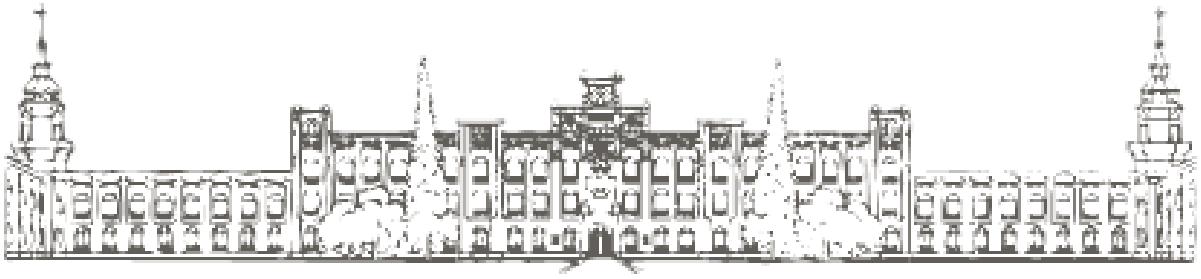


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Звіт лабораторної роботи №7

з курсу

«Технології розроблення програмного забезпечення»

Виконавець:
Нікітченко Наталя Олегівна
студентка групи ІА-33
залікова книжка № ІА-3318

«29» 11 2025 р.

Перевірив: **Мягкий М. Ю.**

Київ – 2025

7. ЛАБОРАТОРНА РОБОТА № 7

Тема: Патерни проектування.

Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

Завдання:

Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати один з розглянутих шаблонів за обраною темою.

• Реалізувати не менше 3-х класів відповідно до обраної теми.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

Теоретичні відомості:

Шаблон «Bridge»

Призначення патерну: Шаблон «Bridge» (міст) використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами.

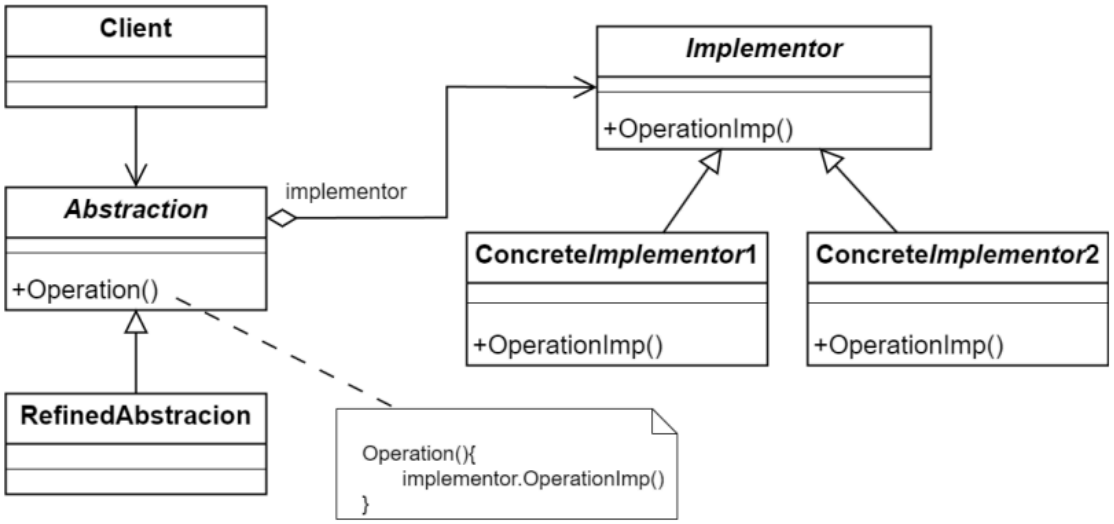


Рисунок 1. Структура патерну «Міст»

Проблема: Ви реалізовуєте графічний векторний редактор, який дозволяє рисувати круги, прямокутники, прямі та довільні лінії. Ви маєте реалізувати функціонал відображення отриманого рисунку на екрані та друкувати на принтер. Спростимо ситуацію: ваш редактор дозволяє рисувати лише лінії та круги. При такому підході, нам потрібно будувати ієрархію фігур з різною реалізацією дочірніх класів: LinePrint, LineDraw, CirclePrint, CircleDraw. якщо добавити прямі, то добавиться ще два підкласи, і т.д. А як бути, коли нам потрібно буде ще і реалізувати збереження в bitmap форматі? добавляємо ще LineBinery, CircleBinery? При такому підході ми отримуємо дуже складну ієрархію класів.

