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 «Теорія розробки прогрманого забезпечення»

Тема: «Патерни проектування.»

Варіант «Powershell terminal»

Виконав:

студент групи ІА-32
Бечке Олексій Ігорович

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

Зміст

Вступ

Тема: Патерни проектування.

Мета: Вивчити структуру шаблонів «Composite», «Flyweight» (Пристосуванець), «Interpreter», «Visitor» та навчитися застосовувати їх в реалізації програмної системи.

Тема проєкту: Powershell terminal (strategy, command, abstract factory, bridge, interpreter, client-server). Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна).

Теоретичні відомості

Шаблон «Interpreter»

Призначення: Даний шаблон використовується для подання граматики і інтерпретатора для вибраної мови (наприклад, скриптової) [6]. Граматика мови представлена термінальними і нетермінальними символами, кожен з яких інтерпретується в контексті використання. Клієнт передає контекст і сформовану пропозицію в використовувану мову в термінах абстрактного синтаксичного дерева (деревоподібна структура, яка однозначно визначає ієрархію виклику підвиразів), кожен вираз інтерпретується окремо з використанням контексту. У разі наявності дочірніх виразів, батьківський вираз інтерпретує спочатку дочірні (рекурсивно), а потім обчислює результат власної операції.

Шаблон зручно використовувати в разі невеликої граматики (інакше розростеться кількість використовуваних класів) і відносно простого контексту (без взаємних залежностей і т.п.).

Даний шаблон визначає базовий каркас інтерпретатора, який за допомогою рекурсії повертає результат обчислення пропозиції на основі результатів окремих елементів.

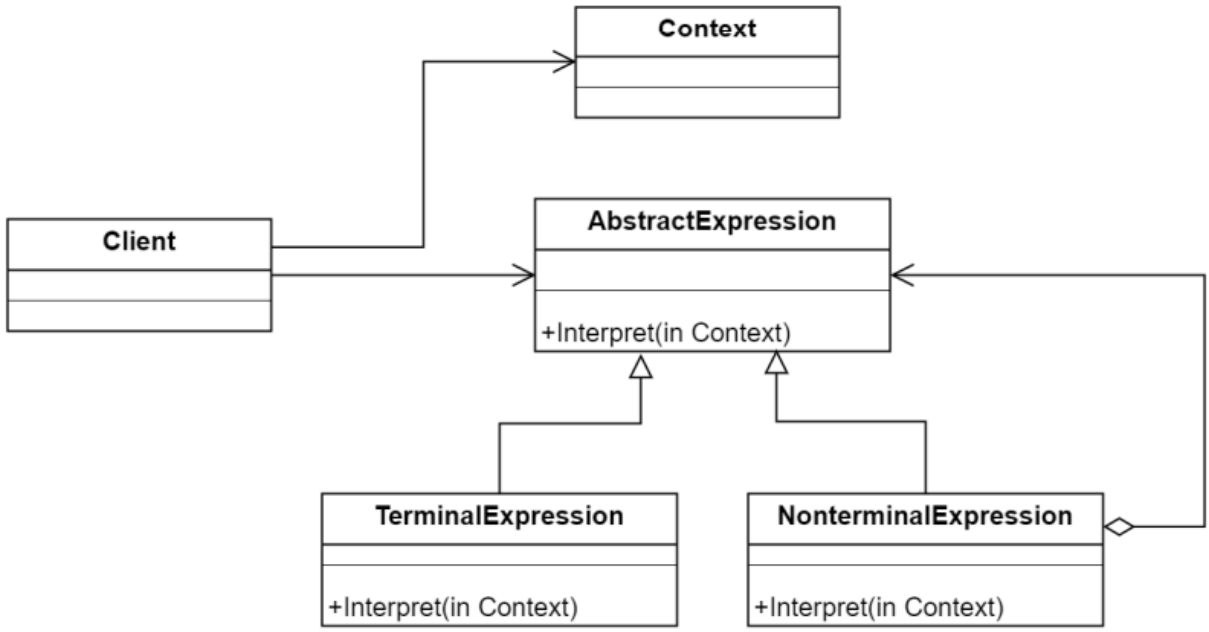


Рисунок 8.4. Структура патерна «Інтерпретатор»

При використанні даного шаблону дуже легко реалізовується і розширюється граматика, а також додаються нові способи інтерпретації виразів.

