Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №8
Технології розроблення програмного забезпечення
«Патерни проєктування»

Виконав
студент групи ІА–34:
Бородай А.С.

Київ 2025

Зміст

Теоретичні відомості 3
Шаблон «Composite» 3
Шаблон «Flyweight» 4
Шаблон «Interpreter» 4
Шаблон «Visitor» 5
Хід роботи 8
ClassDiagram (Composite) 8
Висновок 9

Теоретичні відомості

Шаблон «Composite»

Призначення патерну «Composite»: 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.

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.

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-аналітиці, 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.

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.

Шаблон «Flyweight»

Призначення «Flyweight»: використовується для зменшення кількості об'єктів в додатку шляхом поділу цих об'єктів між ділянками додатку. Flyweight являє собою поділюваний об'єкт. Дуже важливою є концепція «внутрішнього» і «зовнішнього» станів.

Внутрішній стан відображає дані, характерні саме поділюваному об'єкту (наприклад, код букви); зовнішній стан несе інформацію про його застосування в додатку (наприклад, рядок і стовпчик). Внутрішній стан зберігається в самому поділюваному об'єкті, зовнішній – в об'єктах додатку (контексту використання поділюваного об'єкта).

Переваги та недоліки:

+ Заощаджує оперативну пам’ять.

- Витрачає процесорний час на пошук/обчислення контексту.

- Ускладнює код програми внаслідок введення безлічі додаткових класів.

Шаблон «Interpreter»

Призначення «Interpreter»: 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.

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.

Рішення: Може бути вирішення 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.

Шаблон «Visitor»

Призначення «Visitor»: 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). Даний шаблон дозволяє групувати однотипні операції, що застосовуються над різнотипними об'єктами.

Проблема: Ви розробляєте онлайн-корзину інтернет магазина. Товари які представлені в магазині є різних типів, наприклад, електроніка, міцні напої, домашня хімія.

Логіка роботи з товарами в корзині є різна, наприклад, розрахунок вартості, формування замовлення.

Ми можемо всі ці методи зробити в товарах, але тоді ми ускладнюємо товари і змішуємо логіку (розрахунок вартості) з даними (товаром та його кількістю).

Якщо ми в подальшому необхідно буде добавити ще логіку розрахунку вартості з врахуванням знижки, то потрібно буде добавити ще цю логіку до товарів. А якщо буде ще сезонна знижка, то ми знову будемо добавляти нову логіку до класів товарів.

Рішення: Основна ідея патерна «Відвідувач» це рознести логіку і дані в різні класи та ієрархії. Якщо так зробити для нашої онлайн-корзини, то в класі відвідувача ми маємо функції для обрахунку логіки для кожного типу, а об'єкти в корзині знають свій тип, отримують екземпляр відвідувача і в нього викликають метод відповідно до свого типу. В результаті ми робимо різні відвідувачі для розрахунку вартості: один для звичайних розрахунків, інший для розрахунку вартості зі знижкою, ще один для розрахунку сезонних знижок.

За рахунок того, що логіка відокремлена від наших товарів в корзині ми можемо реалізовувати за необхідності нові класи відвідувачі, а класи товарів та і корзини, в цілому, змінюватися не будуть.

Якщо розвивати далі, то можна зробити і клас відвідувача, який буде формувати замовлення з товарів в корзині в залежності від продавців і можливих варіантів доставки. Це дозволить формувати, наприклад, не одне замовлення, а два одна з самовивозом з магазину, а інше з доставкою Новою Поштою

Тема: Патерни проектування.

Мета: Вивчити структуру шаблонів «Composite», «Flyweight», «Interpreter», «Visitor» 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.