Рішення: В даному випадку ми можемо використати патерн «Міст» (Bridge): робимо дві ієрархії – фігур (Shape) та рисування (DrawApi).

При такому підході DrawApi – це інтерфейс імплементації відображення (графічного драйвера), а Shape – інтерфейс абстракції фігур, яка має агрегацію з об'єктом DrawApi. При такому підході фігури будуть делегувати рисування об'єкту DrawApi.

Лінія, коло, та інші будуть дочірніми класами до Shape, а WindowDrawApi та PrinterDrawApi – дочірні класи до DrawApi, які представляють графічні драйвери для відображення на екрані та принтері відповідно. Якщо нам потрібно буде добавити ще і збереження в bitmap форматі, то ми добавимо ще один підклас реалізації графічного драйвера BitmapDrawApi. Таким чином ми маємо дві різні ієрархії об’єктів і вони в нас не перетинаються і не збільшуються в геометричній прогресії при додаванні нових драйверів або фігур.

Також слід відмітити, що DrawApi нічого не знає про фігури (абстракцію), а дочірні класи абстракції не залежать від реалізації графічного драйвера.

Переваги та недоліки:

+ Дозволяє змінювати ієрархії абстракції та реалізації незалежно одна від одної.

+ Розділивши абстракцію від реалізації отримуємо більшу гнучкість та простіший супровід такого коду.

- Підвищена гнучкість при використанні патерну отримується за рахунок більшої складності, введення додаткових проміжних рівнів.

Хід роботи:

Тема: “Графічний редактор”

Реалізувати один з розглянутих шаблонів за обраною темою, а саме патерн «Bridge».

Розглянемо структуру цього прототипу(Рис.2):

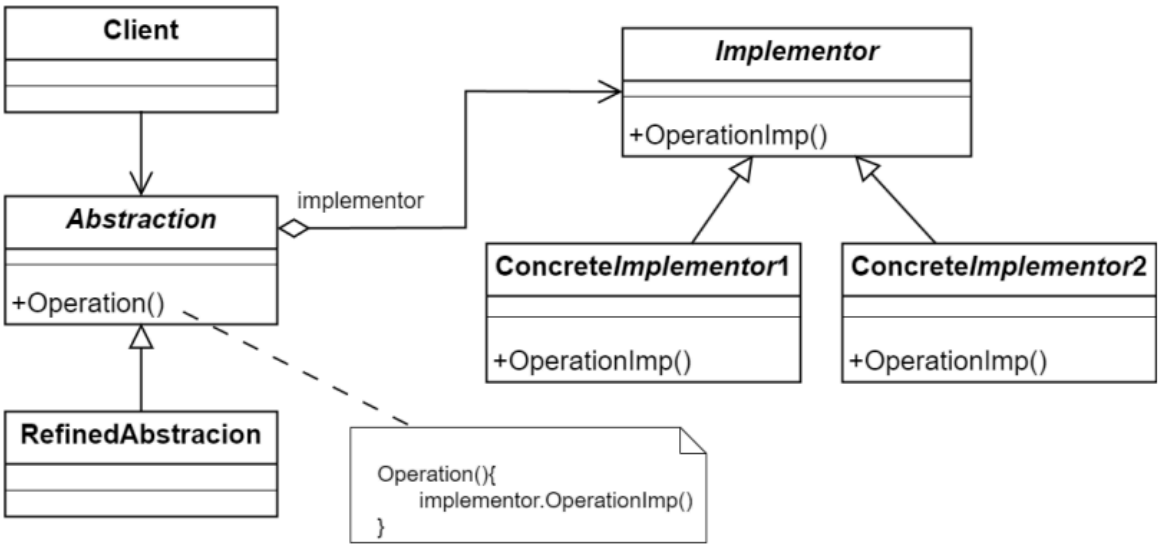


Рис.2 - Структура патерну «Міст»

