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

Лабораторна робота № 8

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 «Composite», «Flyweight» (Пристосуванець), «Interpreter», «Visitor» 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

Шаблон «Composite»
Призначення: Шаблон використовується для складання об'єктів в деревоподібну структуру для подання ієрархій типу «частина цілого». Даний шаблон дозволяє уніфіковано обробляти як поодинокі об'єкти, так і об'єкти з вкладеністю [6].

Простим прикладом може служити складання компонентів всередині звичайної форми. Форма може містити дочірні елементи (поля для введення тексту, цифр, написи, малюнки тощо); дочірні елементи можуть в свою чергу містити інші елементи. Наприклад, при виконанні операції розтягування форми необхідно, щоб вся ієрархія розтягнулася відповідним чином. В такому випадку форма розглядається як композитний об'єкт і операція розтягування застосовується до всіх дочірніх елементів рекурсивно.

Даний шаблон зручно використовувати при необхідності подання та обробки ієрархій об'єктів. Крім того, патерн «Composite» (Компонувальник) краще використовувати, коли ви представляєте структуру даних у вигляді дерева.

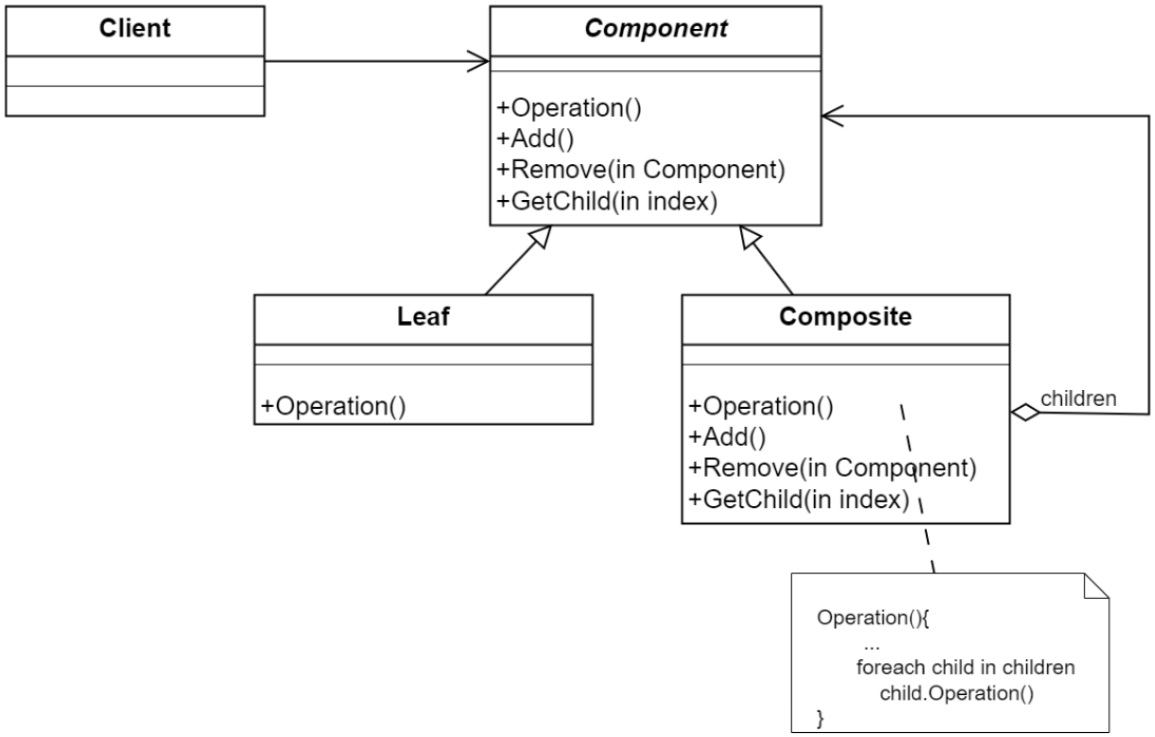


Рисунок 8.1. Структура патерна «Компонувальник»

Проблема: Ви розробляєте систему керування проєктами. Кожен проєкт складається із наборів функцій, кожна функція з userstory, а кожна userstory в свою чергу із задач по її реалізації. Ви реалізуєте функціонал відображення оціночної вартості робіт по кожній із функцій, а також відображення чи всі userstory були оцінені. Це потрібно бізнес-аналітики, щоб розуміти, що всі userstory розробниками були розглянуті і оцінені, а також обговорити необхідність реалізації того чи іншого функціоналу на основі попередньої оцінки.

