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»

Виконала:
студентка групи ІА-33
Самойленко Анастасія

Перевірив:
асистент кафедри ІСТ
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 роботи:
27. Особиста бухгалтерія (state, prototype, decorator, bridge, flyweight, SOA)
Програма повинна бути наочним засобом для ведення особистих фінансів: витрат і прибутку; з можливістю встановлення періодичних витрат / прибутку (зарплата і орендна плата); введення сканованих чеків з відповідними статтями витрат; побудова статистики; експорт/імпорт в Excel, реляційні джерела даних; різні рахунки; ведення єдиного фонду на всі рахунки (всією сім'єю) – на особливі потреби (ремонт, автомобіль, відпустка); можливість введення вкладів / кредитів для контролю банківських рахунків (звірка нарахованих відсотків з необхідними і т.д.).

Вступ

Сучасне програмне забезпечення відрізняється високим рівнем складності та великою кількістю взаємопов’язаних компонентів. Для підвищення ефективності розробки, спрощення підтримки та повторного використання коду широко застосовуються патерни проектування. Патерни проектування – це перевірені практикою рішення типових задач організації програмних систем, які дозволяють структурувати код, робити його гнучким і масштабованим.

Метою цієї лабораторної роботи є вивчення структури та принципів роботи шаблонів «Composite», «Flyweight» (Пристосуванець), «Interpreter» та «Visitor», а також набуття навичок їх практичного застосування при реалізації програмної системи. Застосування цих патернів дозволяє ефективно управляти складними ієрархіями об’єктів, оптимізувати використання пам’яті, реалізовувати інтерпретацію мов і додавати нові операції над об’єктами без зміни їх структури.

У ході лабораторної роботи передбачається ознайомлення з принципами кожного з зазначених патернів, аналіз їх структури, розробка невеликих прикладів реалізації та інтеграція отриманих знань у практичну частину програмної системи.

Теоретичні відомості

Шаблон (патерн) проєктування — це формалізований опис типового рішення, яке багаторазово зустрічається під час розроблення інформаційних систем. Він містить узагальнений спосіб вирішення певної задачі проєктування, рекомендації щодо його застосування та має загальновживану назву.

Головна мета використання шаблонів — повторне застосування вже знайдених ефективних рішень у нових ситуаціях. Важливою умовою є коректне моделювання предметної області, що дозволяє правильно визначити задачу та обрати відповідний патерн.

Застосування шаблонів проєктування забезпечує низку переваг:

- створення структурованої та логічної моделі системи;
- підвищення наочності та спрощення вивчення архітектури;
- глибше опрацювання структури програмного продукту;
- підвищення стійкості системи до змін вимог;
- спрощення подальшого доопрацювання та розширення;
- полегшення інтеграції систем і взаєморозуміння між розробниками завдяки спільному “словнику проєктування”.

Шаблон «Composite»

Шаблон «Composite» дозволяє створювати деревоподібні структури об’єктів для представлення ієрархій «частина–ціле» та уніфіковано обробляти як прості об’єкти, так і складені з вкладених елементів. Наприклад, форма може містити поля введення, написи, малюнки, які, у свою чергу, можуть містити інші елементи. Операції над формою, як-от розтягування, застосовуються рекурсивно до всіх дочірніх елементів.

Проблема:

При розробці системи керування проєктами потрібно відображати оціночну вартість робіт по функціях, userstory та задачах і контролювати, що всі елементи були оцінені.

Рішення:

Використовується патерн «Composite». Класи Feature, Userstory та Task

наслідуються від інтерфейсу ITask. Feature та Userstory — складені об’єкти з колекціями ITask, а Task — кінцеві елементи без дочірніх об’єктів. Метод GetEstimatedPoints() реалізується рекурсивно для об’єктів-компоновщиків, сумуючи оцінки дочірніх елементів. Клієнтський код працює з ITask незалежно від типу об’єкта.