Реалізація цього протипу відповідно даної теми(Рис.3):

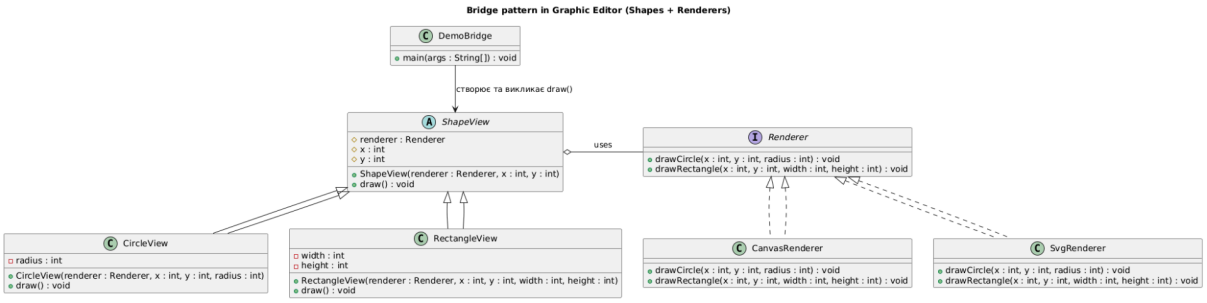


Рис.3 - Діаграма показує реалізацію патерну Bridge у графічному редакторі.

Фігури (ShapeView + її нащадки CircleView / RectangleView) містять абстракцію Renderer і делегують їй реальне малювання.

Renderer має конкретні реалізації CanvasRenderer і SvgRenderer, що дозволяє малювати тими самими фігурами на різних носіях без зміни коду фігур.

Renderer (Інтерфейс):

Це **міст** (Bridge).

Він описує *універсальні операції малювання*:

- drawCircle(x, y, radius)
- drawRectangle(x, y, width, height)

CanvasRenderer та SvgRenderer (Concrete Implementations):

CanvasRenderer малює реальні примітиви на JavaFX

CanvasSvgRenderer генерує SVG-код (рядки XML)

Вони реалізують інтерфейс Renderer, тому ShapeView може працювати з будь-яким.

ShaeView (Абстракція фігури)

- містить координати (x, y)
- має метод draw() — але сам не знає, як саме малювати!

CircleView / RectangleView:

Це конкретні фігури, що успадковують ShapeView.

CircleView:

- додає radius
- у draw(): renderer.drawCircle(...)

RectangleView:

- має width / height
- у draw(): renderer.drawRectangle(...)

Фігура → делегує → Renderer.

DemoBridge

Це просто демонстраційний main().

Фрагменти коду по реалізації цього шаблону:

```
package com.example.graphiceditor.model.bridge;

public class CanvasRenderer implements Renderer {  1 usage  new *

    @Override  1 usage  new *
    public void drawCircle(int x, int y, int radius) {
        System.out.println("[Canvas] Circle at (" + x + "," + y + "), r=" + radius);
    }

    @Override  1 usage  new *
    public void drawRectangle(int x, int y, int width, int height) {
        System.out.println("[Canvas] Rect at (" + x + "," + y + "), w=" + width + ", h=" + height);
    }
}
```

Рис.4 - CanvasRenderer

Renderer – інтерфейс реалізації з патерну Bridge. Він описує *як* малювати примітиви.
CanvasRenderer – конкретна реалізація цього інтерфейсу, яка “малює на Canvas”.

Методи:

- drawCircle(x, y, radius) – “намалювати коло” з центром (x,y) і радіусом radius.

