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 розроблення програмного забезпечення»

Тема: «Патерни проектування»

«2. HTTP–сервер»

Виконав:
студент групи – ІА–32
Непотачев Іван Дмитрович

Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Патерни проектування.

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

Тема проєкту: HTTP–сервер (state, builder, factory method, mediator, composite, p2p) Сервер повинен мати можливість розпізнавати вхідні запити і формувати коректні відповіді (згідно протоколу HTTP), надавати сторінки chtml (html сторінки з додаванням найпростіших C# конструкцій на розсуд студента), вести статистику вхідних запитів, обробку запитів у багатопотоковому/подієвому режимах.

Зміст

Теоретичні відомості 2
Хід роботи 4
Реалізація шаблону проєктування для майбутньої системи 4
Зображення структури шаблону 8
Висновки 10
Питання до лабораторної роботи 10

Теоретичні відомості

Шаблон «Mediator»
Призначення патерну: Шаблон «Mediator» (посередник) використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного) [6]. Даний шаблон схожий на шаблон «команда», проте в даному випадку замість зберігання даних про конкретну дію, зберігаються дані про взаємодії між компонентами.

Даний 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».

«Медіатор» нагадує диригента при управлінні оркестром. Диригент стежить за тим, щоб кожен інструмент грав в правильний час і в злагоді з іншими інструментами. Функції «медіатора» повністю це повторюють.

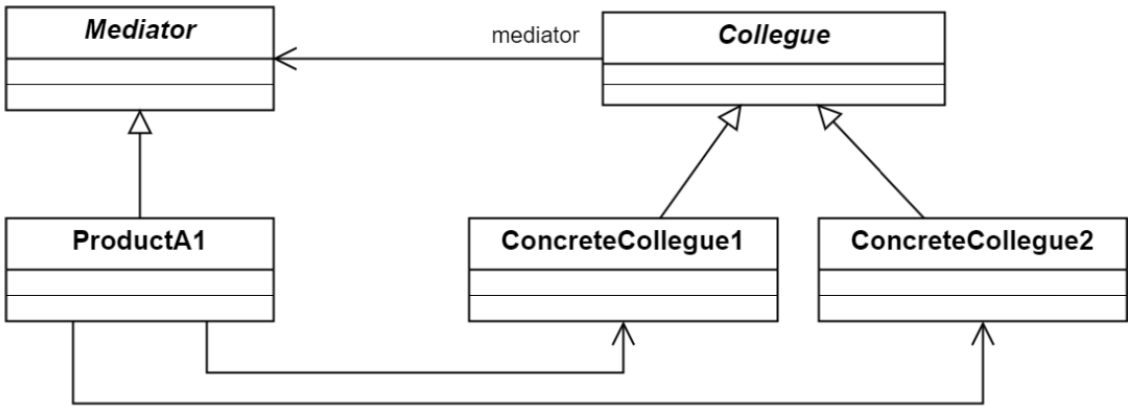


Рисунок 7.1. Структура патерна «Медіатор»

Проблема: Ви розробляєте систему зі складною візуальною частино. Головна форма з якою працює користувач складається з великою кількості візуальних компонентів, які в свою чергу складаються з елементарних візуальних компонентів. Для такої складної форми, як правило, є багато логіки, коли дії в одному компоненті повинні впливати на інші компоненти.

Наприклад, коли ви вибрали чекбокс, то після цього відключається можливість редагування даних в деяких частинах форми, деякі блоки мають приховатися, а інші навпаки, показатися. В такій ситуації, коли на формі може бути сотня, а то і більше візуальних контролів, то зв'язки між ними можуть вимірюватися тисячами. Добавляти зміни в функціонал на таких формах дуже складно, а знайти та виправити помилку, взагалі, може бути дуже затратно по витраченому часу.

Рішення: В таких ситуаціях дуже добре підходить патерн «Mediator» (Посередник). При реалізації такого патерна, ми добавляємо новий клас посередник, і всі взаємодії між компонентами повинні відбуватися вже через нього. За рахунок цього компоненти вже не знають один про одного, а лише знають про посередника і для відпрацювання логіки звертаються вже до нього. А вже посередник взаємодіє з іншими компонентами для відпрацювання цієї логіки.

