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»

«Web crawler»

Виконала
студентка групи – ІА–31
Горлач Дар’я Дмитрівна

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: Web crawler (proxy, chain of responsibility, memento, template method, composite, p2p) Веб–сканер повинен вміти розпізнавати структуру сторінок сайту, переходити за посиланнями, збирати необхідну інформацію про зазначений термін, видаляти не семантичні одиниці (рекламу, об'єкти javascript і т.д.), зберігати знайдені дані у вигляді структурованого набору html файлів вести статистику відвіданих сайтів і метадані.

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.

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.

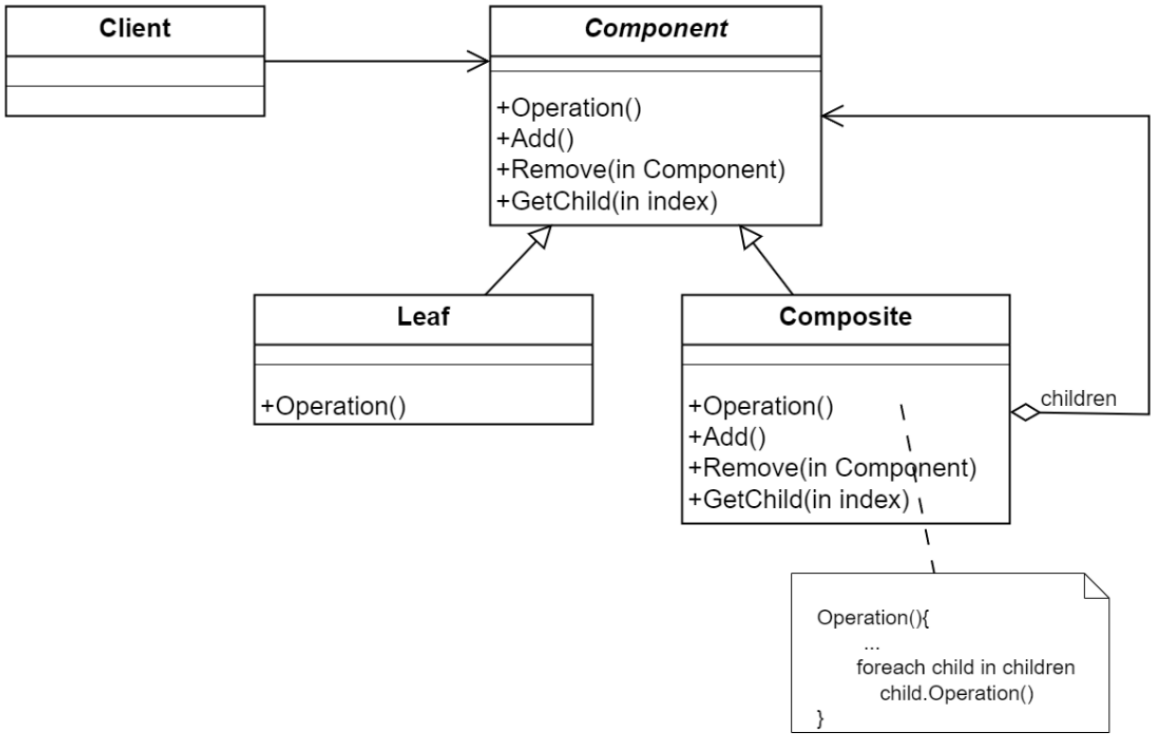


Рисунок 1. Структура патерну «Компонувальник»

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

Реалізація патерну проєктування

Для реалізації Web Crawler використано патерн проєктування Composite, оскільки він забезпечує єдиний інтерфейс для роботи зі складними ієрархічними структурами HTML-документів. Це дозволяє представляти веб-сторінки як деревоподібні структури, де кожен елемент може бути як простим тегом, так і складним компонентом, що містить інші елементи.

Патерн Composite реалізовано у класах HtmlComponent, HtmlComposite та HtmlLeaf, які утворюють ієрархічну структуру для представлення та обробки HTML-контенту. Такий підхід є особливо ефективним у роботі веб-краулера, де необхідно аналізувати та маніпулювати складними веб-сторінками з різним рівнем вкладеності.

```
16 usages   2 implementations   new *
public interface HtmlComponent {
    2 implementations   new *
    void add(HtmlComponent component);
    no usages   2 implementations   new *
    void remove(HtmlComponent component);
    no usages   2 implementations   new *
    List<HtmlComponent> getChildren();
    2 usages   2 implementations   new *
    String render();
}
```