Проблема: Стоїть задача пошуку рядків по зразку, як така, що часто зустрічається. Або ж загалом є якась задача, яка дуже часто змінюється.

Рішення: Може бути вирішена шляхом створення інтерпретатора, який визначає граматику мови. «Клієнт» будує речення у вигляді абстрактного синтаксичного дерева, у вузлах якого знаходяться об’єкти класів «НетермінальнийВираз» і «ТермінальнийВираз» (рекурсивне), потім «Клієнт» ініціалізує контекст і виражає операцію Розібрати(Контекст). На кожному вузлі типу «НетермінальнийВираз» визначається операція Розібрати для кожного підвиразу. Для класу «ТермінальнийВираз» операція Розібрати визначає базу рекурсії. «АбстрактнийВираз» визначає абстрактну операцію Розібрати, загальну для всіх вузлів в абстрактному синтаксичному дереві. «Контекст» містить інформацію, глобальну по відношенню до інтерпретатора.

Переваги та недоліки:

+ Граматику стає легко розширювати та змінювати, реалізації класів, що описують вузли абстрактного синтаксичного дерева схожі (легко кодуються).

+ Можна легко змінювати спосіб обчислення виразів.

- Супроводження граматики с великою кількістю правил є проблематичним.

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

Ми використовуємо патерн Interpreter (Інтерпретатор) для відокремлення логіки аналізу та форматування тексту від основного процесу виконання команд. Замість того, щоб захаращувати код CommandInvoker або контролерів складними умовними конструкціями та жорстко закодованими перевірками рядків (наприклад, для виявлення помилок чи заголовків), ми визначаємо правила інтерпретації у вигляді окремих класів-виразів.

Абстракція (Expression) задає єдиний інтерфейс для аналізу контексту (рядка виводу), а конкретні термінальні вирази (ErrorExpression, HeaderExpression) реалізують специфічні правила розпізнавання шаблонів (наприклад, пошук ключових слів "Error" або "Exception"). Спеціальний клієнтський клас (SyntaxHighlighter) використовує ці вирази для перетворення звичайного тексту на HTML-розмітку з відповідними CSS-стилями. Це дозволяє легко розширювати систему новими правилами підсвічування синтаксису без необхідності змінювати логіку обробки історії команд.

```
/**
 * AbstractExpression: Оголошує інтерфейс для виконання операції інтерпретації.
 */
public interface Expression {  5 usages  2 implementations  new *
    boolean interpret(String context);  1 usage  2 implementations  new *

    String getStyle();  1 usage  2 implementations  new *
}
```

Рис. 1 – Код інтерфейсу Expression

```java
/**
 * TerminalExpression: Реалізує інтерпретацію для помилок.
 */
@Component  new *
public class ErrorExpression implements Expression {

    @Override  1 usage  new *
    public boolean interpret(String context) {
        if (context == null) return false;
        return context.trim().startsWith("Error") ||
                context.contains("Exception") ||
                context.startsWith("[Error]");
    }

    @Override  1 usage  new *
    public String getStyle() {
        return "color: red; font-weight: bold;";
    }
}
```

Рис. 2 – Код класу ErrorExpression

```java
/**
 * TerminalExpression: Реалізує інтерпретацію для заголовків або успішних даних.
 */
@Component  new *
public class HeaderExpression implements Expression {

    @Override  1 usage  new *
    public boolean interpret(String context) {
        if (context == null) return false;
        // Правило: типові заголовки PowerShell таблиць або статус OK
        return (context.contains("Mode") && context.contains("LastWriteTime")) ||
                context.contains("System status: OK");
    }

    @Override  1 usage  new *
    public String getStyle() {
        return "color: #f1c40f; font-weight: bold;";
    }
}
```