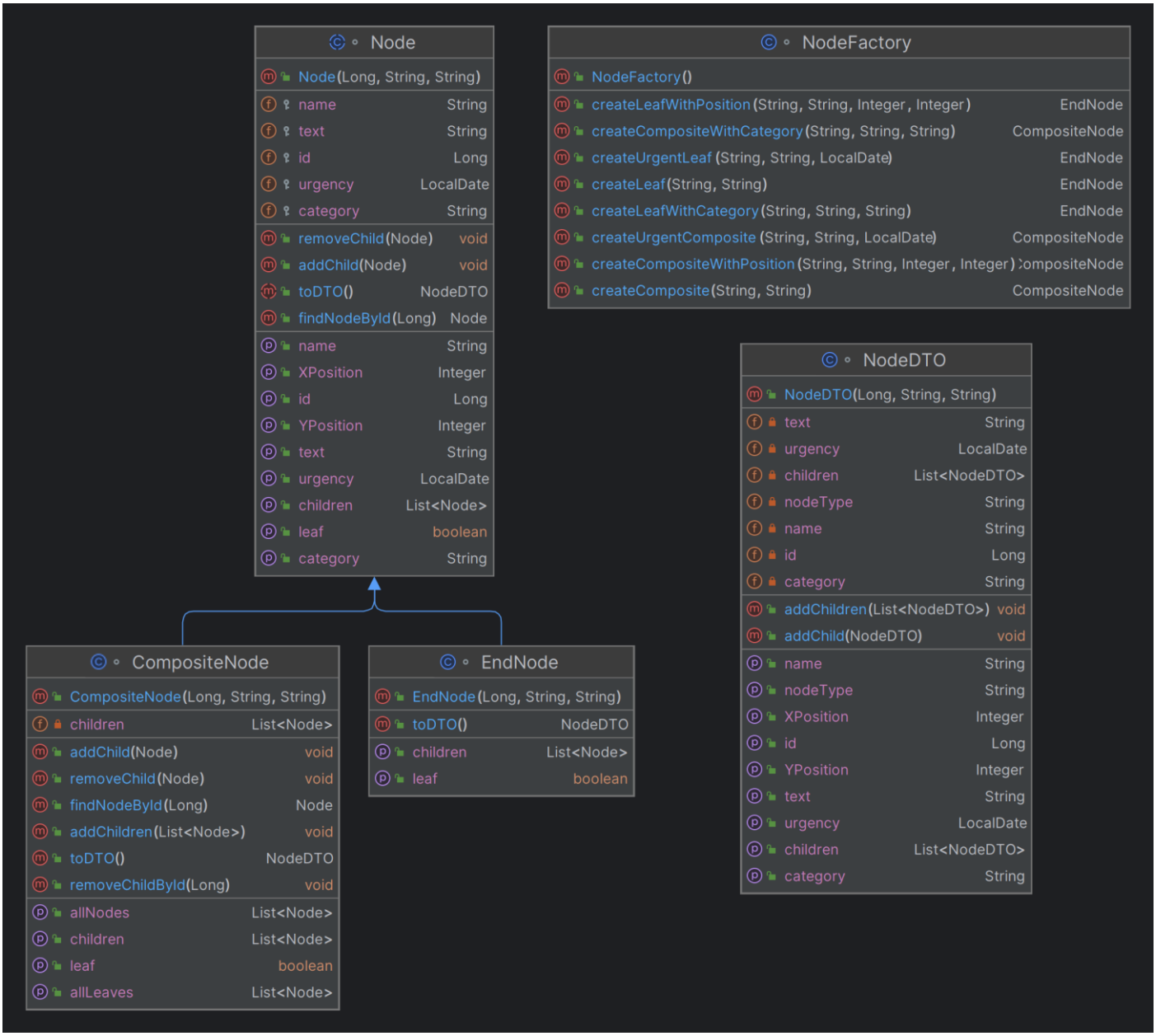
Хід роботи

20. **Mind-mapping software**

(strategy, prototype, abstract factory, bridge, composite, SOA)

Візуальний додаток для складання "карт пам'яті" з можливістю роботи з декількома картами (у вкладках), автоматичного промальовування ліній, додавання вкладених файлів, картинок, відеофайлів (попередній перегляд); можливість додавання значків категорій / терміновості, обведення областей карти (поділ пунктирною лінією).

ClassDiagram (Composite)



Node (Component) – абстрактний клас для всіх об'єктів поміток у ментальній карті. Він має інформацію, яку треба виводити користувачу, та оголошує операцію, toDTO() для її передачі. Також клас містить методи для додавання, видалення та доступу до дочірніх компонентів.

EndNode (Leaf) – кінцевий об'єкт помітки на мапі. Він не має дочірніх компонентів і реалізує тільки операцію toDTO().