Рис. 2 – Код інтерфейсу HtmlComponent

```java
public class HtmlComposite implements HtmlComponent {
    4 usages
    private final List<HtmlComponent> children = new ArrayList<>();
    3 usages
    private final String tagName;

    no usages  new *
    public HtmlComposite(String tagName) {
        this.tagName = tagName;
    }

    new *
    @Override
    public void add(HtmlComponent component) {
        children.add(component);
    }

    no usages  new *
    @Override
    public void remove(HtmlComponent component) {
        children.remove(component);
    }

    no usages  new *
    @Override
    public List<HtmlComponent> getChildren() {
        return children;
    }

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

Рис. 3 – Код класу HtmlComposite

```
6 usages  new *
public class HtmlLeaf implements HtmlComponent {
    2 usages
    private final String content;

    4 usages  new *
    public HtmlLeaf(String content) {
        this.content = content;
    }

    new *
    @Override
    public void add(HtmlComponent component) {
        throw new UnsupportedOperationException("Cannot add child to leaf");
    }

    no usages  new *
    @Override
    public void remove(HtmlComponent component) {
        throw new UnsupportedOperationException("Cannot remove child from leaf");
    }

    no usages  new *
    @Override
    public List<HtmlComponent> getChildren() {
        return List.of();
    }

