностей, вони взаємодіють лише з посередником, який перенаправляє запити відповідним компонентам. Це дозволяє змінювати взаємодію між об’єктами незалежно від самих об’єктів та спрощує повторне використання коду, наприклад, у складних формах інтерфейсу користувача, де елементи управління залежать один від одного.

Шаблон Facade (Фасад) є структурним патерном, який надає простий уніфікований інтерфейс до складної системи класів, бібліотеки або фреймворку.

Він приховує складність внутрішньої архітектури підсистеми, надаючи клієнту лише ті методи, які йому необхідні для роботи. Використання фасаду дозволяє відокремити бізнес-логіку програми від деталей реалізації сторонніх бібліотек, що робить код більш читабельним та полегшує процес оновлення або заміни компонентів системи в майбутньому.

Шаблон Bridge (Міст) — це структурний патерн, який дозволяє розділити абстракцію та її реалізацію так, щоб вони могли змінюватися незалежно одна від одної. Цей підхід замінює статичне спадкування на композицію: абстракція (високорівнева логіка) делегує роботу об’єкту реалізації (низькорівневі деталі), з яким вона пов’язана. Таке рішення є особливо корисним для уникнення «комбінаторного вибуху» кількості класів, коли систему необхідно розширювати у кількох незалежних вимірах, наприклад, підтримувати різні типи графічних фігур на різних операційних системах або у різних форматах рендерингу.

Шаблон Template Method (Шаблонний метод) визначає скелет алгоритму в базовому класі, делегуючи реалізацію окремих його кроків підкласам. Базовий клас задає структуру та порядок виконання операцій, тоді як підкласи можуть перевизначати або доповнювати конкретні кроки, не змінюючи загальної структури алгоритму. Це забезпечує повторне використання коду, оскільки спільна частина алгоритму залишається в одному місці, а варіативна поведінка виноситься в спадкоємців, що часто використовується при розробці фреймворків та бібліотек.

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 гнучкої системи відображення графічних примітивів у різних форматах (растровому та векторному) використано шаблон проєктування «Міст» (Bridge). Його основна ідея полягає у відокремленні абстракції від її реалізації, що дозволяє змінювати їх незалежно одна від одної. У проєкті це досягається шляхом розділення логіки на дві окремі ієрархії: геометричні фігури та механізми рендерингу. Абстрактний клас Shape визначає загальну структуру для фігур (Circle, Square) і містить посилання на інтерфейс IRenderer, який виступає в ролі моста.

Конкретні реалізації рендерера (RasterRenderer, VectorRenderer) виконують побудову зображення відповідним способом (через пікселі або векторні шляхи), що дозволяє малювати будь-яку фігуру в будь-якому форматі без дублювання коду класів.