- drawRectangle(x, y, width, height) – “намалювати прямокутник” з лівим верхнім кутом (x,y) і розмірами width,height.

```java
package com.example.graphiceditor.model.bridge;

public class CircleView extends ShapeView {  2 usages  new *

    private int radius;  2 usages

    public CircleView(Renderer renderer, int x, int y, int radius) {  2 usages  new *
        super(renderer, x, y);
        this.radius = radius;
    }

    @Override  4 usages  new *
    public void draw() {
        renderer.drawCircle(x, y, radius);
    }
}
```

Рис.5 - CircleView

CircleView:

- успадковує ShapeView - це конкретна фігура «коло».
- має своє поле radius те, чого немає в базовому класі.
- в конструкторі:
    - викликає super(renderer, x, y) → передає реалізацію малювання та координати базовому класу;
    - зберігає radius.
- в draw():
    - не малює сам, а делегує в renderer.

```java
package com.example.graphiceditor.model.bridge;

public class RectangleView extends ShapeView {  2 usages  new *

    private int width;  2 usages
    private int height;  2 usages

    public RectangleView(Renderer renderer, int x, int y, int width, int height) {  2 us
        super(renderer, x, y);
        this.width = width;
        this.height = height;
    }

    @Override  4 usages  new *
    public void draw() {
        renderer.drawRectangle(x, y, width, height);
    }
}
```

Рис.6 -RectangleView

Це конкретна фігура «прямокутник».

Знає свої розміри (width, height).

В draw() делегує роботу реалізації:

```java
1   package com.example.graphiceditor.model.bridge;
2
3   public interface Renderer {  8 usages  2 implementations  new *
4
5       void drawCircle(int x, int y, int radius);  1 usage  2 implementations  new *
6
7       void drawRectangle(int x, int y, int width, int height);  1 usage  2 implementations  new *
8   }
```

Рис.7 - Renderer

Це інтерфейс реалізації в патерні Bridge. Він описує що треба робити в даному випадку малювати коло і прямокутник. Він не описує як це робити , а показує тільки сигнатури методів.

```java
package com.example.graphiceditor.model.bridge;

public abstract class ShapeView {  6 usages  2 inheritors  new *

    protected final Renderer renderer;  3 usages

    protected int x;
    protected int y;

    protected ShapeView(Renderer renderer, int x, int y) {  2 usages  new *
        this.renderer = renderer;
        this.x = x;
        this.y = y;
    }

    public abstract void draw();  4 usages  2 implementations  new *
}
```