    2 usages  new *
    @Override
    public String render() {
        return content;
    }
}
```

Рис. 4 – Код класу HtmlLeaf

Використання цього патерну надає:

1. Єдиний інтерфейс для всіх HTML-компонентів: Інтерфейс HtmlComponent визначає стандартні операції (add(), remove(), getChildren(), render()), які підтримують як прості елементи, так і складні контейнери.
2. Рекурсивну обробку ієрархій: Клас HtmlComposite може містити інші компоненти, дозволяючи створювати складні деревоподібні структури. Метод render() рекурсивно обходить всю ієрархію, генеруючи повний HTML.
3. Гнучкість у представленні веб-сторінок: Можливість будувати сторінки як композиції різних елементів дозволяє легко модифікувати структуру, додавати нові типи компонентів або змінювати існуючі.
4. Інтеграцію з обробниками: Клас PageContext тепер зберігає HTML як HtmlComponent, а не простий рядок, що дозволяє обробникам працювати зі структурованим представленням сторінки.

У нашому випадку патерн Composite забезпечує структуроване представлення веб-контенту, дозволяючи аналізувати та модифікувати окремі частини HTML-документів, створювати складні трансформації сторінок зі збереженням структури та з легкістю додавати нові типи HTML-елементів.

Структура патерну

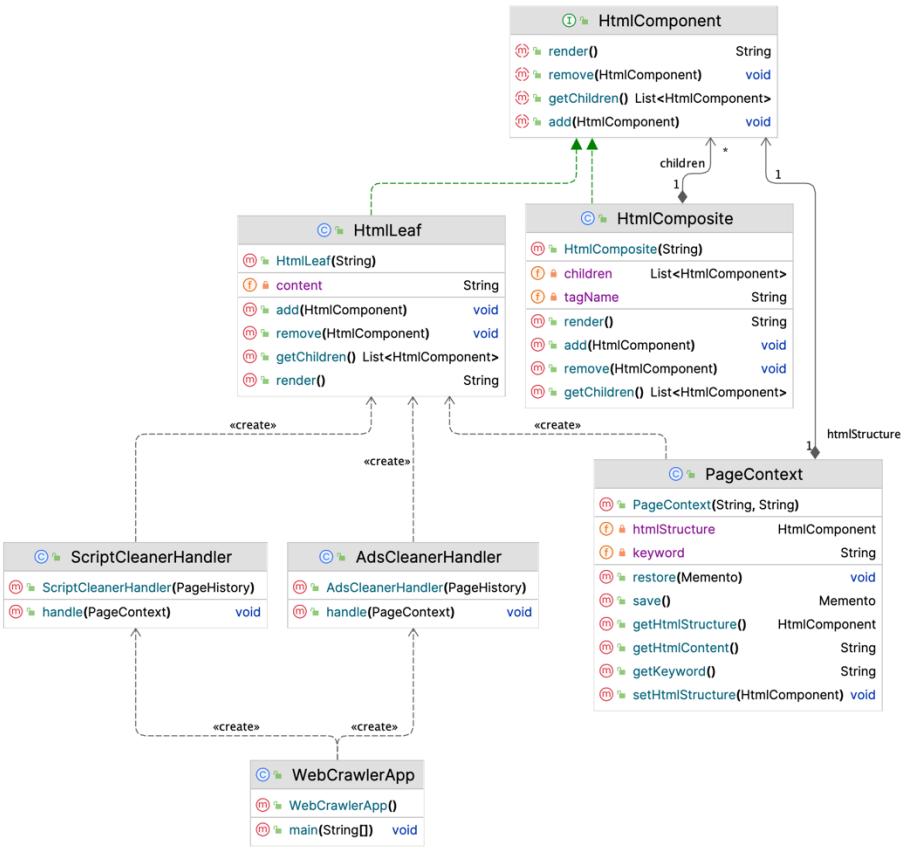


Рис. 5 – Структура патерну «Компонувальник»

Опис реалізації:

- HtmlComponent – інтерфейс, що визначає контракт для всіх HTML-компонентів

- HtmlLeaf – клас-листок, що представляє простий текстовий вміст або атомарний елемент

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

- PageContext – зберігає HTML-сторінку як HtmlComponent для структурованої обробки