Рис. 3 – Код класу HeaderExpression

Зображення структури шаблону

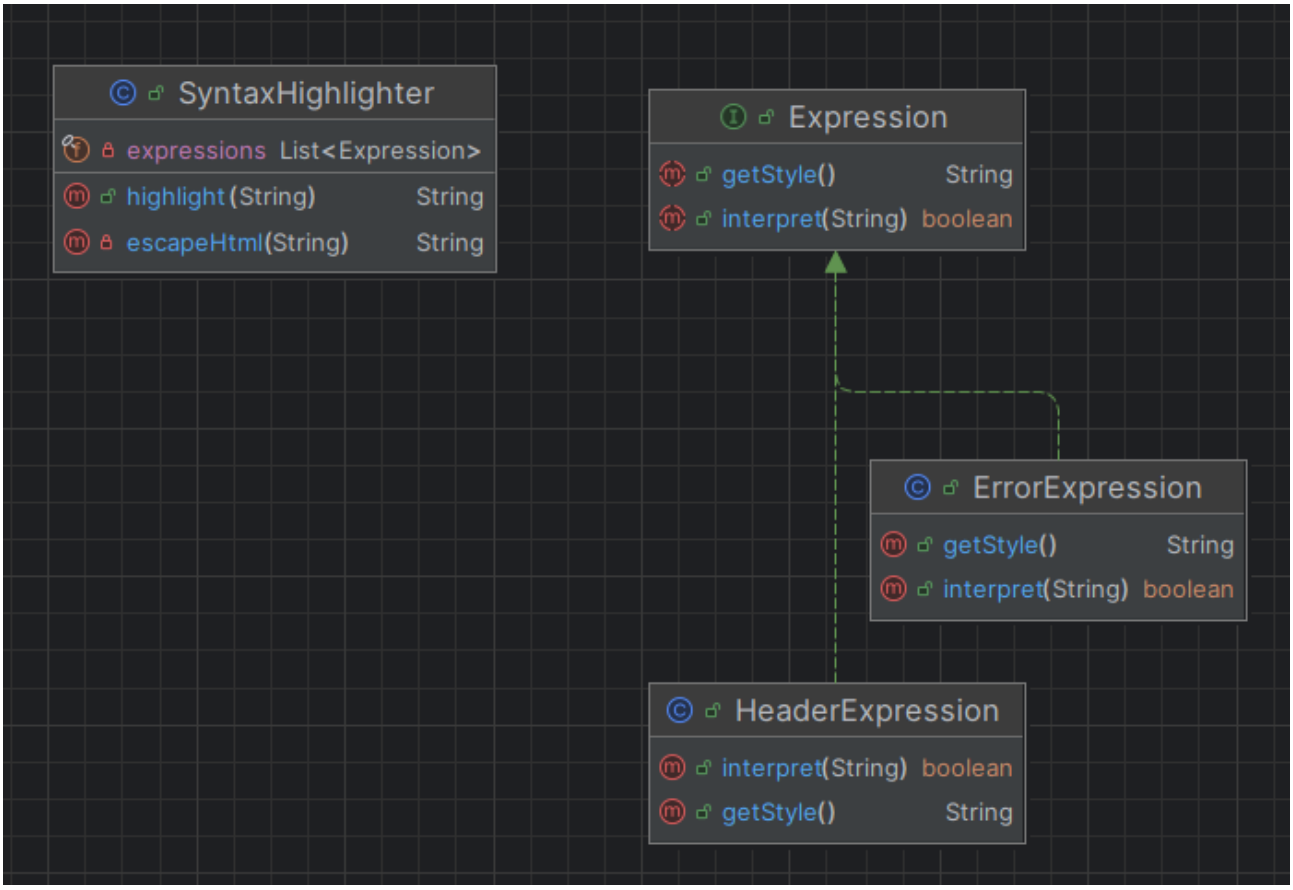


Рис. 4 – Структура шаблону Interpreter

Структура реалізації:

- Expression (AbstractExpression) – інтерфейс, що визначає єдиний контракт для всіх правил аналізу тексту. Він оголошує метод interpret(String context), який перевіряє, чи відповідає вхідний рядок певному шаблону, та метод getStyle(), що повертає відповідний стиль оформлення (CSS). Цей інтерфейс слугує базою для побудови граматики підсвічування.
- ErrorExpression / HeaderExpression (Terminal Expressions) – конкретні реалізації виразів, що визначають правила для ідентифікації окремих елементів виводу. ErrorExpression розпізнає повідомлення про помилки (наприклад, за ключовими словами "Error" або "Exception"), а HeaderExpression ідентифікує заголовки таблиць даних. Кожен з них повертає специфічний CSS-стиль (червоний для помилок, жовтий для заголовків), якщо правило справджується.
- SyntaxHighlighter (Client / Context Handler) – клас, який виступає в ролі клієнта інтерпретатора. Він зберігає колекцію всіх доступних виразів (Expression) і послідовно застосовує їх до вхідного тексту (контексту). Його завдання – просканувати рядок, знайти відповідне правило та перетворити звичайний текст на форматований HTML-код зі стилями, готовий до відображення у браузері.

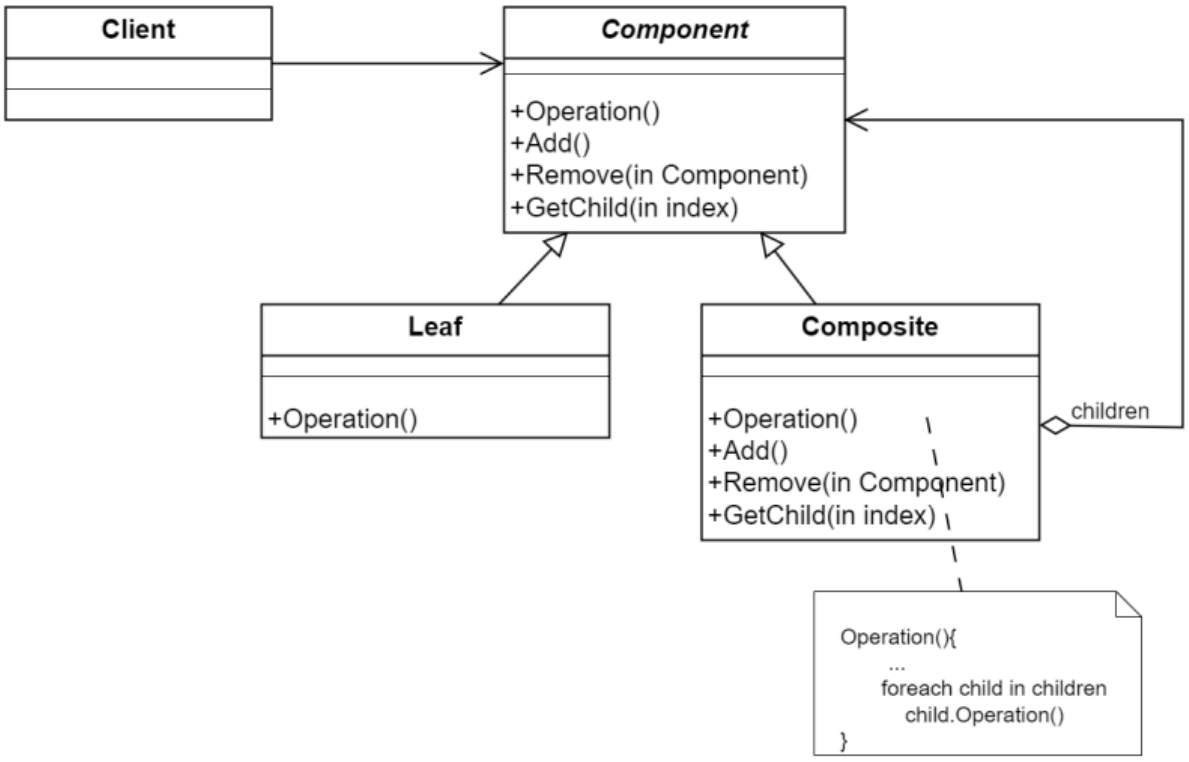
Такий підхід дозволяє відокремити логіку візуалізації та аналізу тексту від бізнес-логіки виконання команд. Це забезпечує гнучкість системи: можна легко додавати нові правила підсвічування (нові Terminal Expressions) для різних типів повідомлень або даних без необхідності вносити зміни в основний код контролерів чи механізмів виконання скриптів.