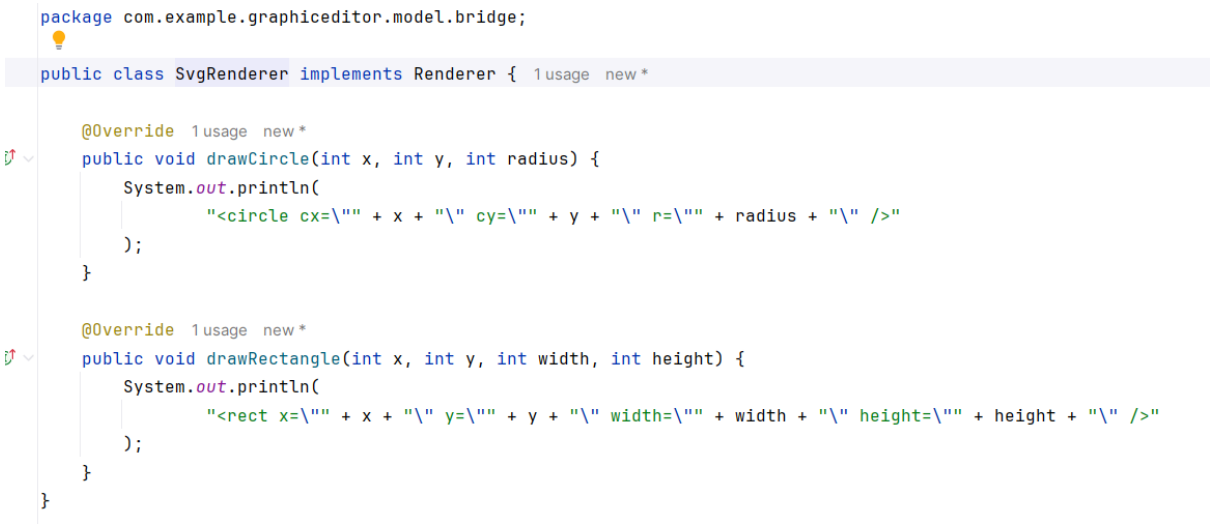
Рис.8 - ShapeView

Цей клас ShapeView зберігає посилання на Renderer:

protected final Renderer renderer;

- Координати x, y – спільні для всіх фігур.
- Конструктор приймає:
    - конкретний Renderer (Canvas, SVG і т.д.),
    - координати фігури.

Метод draw() абстрактний, кожна конкретна фігура сама вирішує, яки саме метод renderer викликати (circle/rectangle…).



```
package com.example.graphiceditor.model.bridge;

public class SvgRenderer implements Renderer {  1 usage  new *

    @Override  1 usage  new *
    public void drawCircle(int x, int y, int radius) {
        System.out.println(
                "<circle cx=\"" + x + "\" cy=\"" + y + "\" r=\"" + radius + "\" />"
        );
    }

    @Override  1 usage  new *
    public void drawRectangle(int x, int y, int width, int height) {
        System.out.println(
                "<rect x=\"" + x + "\" y=\"" + y + "\" width=\"" + width + "\" height=\"" + height + "\" />"
        );
    }
}
```

Рис.9 - SVGRenderer

Це конкретна реалізація інтерфейсу Renderer. Вона генерує SVG-теги і друкує їх у консоль.

Методи реалізують, як саме малювати:

- drawCircle - формує <circle ... />.
- drawRectangle - формує <rect ... />.

Висновки:

У побудованому графічному редакторі патерн Bridge («Міст») застосовано для чіткого розділення моделі графічних примітивів (абстракції) та механізму їхнього відображення (реалізації). Такий поділ дозволяє розвивати обидві частини незалежно, уникнути міцного зв'язування і значно спростити розширюваність проєкту. Реалізована структура відповідає UML-діаграмі патерну Bridge.

Базовий абстрактний клас ShapeView інкапсулює спільні для всіх графічних фігур параметри:

— координати x, y;

— посилання на реалізатор графічних операцій Renderer;

— абстрактний метод draw(), який визначає *що саме* має бути намальовано.

Завдяки такій структурі, ShapeView не містить жодного коду, пов’язаного з конкретною платформою малювання — лише логіку фігури як об’єкта.

Кожна конкретна фігура — CircleView, RectangleView — є RefinedAbstraction. Вони додають власні параметри (radius, width, height) та конкретизують операцію draw(), делегуючи роботу реалізатору.

Таким чином, ShapeView формує стабільну ієрархію абстракцій, яку можна розширювати новими фігурами без втручання в механізм рендерингу.

Реалізація (Renderer + підкласи) – незалежний механізм візуалізації

Інтерфейс Renderer визначає набір елементарних низькорівневих графічних операцій:

- drawCircle(x, y, radius)
- drawRectangle(x, y, width, height)

У застосунку наведено дві реалізації:

- CanvasRenderer — малює на JavaFX Canvas
- SvgRenderer — генерує SVG-теги (<circle cx="..." cy="..." r="..."/>, <rect x="..." y="..." width="..." height="..."/>)

Обидві реалізації є взаємозамінними. Той самий набір фігур можна виводити в Canvas або експортувати у SVG — змінивши лише об’єкт Renderer.