Основною перевагою, яку ми отримуємо при такому підході є сильно зменшена зв'язність між компонентами. А за рахунок цього стає набагато простіше розібратися з взаємодією між об'єктами коли потрібно добавити на форму нові об'єкти або добавити якийсь функціонал. Якщо компоненти будуть взаємодіяти не напряму з посередником, я через загальний інтерфейс посередника, то тоді можна буде підміняти посередника, наприклад, на тестового, щоб емулювати тестові випадки для перевірки.

Переваги та недоліки:

+ Організація взаємодії між об’єктами лише через посередника спрощує розуміння та супроводження такого коду.

+ Додавання нових посередників без зміни існуючих компонентів дозволяє розширювати систему без зміни існуючого коду.

+ Зменшення залежностей між об’єктами підвищує гнучкість системи.

– Посередник, з часом, може перетворитися на дуже складний об’єкт, який робить все («God Object»).

Хід роботи

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.

Реалізація шаблону проєктування для майбутньої системи

Для реалізації HTTP–сервера використано шаблон проєктування Mediator, оскільки він забезпечує централізоване управління взаємодією між різними компонентами сервера та усуває прямий зв'язок між ними. Це дозволяє ефективно координувати роботу обробника запитів, системи логування та станів сервера, забезпечуючи чітке розділення відповідальностей.

Шаблон Mediator реалізовано у класах ServerMediator та ConcreteServerMediator, які виступають посередниками між основними компонентами системи. Такий підхід є особливо ефективним у роботі HTTP–сервера, де необхідно координувати складні взаємодії між різними підсистемами при обробці запитів.

```java
package mediator;

import model.HttpRequest;
import model.HttpResponse;

public interface ServerMediator {  no usages  1 implementation
    HttpResponse handleRequest(HttpRequest request);  no usages  1 implementation
    void logRequest(HttpRequest request);  no usages  1 implementation
}
```

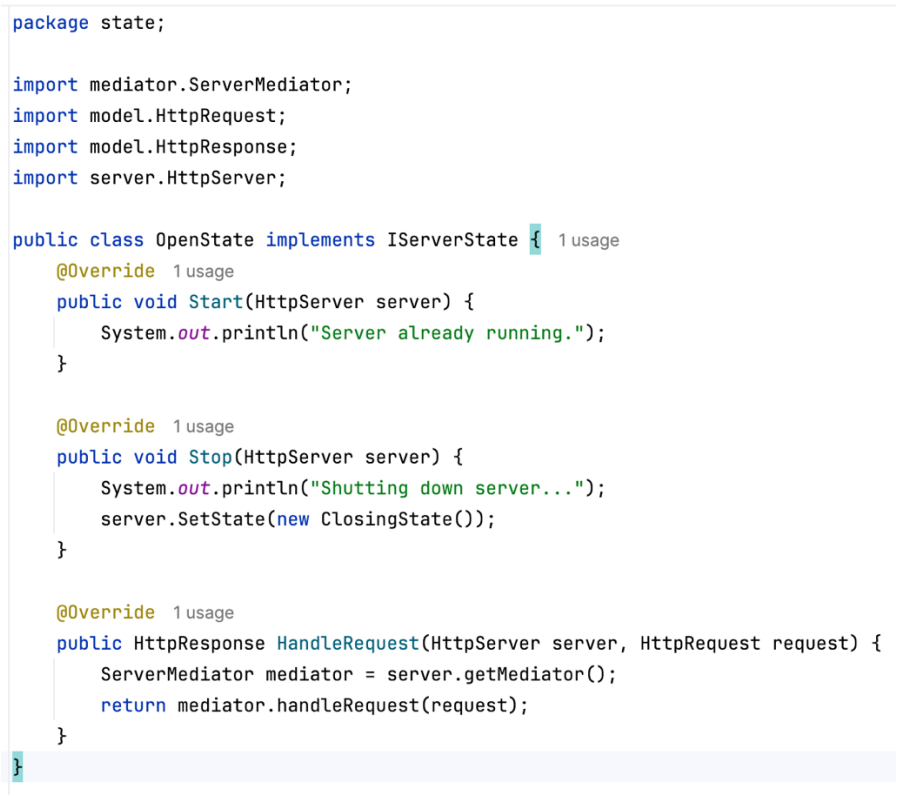
Рис. 1 – Код інтерфейсу ServerMediator

```java
package mediator;

import model.HttpRequest;
import model.HttpResponse;
import server.HttpServer;
import server.RequestHandler;
import server.Statistics;

public class ConcreteServerMediator implements ServerMediator {  2 usages
    private final RequestHandler handler;  2 usages
    private final Statistics statistics;  2 usages

    public ConcreteServerMediator(HttpServer server, RequestHandler handler, Statistics statistics) {  1 usage
        this.handler = handler;
        this.statistics = statistics;
    }

    @Override  1 usage
    public HttpResponse handleRequest(HttpRequest request) {
        logRequest(request);
        return handler.Handle(request);
    }

    @Override  1 usage
    public void logRequest(HttpRequest request) {
        statistics.LogRequest(request);
    }
}
```