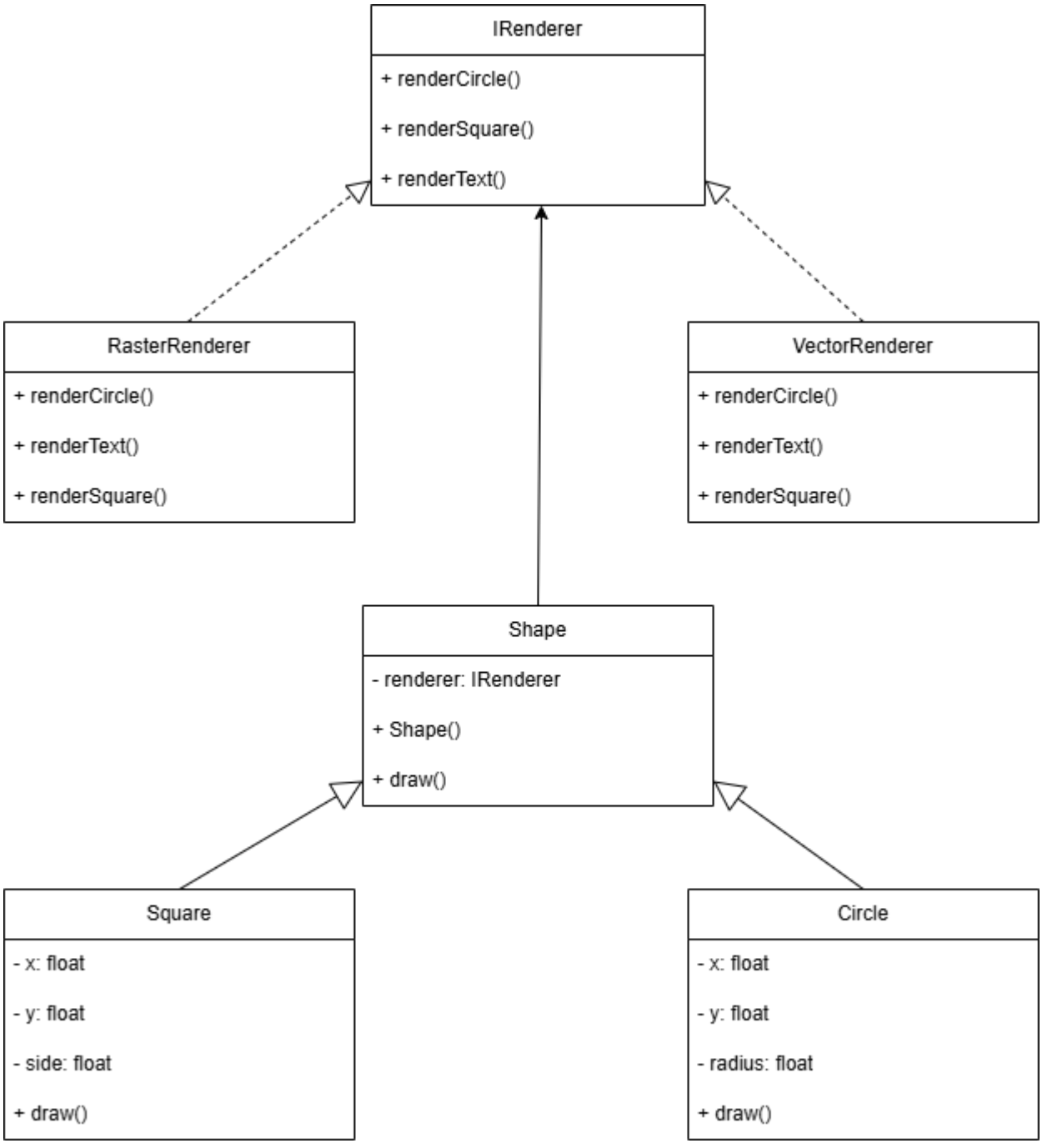


Рисунок 7.1 – Діаграма класів для Bridge

На поданій діаграмі класів відображено структуру шаблону «Міст», де візуально розділено ієрархію абстракції (клас Shape та його нащадки) та ієрархію реалізації (інтерфейс IRenderer та його імплементації). Зв’язок агрегації між Shape та IRenderer демонструє делегування: фігура знає, «що» малювати, але передоручає

завдання «як» малювати об’єкту рендерера. Такий підхід робить архітектуру програми стійкою до змін — додавання нових типів фігур не впливає на класи рендерингу, а впровадження підтримки нових графічних форматів не вимагає модифікації коду існуючих геометричних примітивів.

Вихідні коди класів системи

Лістинг 1 – Інтерфейс IRenderer

```
package com.mycompany.code;

public interface IRenderer {
    void renderCircle(float x, float y, float radius);
    void renderSquare(float x, float y, float side);
}
```

Лістинг 2 – клас RasterRenderer

```
package com.mycompany.code;

public class RasterRenderer implements IRenderer {

    @Override
    public void renderCircle(float x, float y, float radius) {
        System.out.println("RasterRenderer: Малюємо КОЛО пікселями [центр: " + x +
", " + y + "; радіус: " + radius + "]");
    }

    @Override
    public void renderSquare(float x, float y, float side) {
        System.out.println("RasterRenderer: Малюємо КВАДРАТ пікселями [початок: "
+ x + ", " + y + "; сторона: " + side + "]");
    }
}
```

Лістинг 3 – клас VectorRenderer

```
package com.mycompany.code;

public class VectorRenderer implements IRenderer {

    @Override
    public void renderCircle(float x, float y, float radius) {
        System.out.println("VectorRenderer: Малюємо КОЛО як векторний шлях (SVG)
[cx=" + x + " cy=" + y + " r=" + radius + "]");
    }

    @Override
    public void renderSquare(float x, float y, float side) {
        System.out.println("VectorRenderer: Малюємо КВАДРАТ векторами [rect x=" +
x + " y=" + y + " width=" + side + "]");
    }
}
```