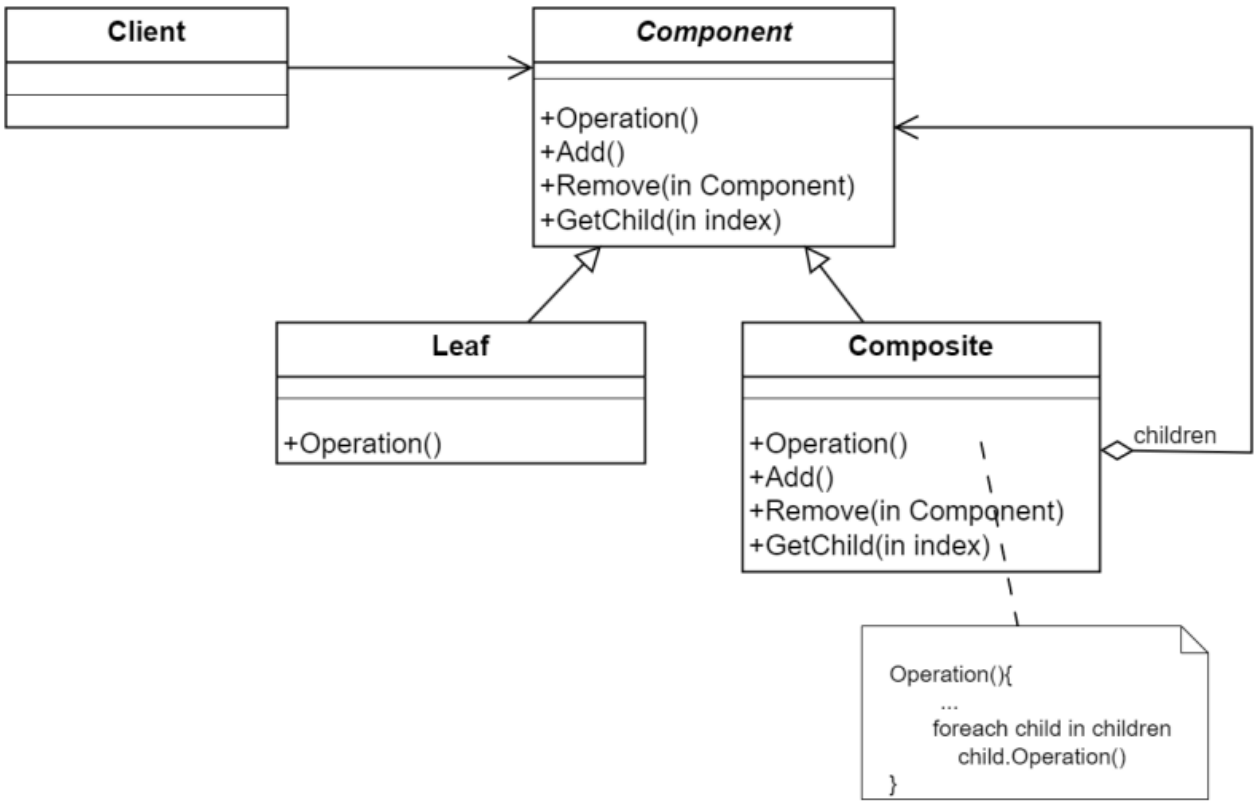
Переваги:

- Спрощує представлення деревоподібних структур.
- Додає гнучкості при роботі з складними об’єктами та рекурсивними операціями.
- Дозволяє додавати або видаляти елементи без зміни клієнтського коду.

Недоліки:

- Потребує додаткових зусиль для початкової реалізації.
- Вимагає продуманого загального інтерфейсу.

Структура патерну Composite на рівні об’єктів:



Шаблон «Flyweight»

Патерн «Flyweight» використовується для зменшення кількості об’єктів у додатку шляхом їх поділу між різними ділянками програми. Він дозволяє

зберігати один поділюваний об’єкт і посилатися на нього з багатьох місць.

Ключовою є концепція внутрішнього та зовнішнього станів:

- Внутрішній стан зберігає дані, характерні саме для об’єкта (наприклад, код букви).
- Зовнішній стан відображає контекст його використання (наприклад, рядок і стовпчик) і зберігається поза об’єктом.

Приклад:

Для відображення тексту фізично створюється лише один об’єкт для кожної літери, а всі позиції літери на екрані використовують посилання на цей об’єкт. Це економить пам’ять при великій кількості однакових об’єктів, наприклад, графічних примітивів або символів тексту.