Рішення: Крашим підходом в даній ситуації буде використання патерну Компоновщик. Класи що представляють функції, userstory, задачі будуть наслідуватися від одного інтерфейсу ITask, функції (Feature) та Userstory будуть складними об'єктами і міститимуть колекції об'єктів ITask, а задачі (Task) будуть представляли кінцеві об'єкти без дочірніх елементів.

Для розрахунку оціночної вартосі робіт, в ITask інтефейс додаємо метод GetEstimatedPoints(). В класах-компоновщиках методи GetEstimatedPoints() реалізовуємо як обхід всіх дочірніх елементів та сумування результатів відповідей GetEstimatedPoints().

Таким чином, візуальні форми будуть працювати з колекцією елементів ITask і їм не потрібно буде знати конкретні типи дочірніх класів з якими вони працюють. В результаті логіка візуальних форм виходить достатньо простою і вона не буде містити бізнес-логіки розрахунку

загальної оцінки по проєктам, а просто викликає метод GetEstimatedPoints() не замислюючись містить цей об'єкт дочірні об'єкти чи ні.

Переваги та недоліки:

+ Спрощує представлення деревоподібної структури.

+ Додає гнучкості в роботі з складними об’єктами та рекурсивними операціями.

+ Дозволяє додавати та видаляти об’єкти в ієрархії без впливу на клієнтський код.

- Потрібні додаткові зусилля для початкового впровадження.

- Вимагає гарно спроєктованого загального інтерфейсу.

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 Composite, оскільки він забезпечує єдиний інтерфейс для роботи з окремими HTML-елементами та їх складними композиціями. Це дозволяє динамічно будувати ієрархічні HTML-структури для різних сторінок сервера, забезпечуючи гнучкість та масштабованість при генерації вмісту.

Шаблон Composite реалізовано у класах HtmlComponent, HtmlElement та HtmlComposite, які відповідають за створення та рендеринг HTML-вмісту. Такий підхід є особливо ефективним у роботі HTTP-сервера, де необхідно генерувати структуровані HTML-сторінки з різним рівнем вкладеності.

```java
package composite;

public interface HtmlComponent {  6 usages  2 implementations
    String render();  4 usages  2 implementations
}
```

Рис. 1 – Код інтерфейсу HtmlComponent

```java
package composite;

public class HtmlElement implements HtmlComponent {  no usages
    private final String tag;  3 usages
    private final String content;  2 usages

    public HtmlElement(String tag, String content) {  no usages
        this.tag = tag;
        this.content = content;
    }

    @Override  no usages
    public String render() {
        return "<" + tag + ">" + content + "</" + tag + ">";
    }
}
```

Рис. 2 – Код класу HtmlElement

```java
package composite;

import java.util.ArrayList;
import java.util.List;

public class HtmlComposite implements HtmlComponent {  13 usages
    private final String tag;  3 usages
    private final List<HtmlComponent> children = new ArrayList<>();  3 usages

    public HtmlComposite(String tag) {  6 usages
        this.tag = tag;
    }

    public void add(HtmlComponent component) {
        children.add(component);
    }

    public void remove(HtmlComponent component) {  no usages
        children.remove(component);
    }

    @Override  4 usages
    public String render() {
        StringBuilder builder = new StringBuilder();
        builder.append("<").append(tag).append(">");
        for (HtmlComponent child : children) {
            builder.append(child.render());
        }
        builder.append("</").append(tag).append(">");
        return builder.toString();
    }
}
```