Лістинг 4 – клас Shape

```
package com.mycompany.code;

public abstract class Shape {
    protected IRenderer renderer;

    public Shape(IRenderer renderer) {
        this.renderer = renderer;
    }

    public abstract void draw();
}
```

Лістинг 5 – клас Circle

```
package com.mycompany.code;

public class Circle extends Shape {
    private float x, y, radius;

    public Circle(float x, float y, float radius, IRenderer renderer) {
        super(renderer);
        this.x = x;
        this.y = y;
        this.radius = radius;
    }

    @Override
    public void draw() {
        renderer.renderCircle(x, y, radius);
    }
}
```

Лістинг 6 – клас Square

```
package com.mycompany.code;

public class Square extends Shape {
    private float x, y, side;

    public Square(float x, float y, float side, IRenderer renderer) {
        super(renderer);
        this.x = x;
        this.y = y;
        this.side = side;
    }

    @Override
    public void draw() {
        renderer.renderSquare(x, y, side);
    }
}
```

Лістинг 7 – Демонстрація використання

```
package
com.mycompany.code;

public
```

```
class EditorUI {

public static void main(String[] args) {
// Створюємо "драйвери" для рендерингу
IRenderer raster = new RasterRenderer();
IRenderer vector = new VectorRenderer();

System.out.println("\n--- Рендеринг у Растровому форматі ---");
Shape circleBmp = new Circle(10, 10, 5, raster);
Shape squareBmp = new Square(5, 5, 15, raster);

circleBmp.draw();
squareBmp.draw();

System.out.println("\n--- Рендеринг у Векторному форматі ---");
Shape circleSvg = new Circle(20, 20, 10, vector);
Shape squareSvg = new Square(100, 100, 50, vector);

circleSvg.draw();
squareSvg.draw();
}
}
```

У поданих лістингах реалізовано застосування шаблону проєктування «Міст» (Bridge) у системі графічного редактора. Інтерфейс IRenderer визначає спільні методи для всіх механізмів малювання, незалежно від формату. Класи RasterRenderer та VectorRenderer є конкретними реалізаціями цього інтерфейсу та відповідають за безпосереднє відображення графіки у растровому (піксельному) або векторному виглядах. Абстрактний клас Shape виступає високорівневою абстракцією, що містить посилання на об’єкт типу IRenderer і делегує йому низькорівневі виклики малювання, відокремлюючи логіку фігури від способу її рендерингу. Конкретні класи фігур (Circle, Square) розширюють абстракцію Shape, визначають свої специфічні властивості (координати, радіус, сторони) і використовують методи рендерера у власному методі draw(). Така реалізація дозволяє додавати нові фігури або нові способи рендерингу незалежно один від одного, не змінюючи існуючий код. Завдяки цьому код залишається модульним, гнучким і зручним для подальшого розширення.

Висновок:

У ході виконання лабораторної роботи було реалізовано шаблон проєктування Bridge (Міст) у контексті системи графічного редактора. Цей шаблон дозволив відокремити абстракцію від її реалізації, що особливо важливо для створення систем, які повинні працювати з різними графічними форматами (растровим та векторним). У проєкті створено структуру класів, де інтерфейс