CompositeNode (Composite) – містить інші помітки (EndNode або інші CompositeNode). Зберігає список дочірніх об'єктів. Реалізує операцію toDTO(), яка делегує виконання всім своїм дочірнім компонентам. Також реалізує методи для додавання, видалення та доступу до дітей.

Додаткові компоненти:

NodeFactory – клас, який відповідає за створення об'єктів деревоподібної структури, приховуючи складність їх конструкції та забезпечуючи узгодженість. Надає зручні методи для створення різних типів вузлів з різними параметрами.

NodeDTO – об’єкт, який виступає як контейнер для передачі даних про помітки.

Висновок

Під час виконання лабораторної роботи я ознайомився із шаблонами «Composite», «Flyweight», «Interpreter», «Visitor». Мною були розроблені основні класи, які реалізують шаблон Composite та його поведінку.

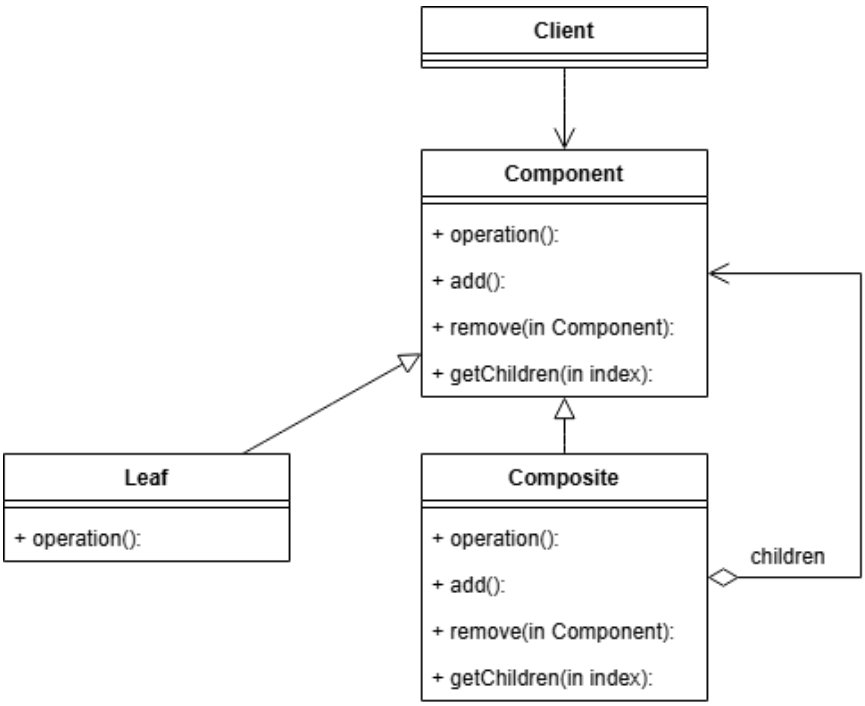
Для майбутнього відображення карти, було реалізовано функціонал виведення стану (поточної інформації) поміток (всіх, поточних чи гілки) на мапі та збереження зв’язків між помітками, що будуть відображатися користувачу на екран.

Відповіді на питання для самоперевірки

1. Яке призначення шаблону «Композит»?

Шаблон дозволяє згрупувати об'єкти в деревоподібні структури для представлення ієрархій "частина-ціле". Клієнт може єдиним чином працювати як з окремими об'єктами, так і з їх групами, ігноруючи різницю між ними.

2. Нарисуйте структуру шаблону «Композит».



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

Component – інтерфейс або абстрактний клас для всіх об'єктів у композиції. Він оголошує спільні операції, такі як operation(), а також методи для додавання, видалення та доступу до дочірніх компонентів (які можуть бути реалізовані за замовчуванням, щоб викидати виняток у Leaf).

Leaf – представляє кінцевий об'єкт у дереві. Він не має дочірніх компонентів. Реалізує операцію operation() для листка. Методи для роботи з дітьми зазвичай нічого не роблять або викидають виняток.