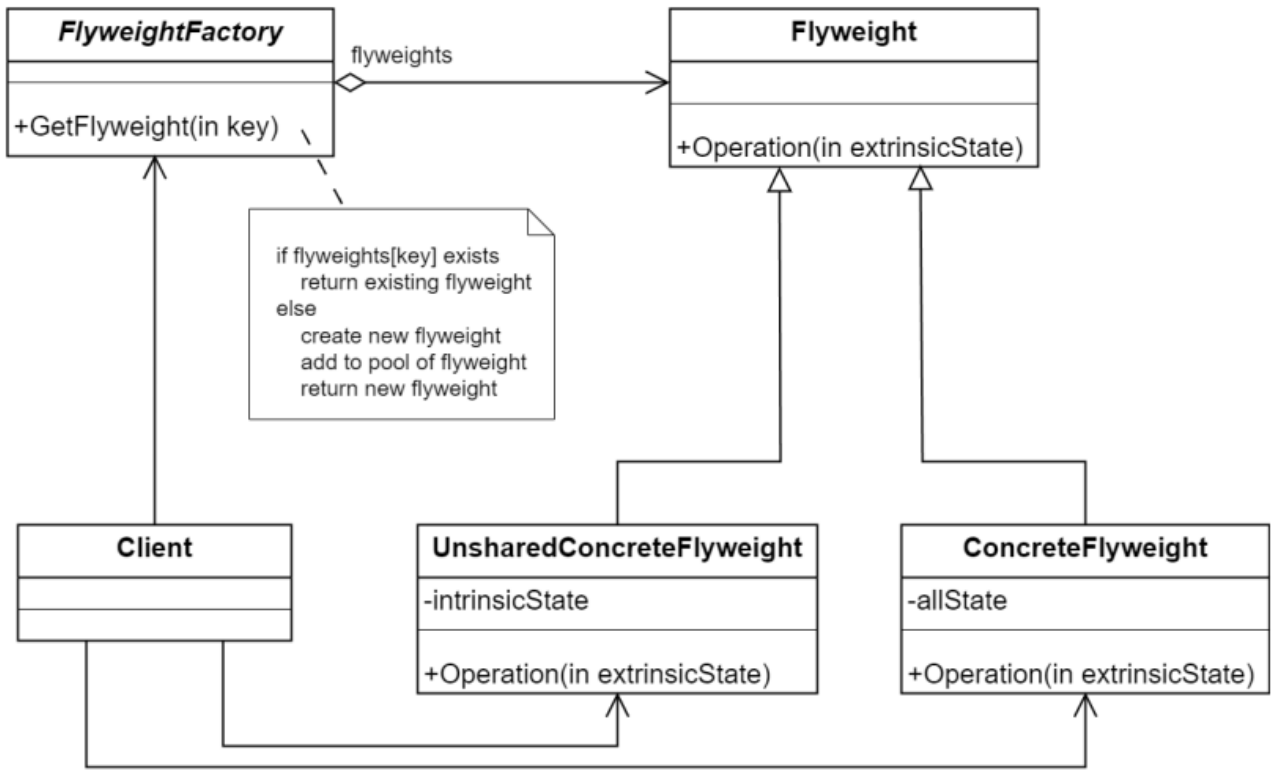
Переваги:

- Економить оперативну пам’ять.

Недоліки:

- Додаткові витрати процесорного часу на управління контекстом.
- Ускладнює код через введення додаткових класів і логіки поділу станів.

Структура патерну Легковаговик:



Шаблон «Interpreter»

Патерн «Interpreter» використовується для представлення граматики мови (наприклад, скриптової) та створення інтерпретатора для її виразів. Граматика визначається термінальними та нетермінальними символами, які інтерпретуються у заданому контексті. Клієнт передає контекст і вираз у вигляді абстрактного синтаксичного дерева (AST), де кожен вузол обчислюється рекурсивно: спочатку обробляються дочірні вирази, потім виконується операція батьківського вузла.

Проблема:

Необхідно реалізувати часто змінювану функціональність, наприклад, пошук рядків за зразком, або інші задачі, що часто змінюються.

Рішення:

Створюється інтерпретатор, який визначає граматику мови. Клієнт формує речення як AST з вузлів класів НетермінальнийВираз і ТермінальнийВираз.