Рис. 2 – Код класу ConcreteServerMediator

```java
public class HttpServer {  22 usages
    private IServerState state;  5 usages
    private final int port;  2 usages
    private ServerMediator mediator;  3 usages

    public HttpServer(int port, ServerMediator mediator) {  1 usage
        this.port = port;
        this.state = new InitializingState();
        this.mediator = mediator;
    }

    public void SetState(IServerState newState) {  2 usages
        this.state = newState;
    }

    public void Start() {  1 usage
        state.Start( server: this);
    }

    public void Stop() {  1 usage
        state.Stop( server: this);
    }

    public HttpResponse HandleRequest(HttpRequest req) {  3 usages
        return state.HandleRequest( server: this, req);
    }

    public int getPort() {  1 usage
        return port;
    }

    public void setMediator(ServerMediator mediator) {  1 usage
        this.mediator = mediator;
    }

    public ServerMediator getMediator() {  1 usage
        return mediator;
    }
}
```

Рис. 3 – Код класу HttpServer



```java
package state;

import mediator.ServerMediator;
import model.HttpRequest;
import model.HttpResponse;
import server.HttpServer;

public class OpenState implements IServerState {  1 usage
    @Override  1 usage
    public void Start(HttpServer server) {
        System.out.println("Server already running.");
    }

    @Override  1 usage
    public void Stop(HttpServer server) {
        System.out.println("Shutting down server...");
        server.SetState(new ClosingState());
    }

    @Override  1 usage
    public HttpResponse HandleRequest(HttpServer server, HttpRequest request) {
        ServerMediator mediator = server.getMediator();
        return mediator.handleRequest(request);
    }
}
```

Рис. 4 – Код класу OpenState

Застосування цього шаблону забезпечує:

1. Усування прямих залежностей між компонентами: Класи HttpServer, RequestHandler та Statistics більше не мають прямих посилань один на одного. Вся комунікація відбувається через посередника, що зменшує зв'язаність системи.
2. Централізація логіки взаємодії: Клас ConcreteServerMediator інкапсулює складну логіку координації між компонентами. При обробці запиту він автоматично викликає логування через Statistics та передає запит на обробку до RequestHandler.
3. Спрощення розширення функціоналу: Додавання нових компонентів або зміна існуючої логіки взаємодії потребує модифікації лише класу–посередника, без змін в інших компонентах системи.
4. Покращена підтримка та тестування: Кожен компонент може тестуватися ізольовано, а посередник дозволяє легко відстежувати всі взаємодії в системі.

У нашому випадку шаблон Mediator забезпечує ефективну координацію між станами сервера та іншими компонентами. Особливо це корисно в класі OpenState, де обробка запиту делегується посереднику.