Рис. 3 – Код класу HtmlComposite

```java
package server;

import composite.HtmlComposite;
import composite.HtmlElement;
import factory.ErrorResponseCreator;
import factory.HttpResponseCreator;
import factory.SuccessResponseCreator;
import model.HttpRequest;
import model.HttpResponse;

public class RequestHandler {  6 usages
    public RequestHandler() {}  1 usage

    public HttpResponse Handle(HttpRequest req) {  1 usage
        String url = req.getUrl();
        String body = switch (url) {
            case "/home" -> buildHomePage();
            case "/about" -> buildAboutPage();
            case "/contact" -> buildContactPage();
            default -> "<h1>404 Page Not Found</h1>";
        };

        int statusCode = (url.equals("/home") || url.equals("/about") || url.equals("/contact")) ? 200 : 404;

        HttpResponseCreator creator = (statusCode == 200)
                ? new SuccessResponseCreator()
                : new ErrorResponseCreator();

        return creator.createResponse(statusCode, body);
    }

    private String buildHomePage() {  1 usage
        HtmlComposite html = new HtmlComposite( tag: "html");
        HtmlComposite body = new HtmlComposite( tag: "body");
        body.add(new HtmlElement( tag: "h1",  content: "Welcome to Home!"));
        body.add(new HtmlElement( tag: "p",  content: "This page is generated using the Composite pattern."));
        html.add(body);
        return html.render();
    }
private String buildAboutPage() {  1 usage
    HtmlComposite html = new HtmlComposite( tag: "html");
    HtmlComposite body = new HtmlComposite( tag: "body");
    body.add(new HtmlElement( tag: "h1",  content: "About Us"));
    body.add(new HtmlElement( tag: "p",  content: "This page demonstrates Composite usage for dynamic HTML."));
    html.add(body);
    return html.render();
}

private String buildContactPage() {  1 usage
    HtmlComposite html = new HtmlComposite( tag: "html");
    HtmlComposite body = new HtmlComposite( tag: "body");
    body.add(new HtmlElement( tag: "h1",  content: "Contact information"));
    body.add(new HtmlElement( tag: "p",  content: "Email: contact@server.com"));
    html.add(body);
    return html.render();
}
```

Рис. 4 – Код класу RequestHandler

Застосування цього шаблону забезпечує:

1. Єдиний інтерфейс для простих та складних елементів: Інтерфейс HtmlComponent визначає метод render(), який реалізує як прості елементи (HtmlElement), так і складні композиції (HtmlComposite). Це дозволяє клієнтському коду працювати з будь-якими компонентами однаково.
2. Рекурсивну побудову складних структур: Клас HtmlComposite може містити як прості елементи, так і інші композиції, що дозволяє створювати складні ієрархії HTML-тегів:
3. Гнучкість та розширюваність: Додавання нових типів HTML-компонентів не вимагає змін в існуючій логіці рендерингу.

4. Спрощення коду генерації сторінок: Клас RequestHandler використовує компоненти для побудови різних сторінок, приховуючи складність HTML-структури.

У нашому випадку шаблон Composite забезпечує чітке розділення відповідальності між логікою обробки HTTP-запитів та логікою генерації HTML-вмісту. Це дозволяє легко змінювати структуру сторінок, додавати нові типи контенту та підтримувати читабельність та гнучкість коду.