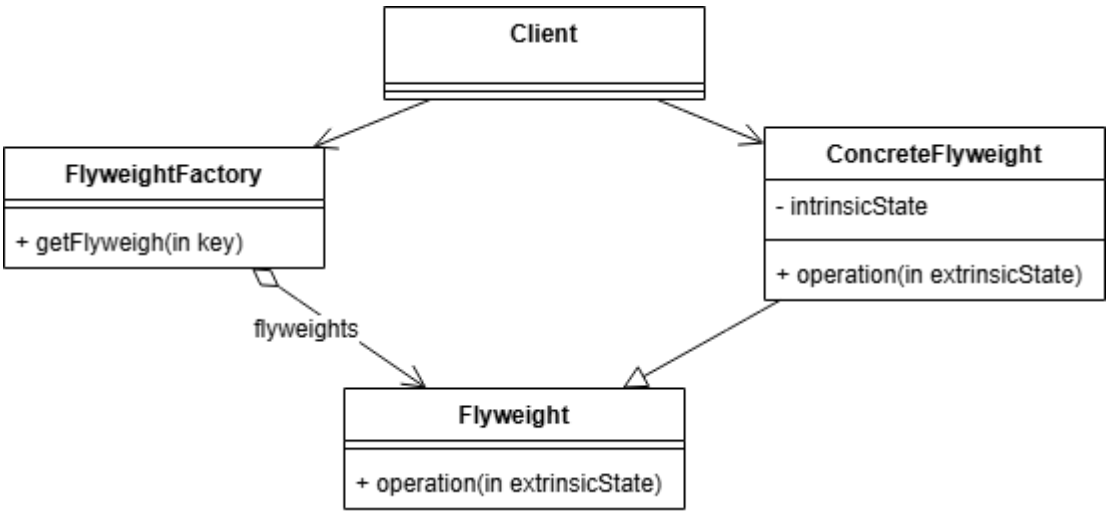
Composite – містить інші компоненти (як Leaf, так і інші Composite). Зберігає список дочірніх об'єктів. Він реалізує операцію operation(), яка зазвичай делегує виконання всім своїм дочірнім компонентам. Також реалізує методи для додавання, видалення та доступу до дітей.

Клієнт працює з Component, не знаючи, це Leaf чи Composite.

4. Яке призначення шаблону «Легковаговик»?

Шаблон дозволяє ефективно підтримувати велику кількість дрібних об'єктів, розділяючи спільний стан між ними. Він зменшує використання пам'яті, уникаючи дублювання однакових даних.

5. Нарисуйте структуру шаблону «Легковаговик».



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

Flyweight – інтерфейс, через який легковаговики можуть отримувати зовнішній стан та працювати з ним. Визначає метод operation(extrinsicState).

ConcreteFlyweight – реалізує інтерфейс Flyweight і зберігає внутрішній стан (intrinsic state) – дані, які є спільними та незмінними. Цей стан не залежить від контексту

FlyweightFactory – створює та керує легковаговиками. Забезпечує коректне розподілення легковаговиків. Коли клієнт запитує легковаговик, фабрика або повертає існуючий, або створює новий, якщо такого ще немає.

Клієнт обчислює або зберігає зовнішній стан і передає його легковаговику при виклику його методів. При виклику operation(extrinsicState) легковаговик використовує внутрішній стан і переданий зовнішній стан.