Зображення структури шаблону

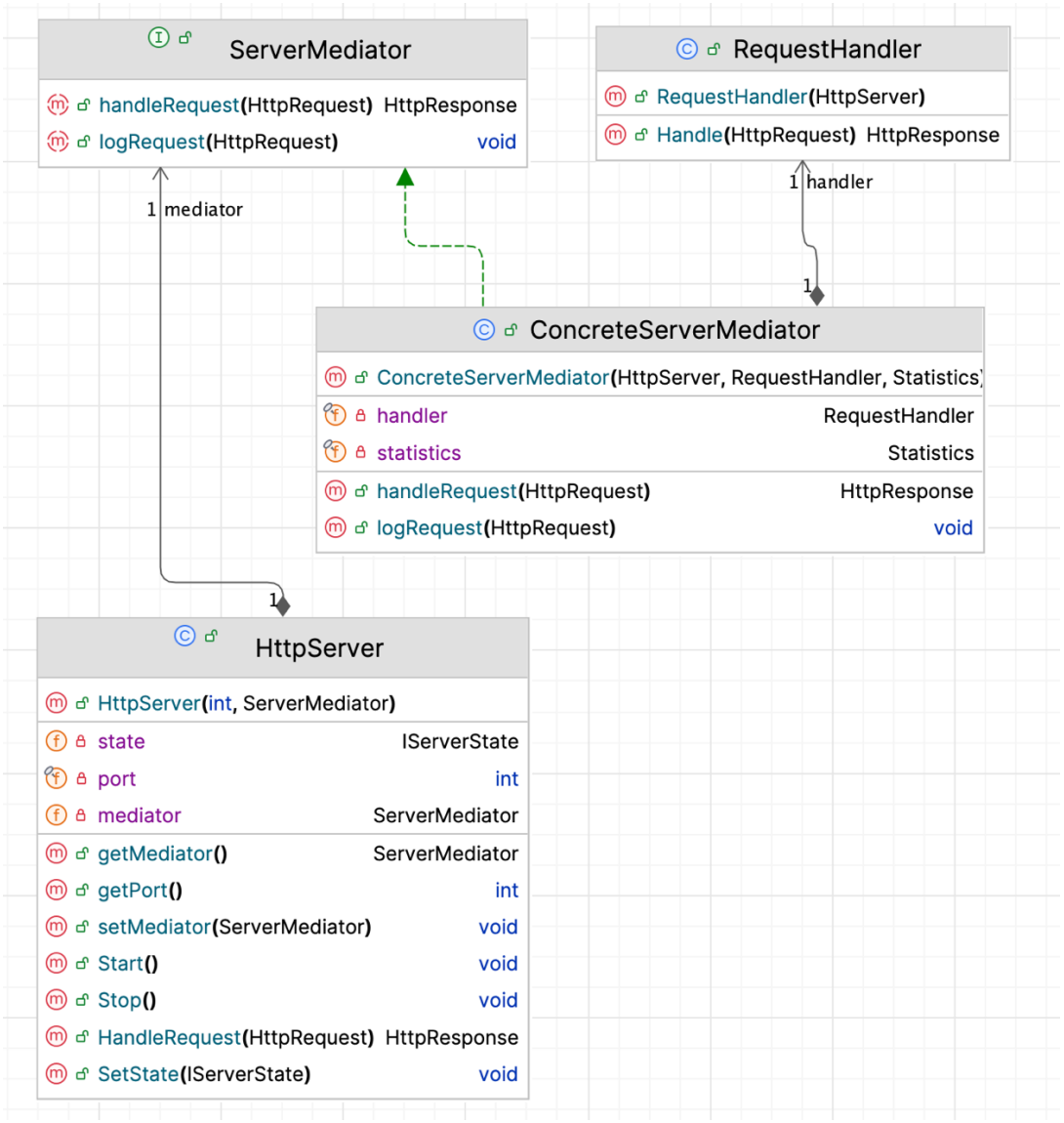


Рис. 2 – Структура шаблону Mediator

Структура реалізації:

- ServerMediator – інтерфейс, що визначає методи для взаємодії між компонентами.