Загалом реалізація патерну Bridge у графічному редакторі є повною, та демонструє всі ключові властивості шаблону:

- чітке розділення відповідальностей (фігури, рендерери);
- можливість незалежного розширення обох частин;
- відсутність дублювання коду;
- гнучкість у зміні “способу візуалізації”;
- універсальність під час експорту/рендеру на різні платформи;

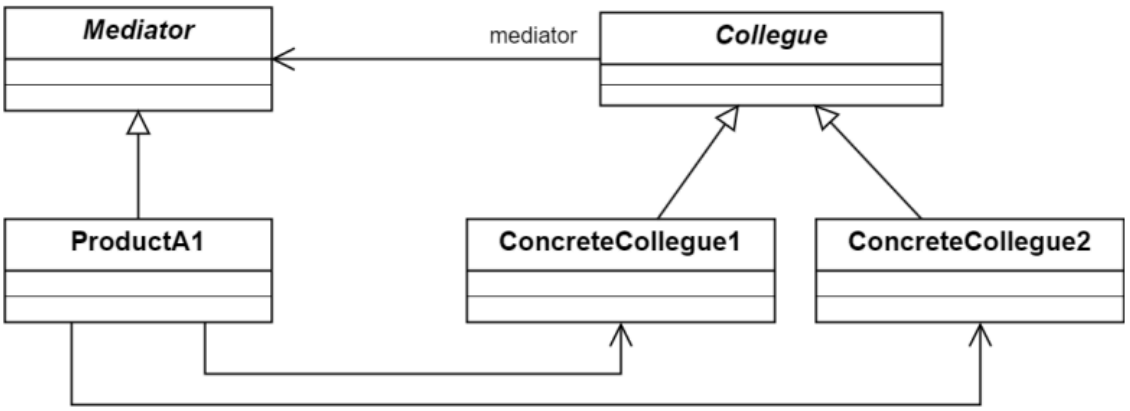
Контрольні питання:

1. Яке призначення шаблону «Посередник»?

Шаблон «Mediator» (посередник) використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного

Також цей шаблон зручно застосовувати у випадках, коли безліч об'єктів взаємодіє між собою деяким структурованим чином, однак складним для розуміння. У такому випадку вся логіка взаємодії виноситься в окремий об'єкт. Кожен із взаємодіючих об'єктів зберігає посилання на об'єкт «медіатор»

2. Нарисуйте структуру шаблону «Посередник».



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

Mediator (абстрактний клас / інтерфейс):

- центральний координатор
- знає про всі об’єкти-колеги
- приймає повідомлення та розподіляє їх іншим

ConcreteMediator:

- реалізує логіку посередника
- зберігає посилання на конкретних колег
- вирішує, як саме вони взаємодіють

Colleague (базовий клас колег):

- містить посилання на Mediator
- не напряму викликає інші об’єкти
- для взаємодії завжди використовує mediator.notify(...)

ConcreteColleague1, ConcreteColleague2:

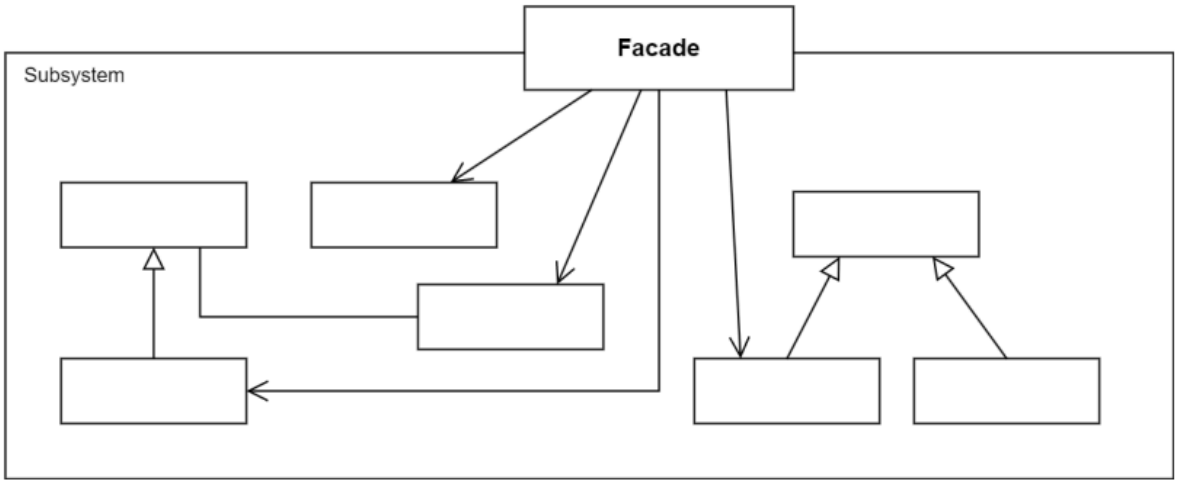
- реальні об’єкти, які мають спілкуватися
- коли їм потрібно взаємодіяти з кимось — вони повідомляють посередника