Метод Розібрати(Контекст) рекурсивно обчислює результат:

- НетермінальнийВираз виконує розбір своїх підвиразів.
- ТермінальнийВираз визначає базу рекурсії.
- АбстрактнийВираз задає загальний інтерфейс для всіх вузлів.
- Контекст містить глобальні дані для інтерпретатора.

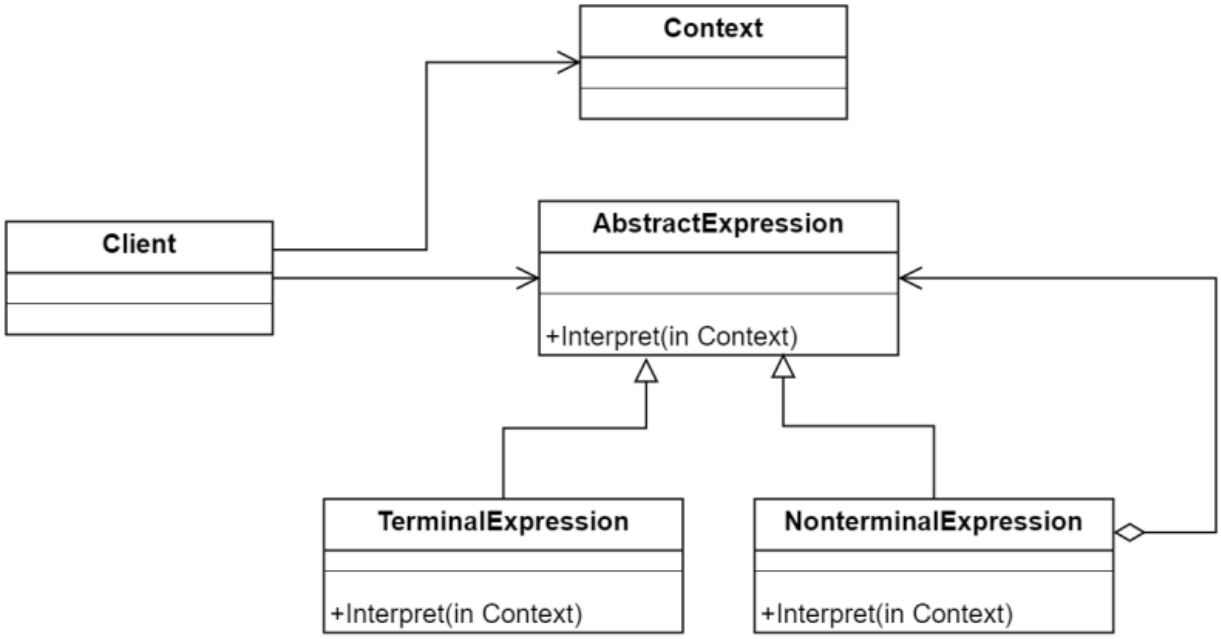
Переваги:

- Граматика легко розширюється та змінюється.
- Реалізації класів вузлів AST схожі, що спрощує кодування.
- Легко змінювати спосіб обчислення виразів.

Недоліки:

- Ускладнюється супровід граматики з великою кількістю правил.

Структура шаблону Інтерпретатор:



Шаблон «Visitor»

Патерн «Visitor» дозволяє визначати операції над елементами об’єктної структури без зміни їх класів. Це зручно для додавання нових операцій, проте ускладнює додавання нових типів елементів, оскільки потрібно оновлювати всіх відвідувачів.

Проблема:

В онлайн-корзині інтернет-магазину товари різних типів (електроніка, напої, хімія) потребують різної логіки обчислення вартості, формування замовлення та застосування знижок. Якщо реалізовувати всю логіку у класах товарів, це змішуватиме дані та бізнес-логіку, ускладнюватиме супровід та розширення.

Рішення:

Логіку відокремлюють від даних: створюють клас відвідувача з методами для обробки кожного типу товару. Об’єкти в корзині викликають відповідний метод відвідувача залежно від свого типу. Можна створювати різних відвідувачів для різних операцій: розрахунок вартості, застосування знижок, формування замовлення. Класи товарів і корзини залишаються незмінними при додаванні нових операцій.