- ConcreteServerMediator – конкретний посередник, що реалізує логіку координації між RequestHandler та Statistics

- HttpServer – містить посилання на посередника та делегує йому обробку запитів у робочому стані

- RequestHandler – клас, що спеціалізується на обробці HTTP–запитів та генерації відповідей. Через посередника він отримує запити для обробки, при цьому не маючи прямих залежностей від системи логування чи станів сервера.

Такий підхід особливо ефективний у поєднанні з патерном State, оскільки посередник дозволяє кожному стану сервера ефективно взаємодіяти з іншими компонентами системи без необхідності знати про їх внутрішню реалізацію. Це забезпечує високий рівень інкапсуляції та спрощує додавання нових станів сервера в майбутньому.

Посилання на репозиторій: https://github.com/IvanGodPro24/trpz
Посилання на звіти: https://github.com/IvanGodPro24/trpz_reports

Висновки

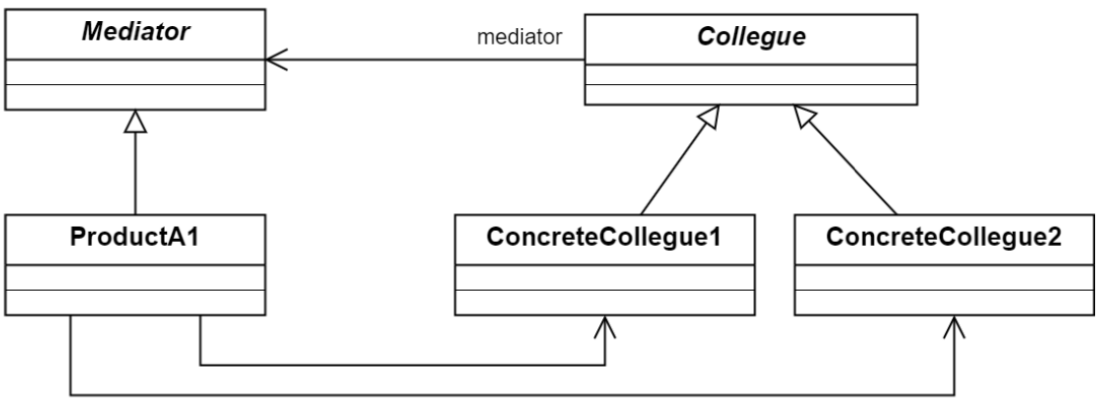
Висновки: під час виконання лабораторної роботи, ми вивчили структуру шаблону Mediator та застосували його для реалізації HTTP-сервера. Шаблон Mediator дозволив усунути прямі залежності між основними компонентами системи, що значно підвищило модульність нашої архітектури. Клас ConcreteServerMediator став єдиним центром координації, який інкапсулював складну логіку взаємодії між компонентами, спростивши їхню індивідуальну реалізацію. Також, шаблон Mediator ефективно поєднався з існуючою реалізацією патерну State, де OpenState делегує обробку запитів посереднику, що демонструє гнучкість шаблону у роботі з іншими архітектурними патернами. Реалізація демонструє, що шаблон Mediator є оптимальним вибором для складних систем, де необхідно координувати взаємодію між багатьма об'єктами, забезпечуючи при цьому низьку зв'язаність та високу гнучкість.

Питання до лабораторної роботи

1. Яке призначення шаблону «Посередник»?

Шаблон «Посередник» (Mediator) забезпечує централізовану взаємодію між об’єктами, щоб вони не взаємодіяли безпосередньо один з одним, зменшуючи кількість зв’язків між класами.

2. Нарисуйте структуру шаблону «Посередник».



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

Mediator – інтерфейс, який визначає метод сповіщення.

ConcreteCollegue – реалізує координацію між компонентами.

Collegue – базовий клас об’єктів, які спілкуються через посередника.