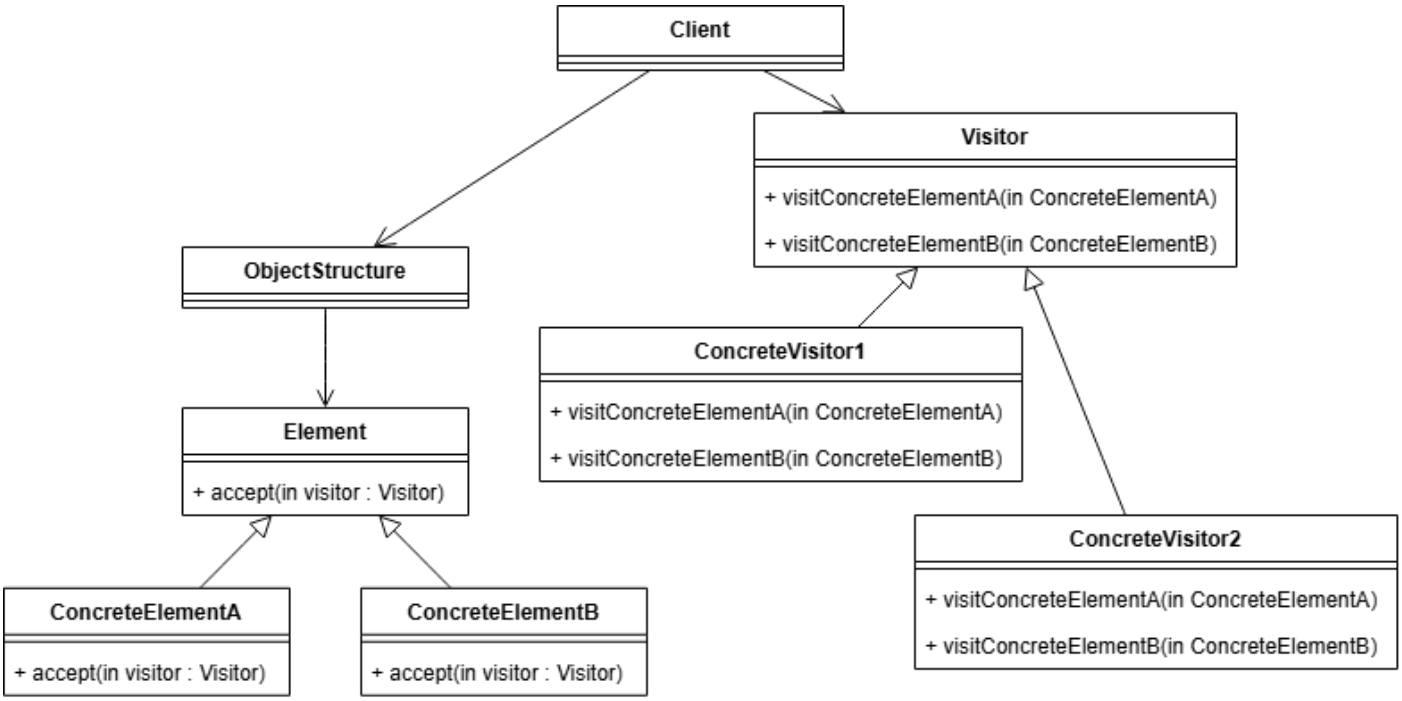
7. Яке призначення шаблону «Інтерпретатор»?

Шаблон визначає граматику для заданої мови та інтерпретує її речення. Він використовується для написання інтерпретаторів, компіляторів, парсерів тощо.

8. Яке призначення шаблону «Відвідувач»?

Шаблон дозволяє додавати нові операції до ієрархії класів, не змінюючи самі ці класи. Він розділяє алгоритм від структури об'єктів, на яких він працює. 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 – інтерфейс, який оголошує методи visitConcreteElement, для кожного конкретного класу елементів. Кожен метод приймає посилання на відповідний елемент, що дозволяє відвідувачу отримувати доступ до його стану.

ConcreteVisitor – реалізує кожну операцію, оголошену у Visitor. Кожна операція це частина загального алгоритму, який працює зі структурою. Він містить контекст та зберігає проміжний стан (якщо потрібно). При виклику element.accept(visitor) саме цей об'єкт буде передано в елемент.

Element – базовий інтерфейс елементів, який оголошує метод accept(visitor: Visitor). Цей метод є єдиним, який потрібно реалізувати в елементах. Він зазвичай просто викликає visitor.visitThisElement(this).

ConcreteElement – конкретний клас елемента, який реалізує метод accept(visitor). Він викликає специфічний для свого класу метод відвідувача.

ObjectStructure – набір елементів (список, дерево чи ін.). Може надавати метод для "відвідування" себе, запускаючи accept для кожного елемента.

Клієнт передає відвідувача елементам структури через метод accept(). Кожен елемент викликає у відвідувача специфічний метод visitElement(), передаючи себе як аргумент. Відвідувач виконує операцію з елементом, маючи доступ до його стану. Ця подвійна диспетчеризація дозволяє відвідувачу коректно обробляти різні типи елементів без змін їхнього коду.