Приклад із життя:

У компіляторі синтаксичні елементи (виклики методів, умовні вирази) обробляються різними відвідувачами: один перевіряє безпеку типів, інший

генерує код. Додавання нових кроків компіляції вимагає лише створення нового відвідувача без змін у класах елементів.

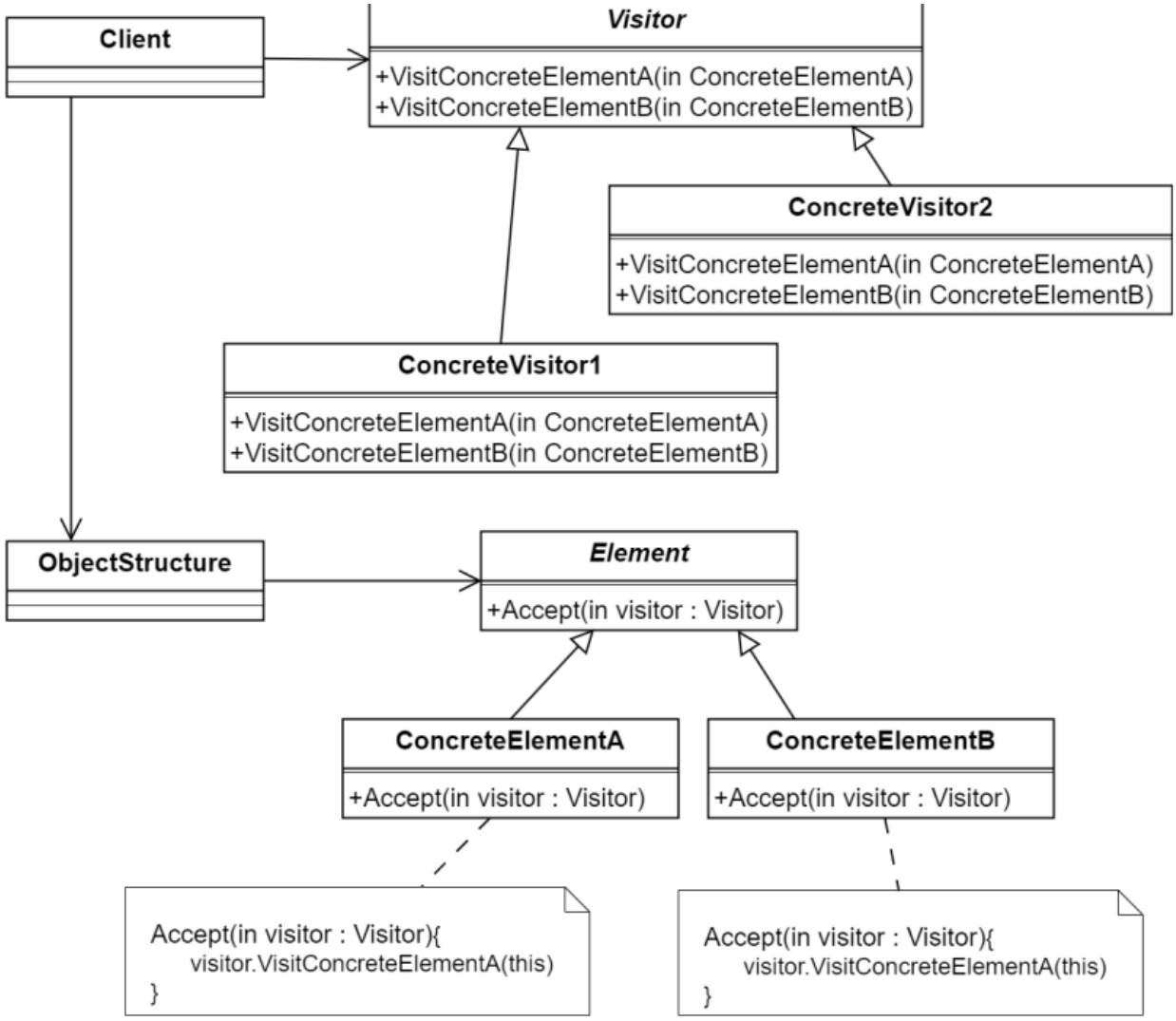
Переваги:

- Легко додавати нові операції над об’єктами.
- Логіка відокремлена від структури даних, що спрощує супровід.