Взаємодія: ConcreteColleague - Mediator - інші ConcreteColleague.

4. Яке призначення шаблону «Фасад»?

Шаблон «Facade» (фасад) передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми. Оскільки підсистема може складатися з безлічі класів, а кількість її функцій – не більше десяти, то щоб уникнути створення «спагетікоду» (коли все тісно пов'язано між собою) виділяють один загальний інтерфейс доступу, здатний правильним чином звертатися до внутрішніх деталей.

5. Нарисуйте структуру шаблону «Фасад».



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

Facade:

- єдиний публічний інтерфейс
- спрощує роботу складної системи
- приховує взаємодії між компонентами підсистеми

Класи підсистеми (Subsystem):

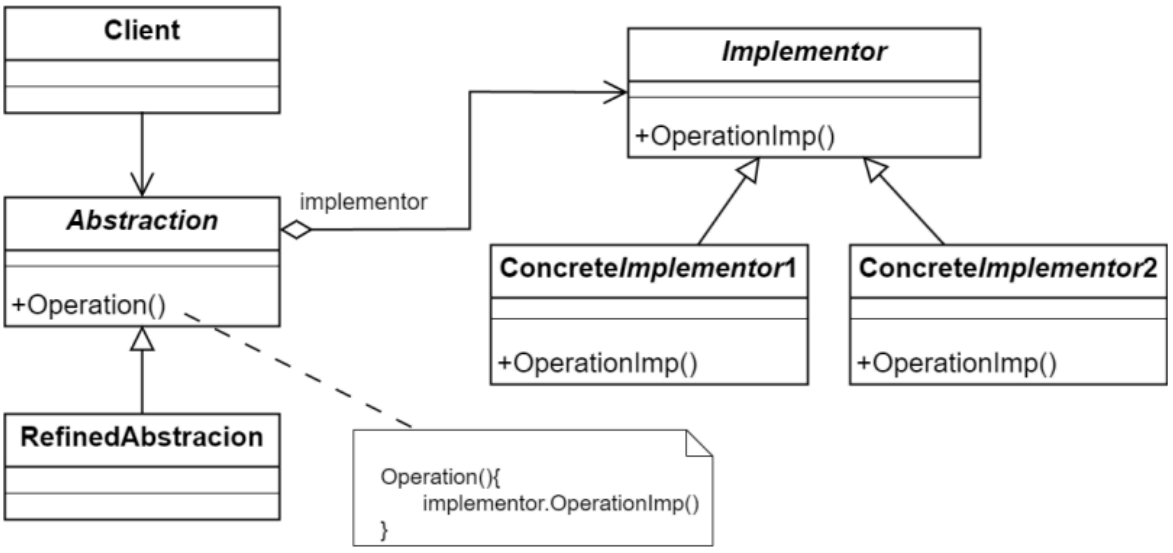
- виконують «реальну роботу»
- між собою можуть мати складні зв’язки
- Фасад викликає їх у правильній послідовності

Взаємодія: Client → Facade → Subsystem.

7. Яке призначення шаблону «Міст»?

Шаблон «Bridge» (міст) використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами.

8. Нарисуйте структуру шаблону «Міст».