Питання до лабораторної роботи

1. Яке призначення шаблону «Композит»?

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.

2. Нарисуйте структуру шаблону «Композит».



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

Component (Компонент): Це абстракція (інтерфейс або абстрактний клас), яка оголошує операції, спільні як для простих, так і для складних об'єктів композиції. Вона дозволяє клієнту поводитися з усіма елементами однаково.

Leaf (Лист): Представляє кінцеві об'єкти (листки) структури, які не мають дочірніх елементів. Цей клас реалізує базову поведінку, визначену в компоненті.

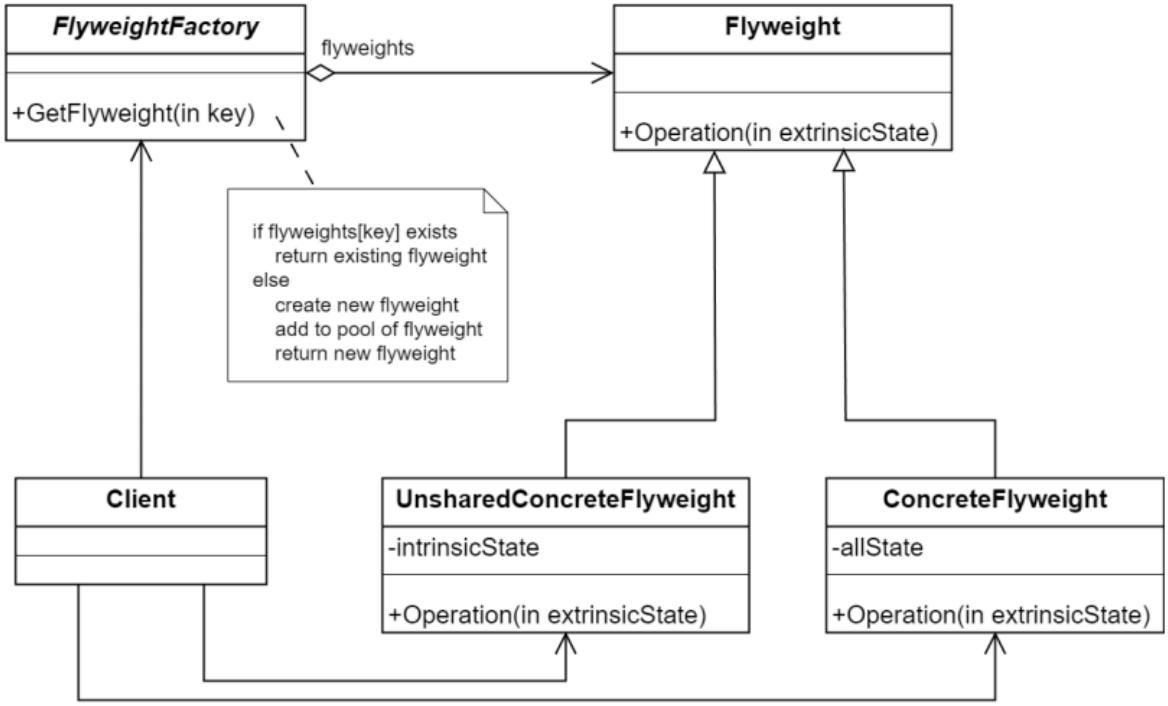
Composite (Композит): Клас, який містить дочірні елементи (які можуть бути як листками, так і іншими композитами) і реалізує поведінку компонентів, що мають нащадків. Він зберігає колекцію своїх дочірніх компонентів (children).

Client (Клієнт): Взаємодіє з об'єктами структури через інтерфейс Component. Завдяки цьому клієнт може працювати як з окремими об'єктами, так і з цілими ієрархіями, не знаючи конкретних класів об'єктів.

4. Яке призначення шаблону «Легковаговик»?