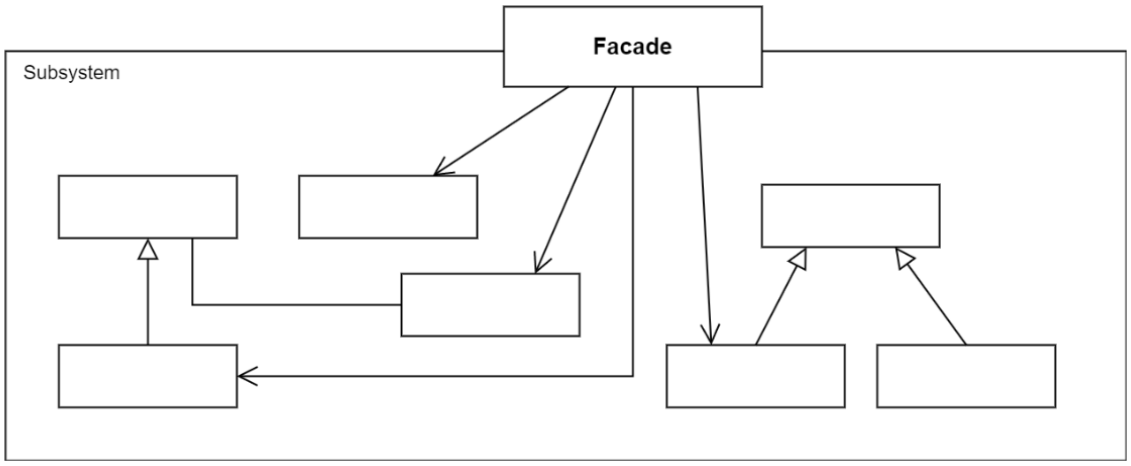
ConcreteComponent – конкретний об’єкт, що надсилає повідомлення посереднику.

Компоненти не звертаються один до одного напряму, а відправляють запити посереднику, який вирішує, хто має на них реагувати.

4. Яке призначення шаблону «Фасад»?

Шаблон «Фасад» (Facade) спрощує роботу зі складною підсистемою, надаючи уніфікований інтерфейс для взаємодії з нею.

5. Нарисуйте структуру шаблону «Фасад».



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

Facade – забезпечує спрощений інтерфейс до складної системи.

Subsystem classes – виконують конкретні функції підсистеми.

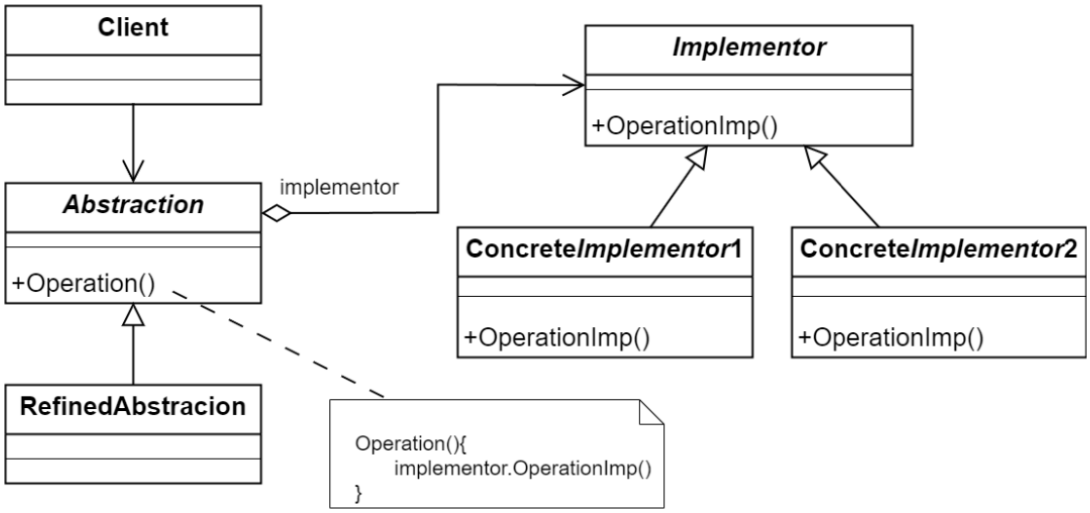
Client – звертається до Facade, а не до підсистем напряму.