Зображення структури шаблону

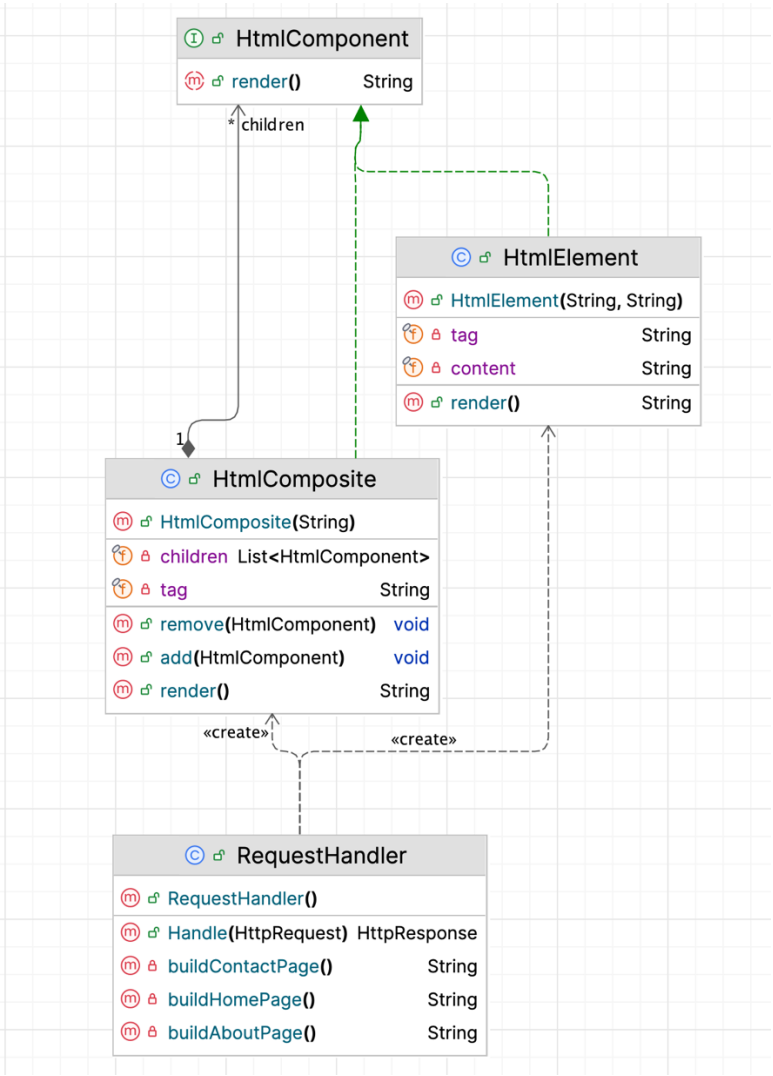


Рис. 2 – Структура шаблону Composite

Структура реалізації:

- HtmlComponent - інтерфейс, що визначає метод render() для всіх HTML-компонентів

- HtmlElement - клас-листок, що представляє простий HTML-тег з текстовим вмістом

- HtmlComposite - клас-контейнер, що може містити інші компоненти та формує ієрархічні структури

- RequestHandler – використовує компоненти для динамічної генерації HTML-вмісту різних сторінок сервера

Такий підхід особливо ефективний у поєднанні з патернами Factory Method та Mediator, оскільки дозволяє створювати складний HTML-вміст у стандартизований спосіб, який потім використовується фабриками для генерації HTTP-відповідей та координується посередником для обробки запитів.

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

Висновки

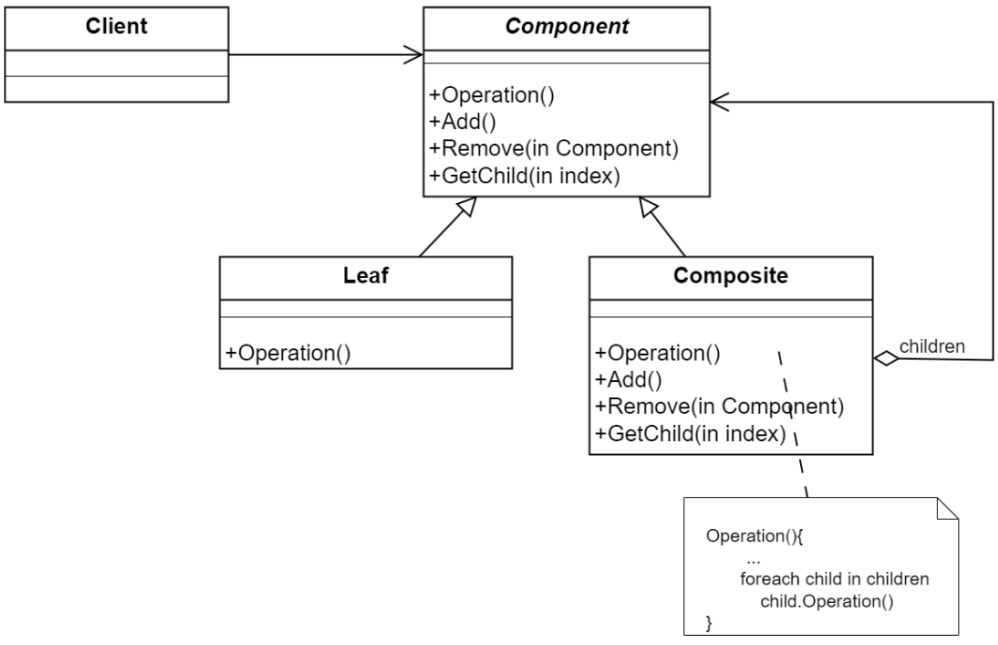
Висновки: під час виконання лабораторної роботи, ми вивчили структуру шаблону Composite та застосували його для реалізації генерації HTML-вмісту в HTTP-сервері. Шаблон Composite дозволив створити єдиний інтерфейс для роботи з простими HTML-елементами та складними ієрархічними структурами, що значно підвищило гнучкість та масштабованість нашої системи. Класи HtmlElement та HtmlComposite реалізують спільний інтерфейс HtmlComponent, що дозволяє будувати складні HTML-сторінки рекурсивним способом, приховуючи складність структури. Шаблон Composite ефективно поєднався з існуючою реалізацією патернів Factory Method та Mediator, де RequestHandler використовує компоненти для динамічної генерації вмісту різних сторінок сервера. Застосування Composite дозволило легко створювати та модифікувати структуру HTML-сторінок через методи buildHomePage(), buildAboutPage() та buildContactPage(), спрощуючи додавання нових типів контенту в майбутньому.