Шаблон використовується для зменшення кількості об'єктів в додатку шляхом поділу цих об'єктів між ділянками додатку. Flyweight являє собою поділюваний об'єкт.

5. Нарисуйте структуру шаблону «Легковаговик».



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

Flyweight (Легковаговик): Інтерфейс або абстрактний клас, який оголошує операції, що приймають зовнішній стан як аргумент (наприклад, Operation(in extrinsicState)). Це дозволяє об'єктам використовувати зовнішній стан, не зберігаючи його всередині.

ConcreteFlyweight (Конкретний легковаговик): Реалізує інтерфейс Flyweight і зберігає внутрішній стан (intrinsicState). Цей стан має бути незалежним від контексту об'єкта, оскільки об'єкт цього класу поділяється (використовується спільно) .

UnsharedConcreteFlyweight (Неподілюваний конкретний легковаговик): Не всі підкласи Flyweight обов'язково мають бути розділюваними. Цей клас представляє об'єкти, які не поділяються, але все одно реалізують інтерфейс Flyweight (часто використовуються для композиції легковаговиків) .

FlyweightFactory (Фабрика легковаговиків): Створює та керує пулом легковаговиків. Вона гарантує, що легковаговики використовуються коректно: коли клієнт запитує легковаговика, фабрика або повертає існуючий екземпляр, або створює новий, якщо такого ще немає .

Client (Клієнт): Зберігає посилання на легковаговиків і обчислює або зберігає їхній зовнішній стан (extrinsicState).

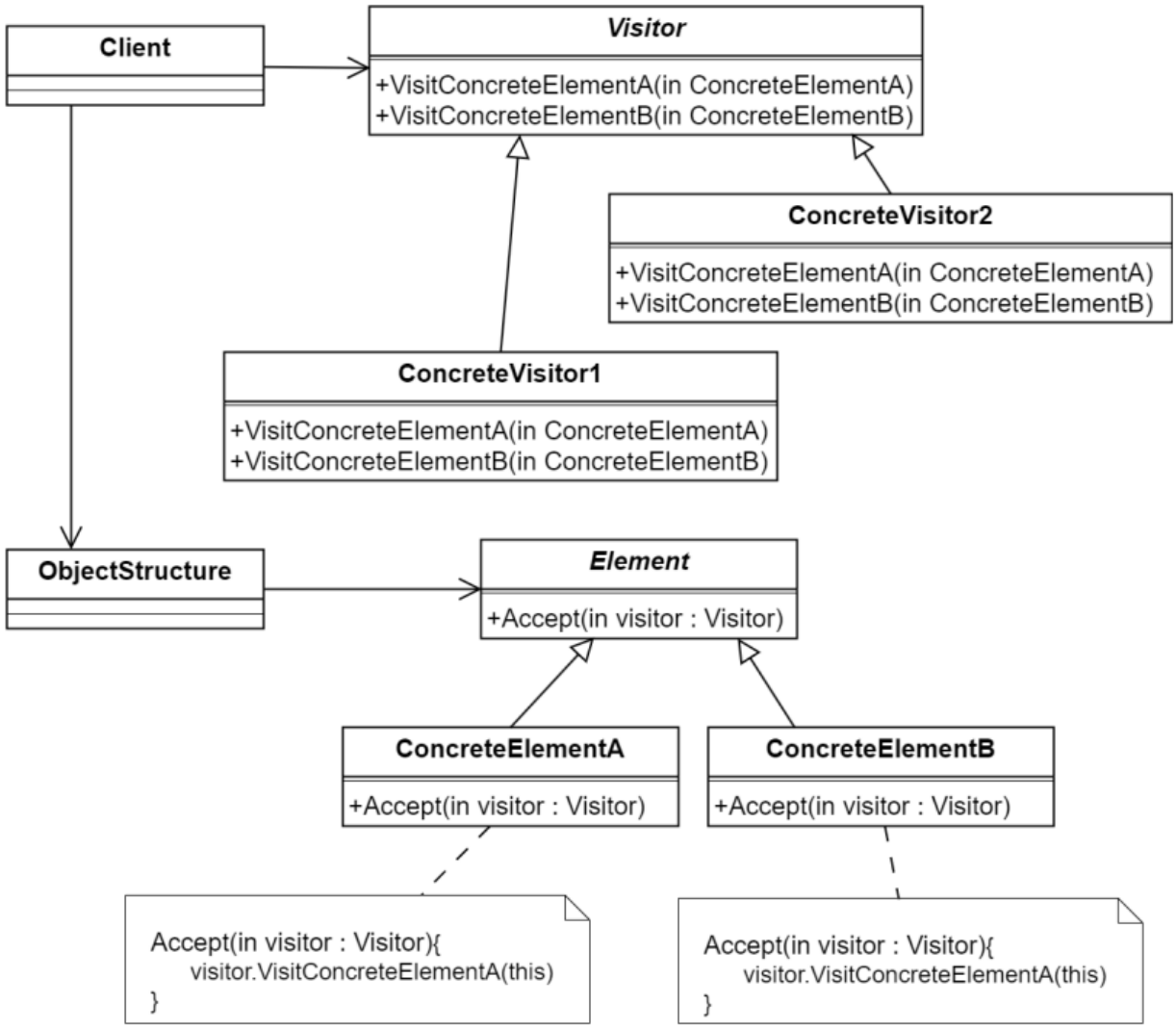
7. Яке призначення шаблону «Інтерпретатор»?