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

Abstraction:

- верхній рівень, яким користується клієнт
- містить посилання на Implementor
- метод Operation() делегує роботу реалізатору

RefinedAbstraction:

- розширює поведінку Abstraction

Implementor (інтерфейс / абстрактний клас):

- описує «низькорівневі» операції
- не залежить від абстракції

ConcreteImplementor1 / ConcreteImplementor2:

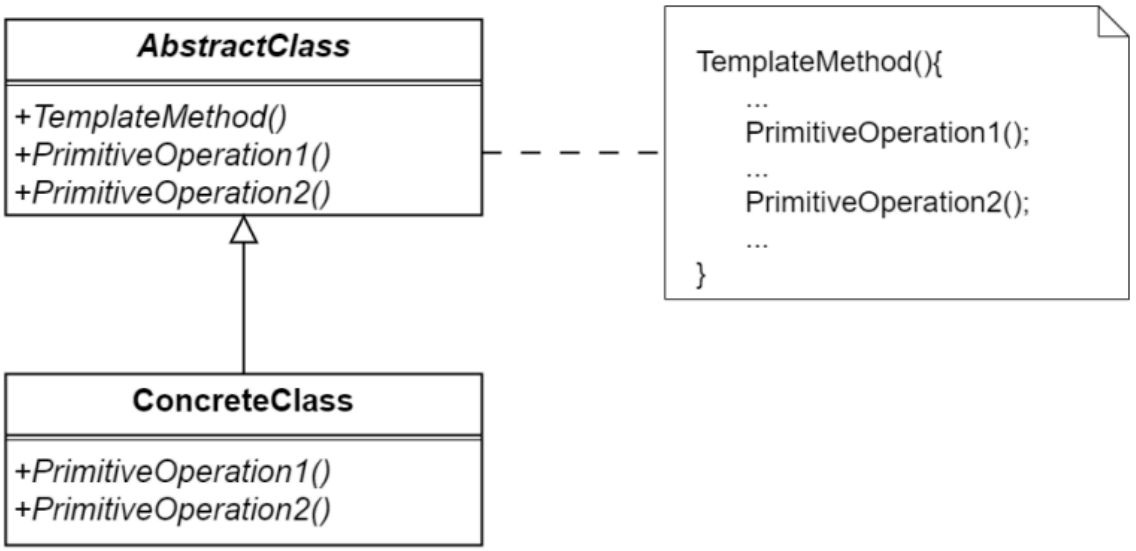
- різні способи виконання операцій
- напр. CanvasRenderer, SvgRenderer у твоєму редакторі

Взаємодія: Client → Abstraction → Implementor → ConcreteImplementor.

10.Яке призначення шаблону «Шаблоний метод»?

Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам . Можна привести в приклад формування вебсторінки: необхідно додати заголовки, вміст сторінки, файли, що додаються, і нижню частину сторінки. Код для додавання вмісту сторінки може бути абстрактним і реалізовуватися в різних класах – AspNetCompiler, HtmlCompiler, PhpCompiler і т.п. Додавання всіх інших елементів виконується за допомогою вихідного абстрактного класу з алгоритмом.

11.Нарисуйте структуру шаблону «Шаблоний метод».



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

AbstractClass:

- містить готовий алгоритм templateMethod()
- викликає в ньому «примітивні операції»

- деякі кроки реалізовані тут, інші абстрактні

ConcreteClass:

- реалізує абстрактні примітивні операції
- визначає конкретну поведінку для окремих кроків алгоритму

Взаємодія: Client → TemplateMethod() → PrimitiveOperation1() / PrimitiveOperation2().

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

Даний шаблон дещо нагадує шаблон «Фабричний метод», однак область його використання абсолютно інша – для покрокового визначення конкретного алгоритму.

14.Яку функціональність додає шаблон «Міст»?

Дає можливість:

- незалежно розвивати абстракцію (фігури) й реалізацію (рендерерів);
- комбінувати будь-який підвид абстракції з будь-якою реалізацією;
- зменшити кількість класів у порівнянні з варіантом «успадкування всього від всього»