Питання до лабораторної роботи

1. Яке призначення шаблону «Композит»?

Шаблон «Композит» (Composite) використовується для представлення ієрархічних деревоподібних структур, де окремі об’єкти та групи об’єктів обробляються однаково.

2. Нарисуйте структуру шаблону «Композит».



3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?
Component – спільний інтерфейс для всіх елементів дерева.

Leaf – представник кінцевого об’єкта.

Composite – містить колекцію об’єктів типу Component і реалізує методи для роботи з ними.

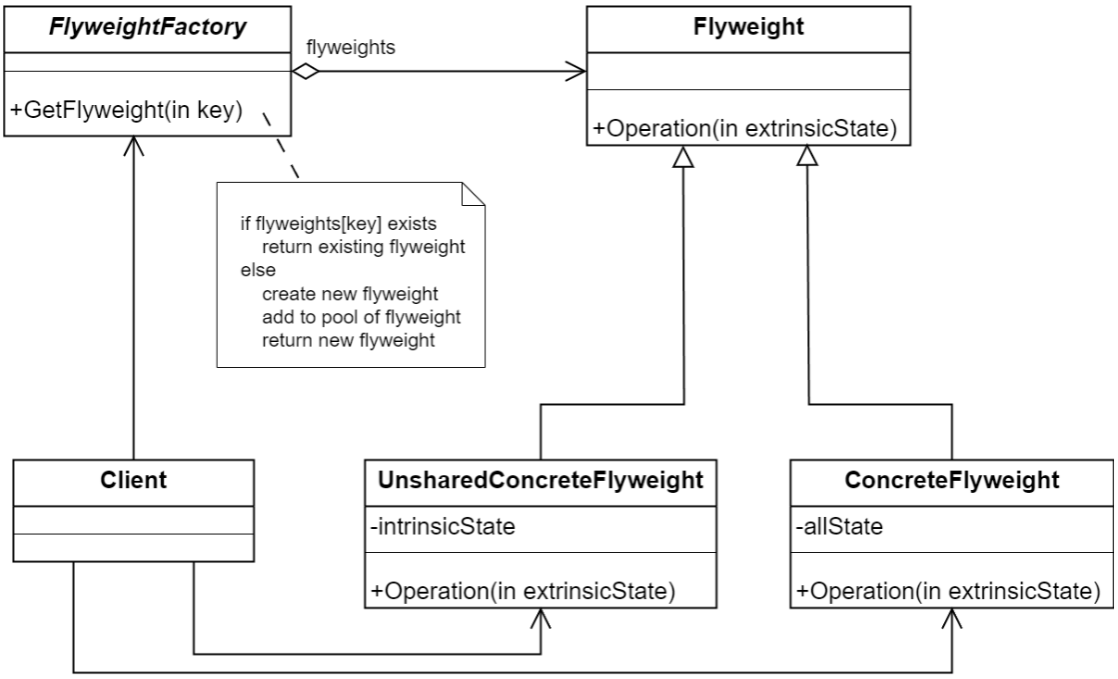
Client – працює з усіма об’єктами через інтерфейс Component.

Composite містить і викликає методи дочірніх компонентів (Leaf або інших Composite).

4. Яке призначення шаблону «Легковаговик»?

Шаблон «Легковаговик» (Flyweight) зменшує споживання пам’яті, дозволяючи розділяти однакові об’єкти між різними контекстами замість створення копій.

5. Нарисуйте структуру шаблону «Легковаговик».



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

Flyweight – спільний інтерфейс для легковагових об’єктів.