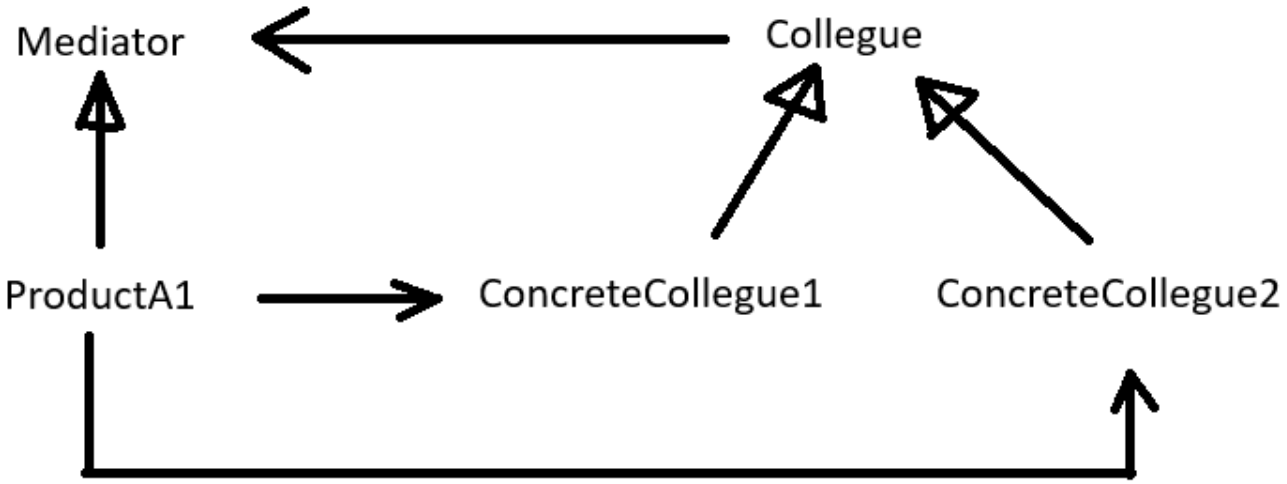
IRenderer та його реалізації (RasterRenderer, VectorRenderer) відповідають за технічні деталі відображення, а клас Shape та його нащадки (Circle, Square) керують логікою геометричних примітивів. Такий підхід спростив підтримку та розширення системи, дозволив уникнути надмірного зростання кількості класів при додаванні нових фігур або форматів.

Відповіді на контрольні запитання:

1. Яке призначення шаблону «Посередник»?

Забезпечує зменшення зв'язаності множини класів між собою шляхом переміщення їхньої взаємодії в один клас-посередник.

2. Нарисуйте структуру шаблону «Посередник».



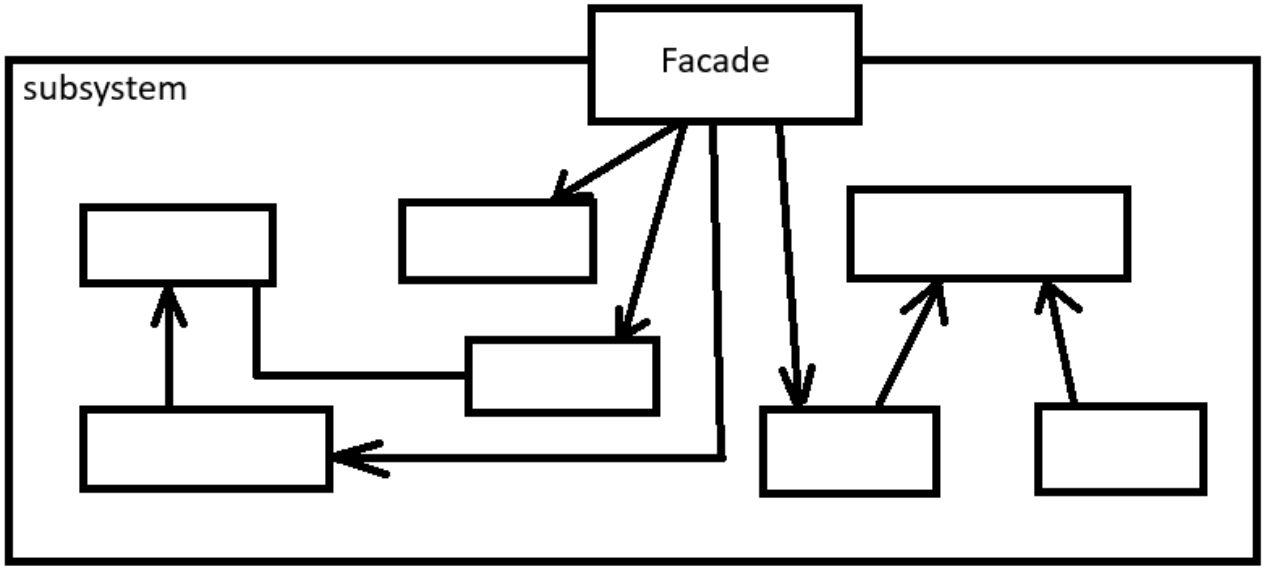
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

Класи: Mediator (інтерфейс), ConcreteMediator, Colleague (інтерфейс/базовий клас), ConcreteColleague. Колеги не спілкуються напряму; вони надсилають повідомлення Посереднику, який перенаправляє їх іншим Колегам, координуючи роботу.