Даний шаблон використовується для подання граматики і інтерпретатора для вибраної мови (наприклад, скриптової). Граматика мови представлена термінальними і нетермінальними символами, кожен з яких інтерпретується в контексті використання. Клієнт передає контекст і сформовану пропозицію в використовувану мову в термінах абстрактного синтаксичного дерева (деревоподібна структура, яка однозначно визначає ієрархію виклику підвиразів), кожен вираз інтерпретується окремо з використанням контексту. У разі наявності дочірніх виразів, батьківський вираз інтерпретує спочатку дочірні (рекурсивно), а потім обчислює результат власної операції.

8. Яке призначення шаблону «Відвідувач»?

Шаблон відвідувач дозволяє вказувати операції над елементами без зміни структури конкретних елементів. Таким чином вкрай зручно додавати нові операції, проте дуже важко додавати нові елементи в ієрархію (необхідно додавати відповідні методи для обробки їх відвідувань в кожного відвідувача).

9. Нарисуйте структуру шаблону «Відвідувач».



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

Visitor (Відвідувач): Інтерфейс або абстрактний клас, який оголошує методи відвідування для кожного класу конкретного елемента (наприклад, VisitConcreteElementA, VisitConcreteElementB). Сигнатура методу дозволяє відвідувачу визначити конкретний клас елемента, з яким він працює .

ConcreteVisitor (Конкретний відвідувач): Реалізує методи інтерфейсу Visitor. Кожен метод реалізує фрагмент алгоритму, призначений для відповідного класу елемента. Цей клас також може зберігати локальний стан алгоритму .

Element (Елемент): Оголошує метод прийняття відвідувача (зазвичай Accept(visitor: Visitor)), який приймає об'єкт відвідувача як аргумент .

ConcreteElement (Конкретний елемент): Реалізує метод Accept. У цьому методі елемент викликає відповідний метод відвідувача, передаючи себе як аргумент (наприклад, visitor.VisitConcreteElementA(this)). Це ключовий момент патерну (подвійна диспетчеризація) .

ObjectStructure (Структура об'єктів): Клас (або колекція), що містить набір елементів. Він може надавати інтерфейс високого рівня, що дозволяє відвідувачу відвідати кожен елемент у структурі.

Client (Клієнт): Ініціює процес, створюючи об'єкт конкретного відвідувача та передаючи його в структуру об'єктів (або викликаючи Accept на елементах).

Висновок

Під час виконання лабораторної роботи я вивчив структуру шаблонів «Composite», «Flyweight» (Пристосуванець), «Interpreter», «Visitor» та навчитися застосовувати їх в реалізації програмної системи. Реалізував шаблон «Interpreter» у проєкті Power Shell термінал.