ConcreteFlyweight – реалізує спільну частину стану (внутрішній стан).

FlyweightFactory – створює та зберігає екземпляри легковагових об’єктів.

Client – використовує легковаговики, зберігаючи зовнішній стан окремо.

Client запитує об’єкт у FlyweightFactory, яка або повертає існуючий, або створює новий об’єкт.

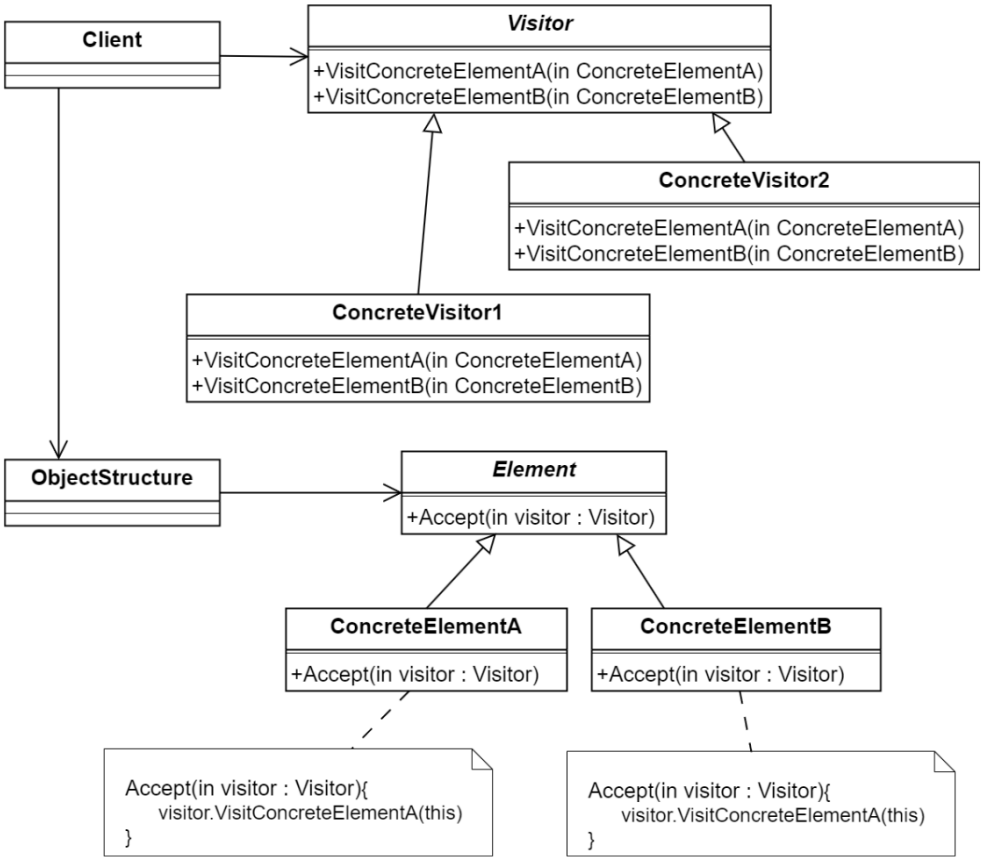
7. Яке призначення шаблону «Інтерпретатор»?

Шаблон «Інтерпретатор» (Interpreter) визначає граматику певної мови та надає механізм для інтерпретації речень цієї мови. Використовується для аналізу, обчислення чи виконання простих мов, виразів або команд.

8. Яке призначення шаблону «Відвідувач»?

Шаблон «Відвідувач» (Visitor) дозволяє додавати нові операції до об’єктів без зміни їхніх класів. Корисний, коли потрібно виконувати різні операції над об’єктами складної структури, наприклад, дерева.

9. Нарисуйте структуру шаблону «Відвідувач».



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

Visitor – інтерфейс для відвідувача, який визначає методи visit() для різних типів елементів.

ConcreteVisitor – реалізує конкретні дії, які виконуються над елементами.

Element – інтерфейс елемента, що приймає відвідувача (accept()).

ConcreteElementA(B) – конкретні елементи, що викликають у відвідувача відповідний метод visit().

Client – створює об’єкти елементів і відвідувача, а потім викликає accept().

Element.accept(visitor) викликає visitor.visit(this), після цього відвідувач виконує дію, залежно від типу елемента.