Client викликає метод Facade, який усередині звертається до кількох класів підсистеми, приховуючи складність.

7. Яке призначення шаблону «Міст»?

Шаблон «Міст» (Bridge) розділяє абстракцію та реалізацію так, щоб вони могли змінюватися незалежно одна від одної.

8. Нарисуйте структуру шаблону «Міст».



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

Abstraction – визначає високорівневий інтерфейс.

RefinedAbstraction – розширює функціональність Abstraction.

Implementor – інтерфейс для реалізації.

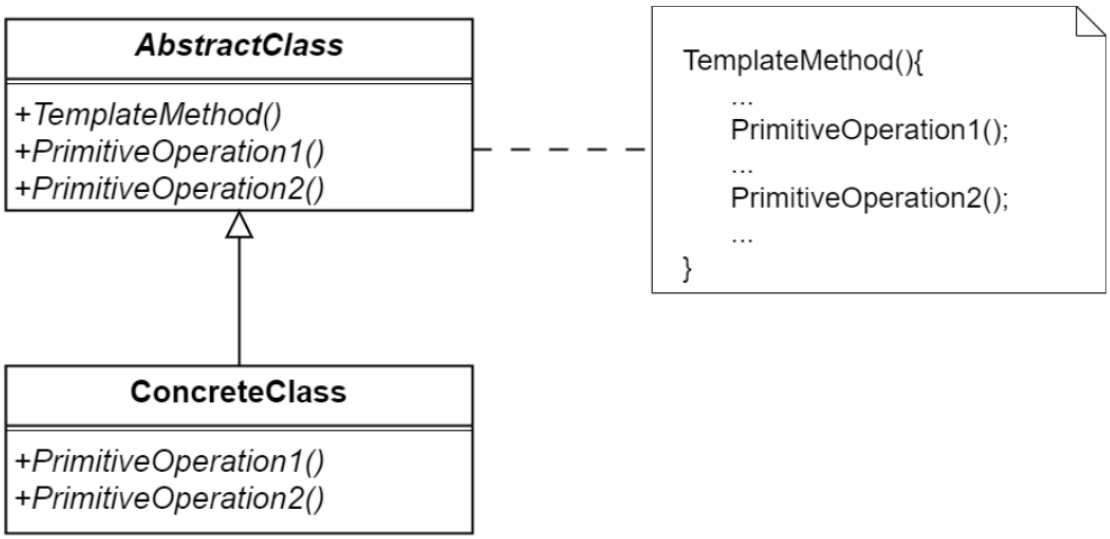
ConcreteImplementor – конкретна реалізація інтерфейсу Implementor.

Abstraction зберігає посилання на Implementor і делегує йому частину роботи. Це дозволяє змінювати реалізацію незалежно від абстракції.

10. Яке призначення шаблону «Шаблоний метод»?

Шаблон «Шаблонний метод» (Template Method) визначає кістяк алгоритму в базовому класі, дозволяючи підкласам перевизначати окремі кроки алгоритму без зміни його структури.

11. Нарисуйте структуру шаблону «Шаблоний метод».



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

AbstractClass – визначає структуру алгоритму через templateMethod() і абстрактні методи.

ConcreteClass – реалізує конкретні кроки алгоритму.

Клас AbstractClass викликає методи, реалізовані в ConcreteClass, щоб виконати певні частини алгоритму.

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

Шаблонний метод задає структуру алгоритму, а фабричний метод контролює створення об’єктів.У Шаблонному методі підкласи змінюють окремі кроки алгоритму, а в фабричному – визначають, який об’єкт створювати.

14. Яку функціональність додає шаблон «Міст»?

Шаблон «Міст» додає гнучкість у зміні або розширенні як абстракції, так і реалізації незалежно одна від одної. Це дозволяє уникнути множинного успадкуванн, розділити логіку на рівні інтерфейсу та реалізації та легко змінювати реалізації в процесі виконання програми.