Такий підхід особливо ефективний для веб–краулерів, де необхідно аналізувати структуру веб-сторінок, потрібно виконувати селективні

модифікації певних частин документа, а також важливо зберігати семантичну структуру HTML при обробці.

Посилання на репозиторій: https://github.com/dariahorlach/Lab–TRPZ

Висновки

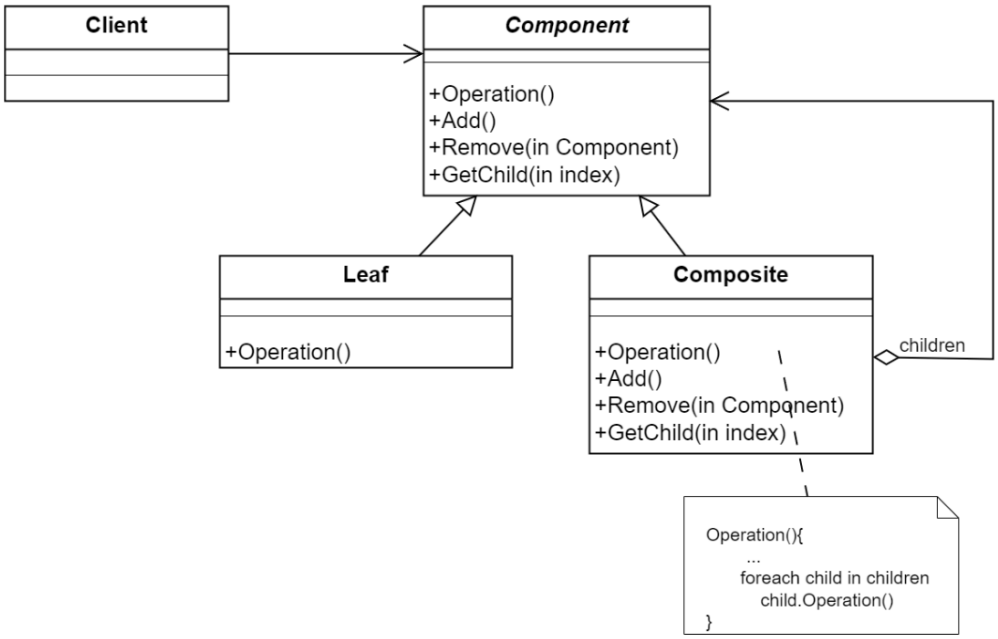
Висновки: під час виконання лабораторної роботи, було реалізовано патерн Composite для веб–краулера, що дозволило створити структуроване представлення HTML-документів у вигляді ієрархічних дерев. Інтерфейс HtmlComponent забезпечив єдиний контракт для роботи як з простими елементами (HtmlLeaf), так і зі складними контейнерами (HtmlComposite), що дозволило обробляти веб-сторінки як семантичні структури, а не плоскі текстові рядки. Реалізація продемонструвала, що патерн Composite ефективно вирішує завдання структурованого представлення веб-контенту шляхом забезпечення єдиного інтерфейсу для всіх типів HTML-компонентів, дозволу рекурсивної обробки складних ієрархій через метод render() та прощення додавання нових типів HTML-елементів та операцій обробки. Архітектура з Composite інтегрувалася з існуючою системою, де PageContext тепер зберігає HTML як структурований HtmlComponent, а не як простий рядок. Це дозволило обробникам працювати зі складними деревоподібними структурами, зберігаючи семантику документа під час трансформацій.

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.

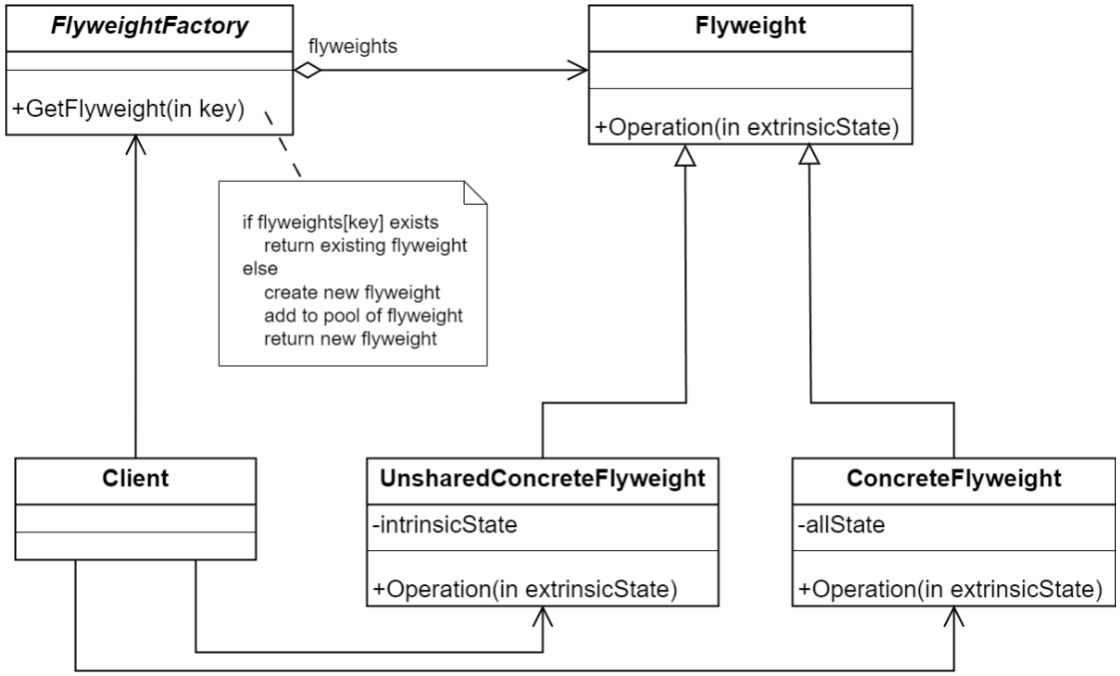
Client – працює з усіма об’єктами через інтерфейс Component.

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.

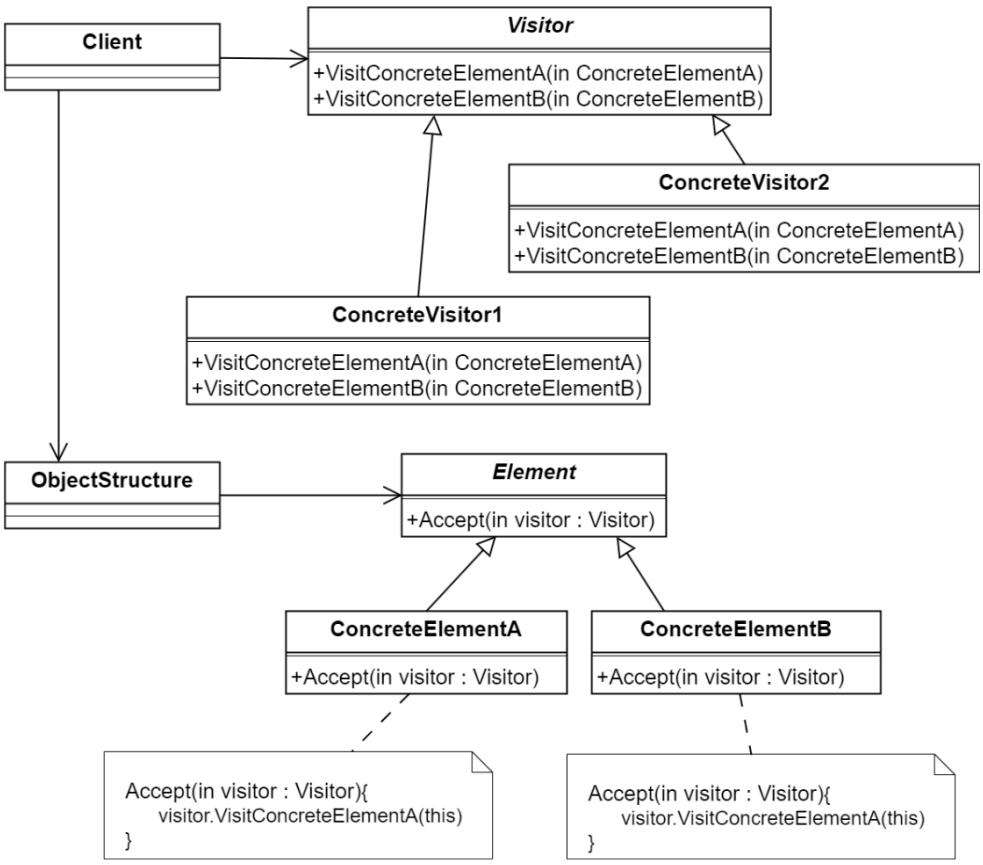
7. Яке призначення шаблону «Інтерпретатор»?

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.