4. Яке призначення шаблону «Фасад»?

Надає простий уніфікований інтерфейс для доступу до складної системи класів, бібліотеки або фреймворку, приховуючи складність їхньої внутрішньої роботи.

5. Нарисуйте структуру шаблону «Фасад».



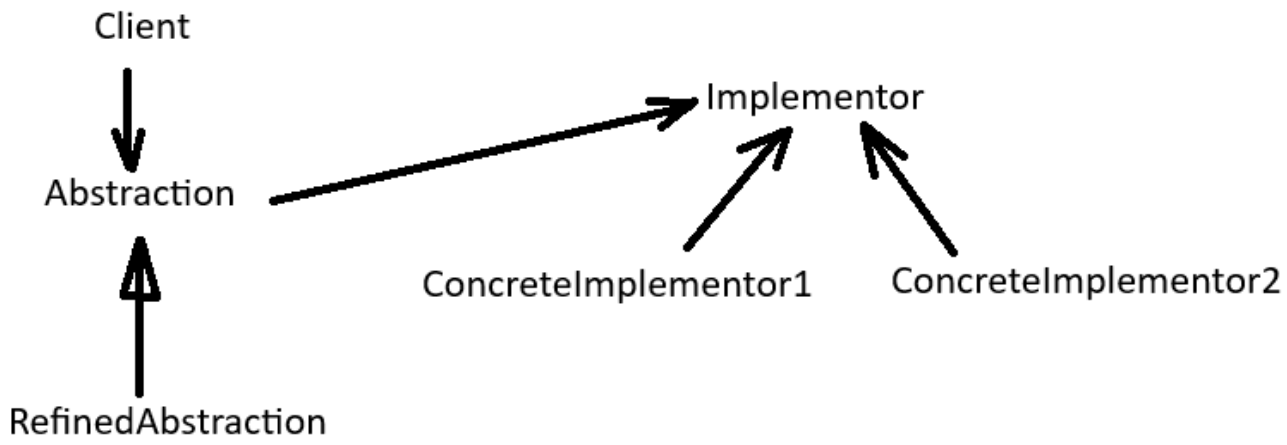
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

Класи: Facade (фасад), Subsystem classes (класи підсистеми), Client (клієнт). Клієнт звертається до методів Фасаду, а Фасад делегує виконання відповідним об'єктам підсистеми. Клієнт не взаємодіє з класами підсистеми напряму.

7. Яке призначення шаблону «Міст»?

Розділяє абстракцію та її реалізацію на дві окремі ієрархії, дозволяючи змінювати їх незалежно одна від одної (використовує композицію замість успадкування).

8. Нарисуйте структуру шаблону «Міст».



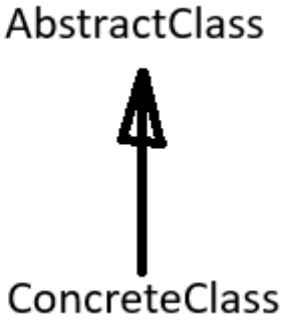
9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

Класи: Abstraction, RefinedAbstraction, Implementor, ConcreteImplementor. Abstraction містить посилання на об'єкт типу Implementor і делегує йому виконання низькорівневих операцій.

10. Яке призначення шаблону «Шаблонний метод»?

Визначає скелет алгоритму в базовому класі, делегуючи реалізацію окремих кроків цього алгоритму підкласам, не змінюючи загальної структури.

11. Нарисуйте структуру шаблону «Шаблонний метод».



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

Класи: AbstractClass (оголошує кроки алгоритму), ConcreteClass (реалізує кроки). Базовий клас викликає методи (кроки) у визначеному порядку; конкретні класи реалізують ці методи.

13. Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

«Шаблонний метод» — це поведінковий патерн, що керує потоком виконання алгоритму. «Фабричний метод» — це породжуючий патерн, який відповідає лише за створення об'єктів (часто Фабричний метод є одним із кроків усередині Шаблонного методу).

14. Яку функціональність додає шаблон «Міст»?

Він додає можливість незалежного розвитку (розширення) абстракції та

реалізації, а також дозволяє динамічно змінювати реалізацію під час виконання програми.