Недоліки:

- Важко додавати нові типи елементів (потрібно змінювати всіх відвідувачів).

Структура патерна Відвідувач:



Хід роботи

1) Діаграма класів частини функціоналу робочої програми. (рис. 1)

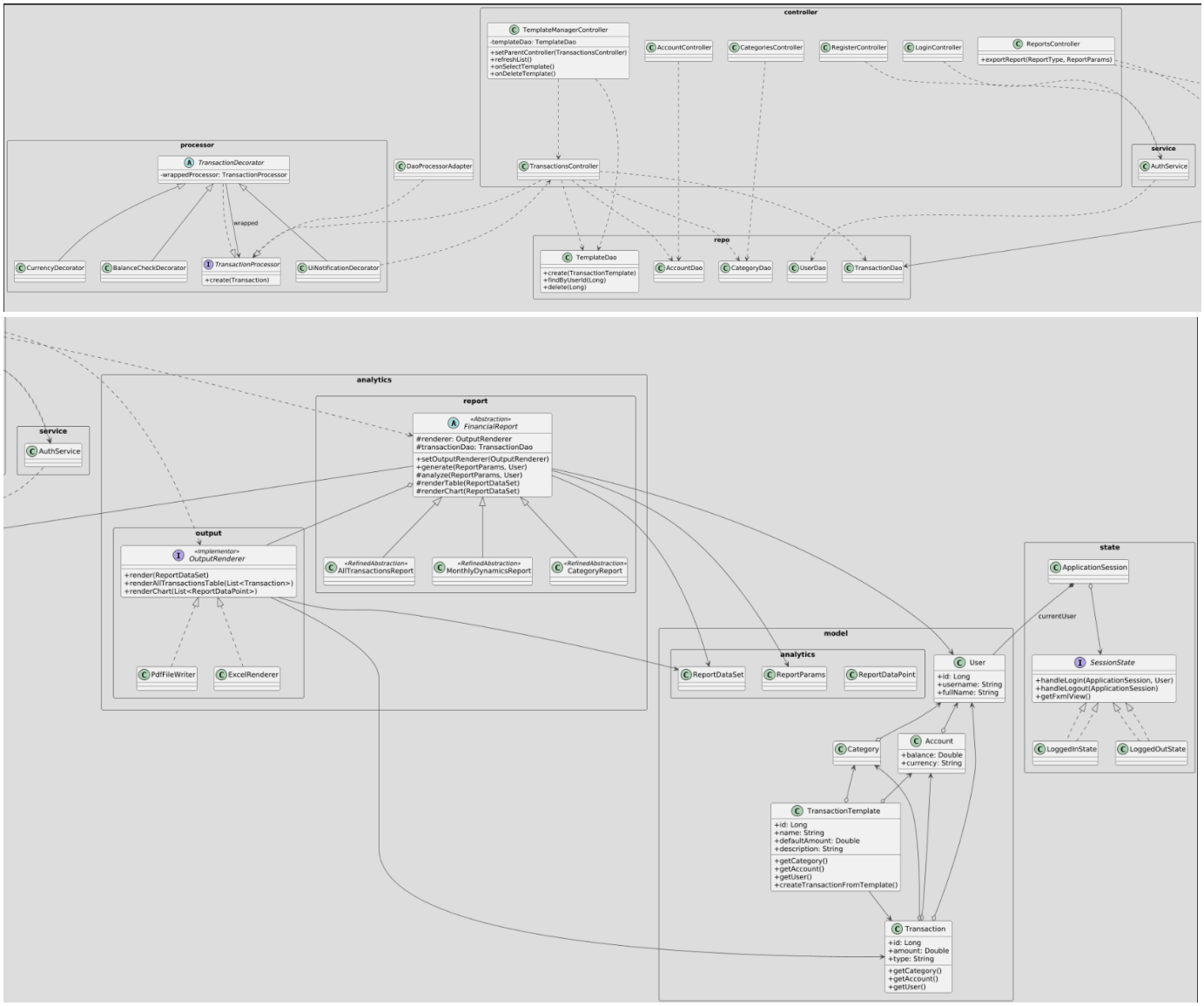
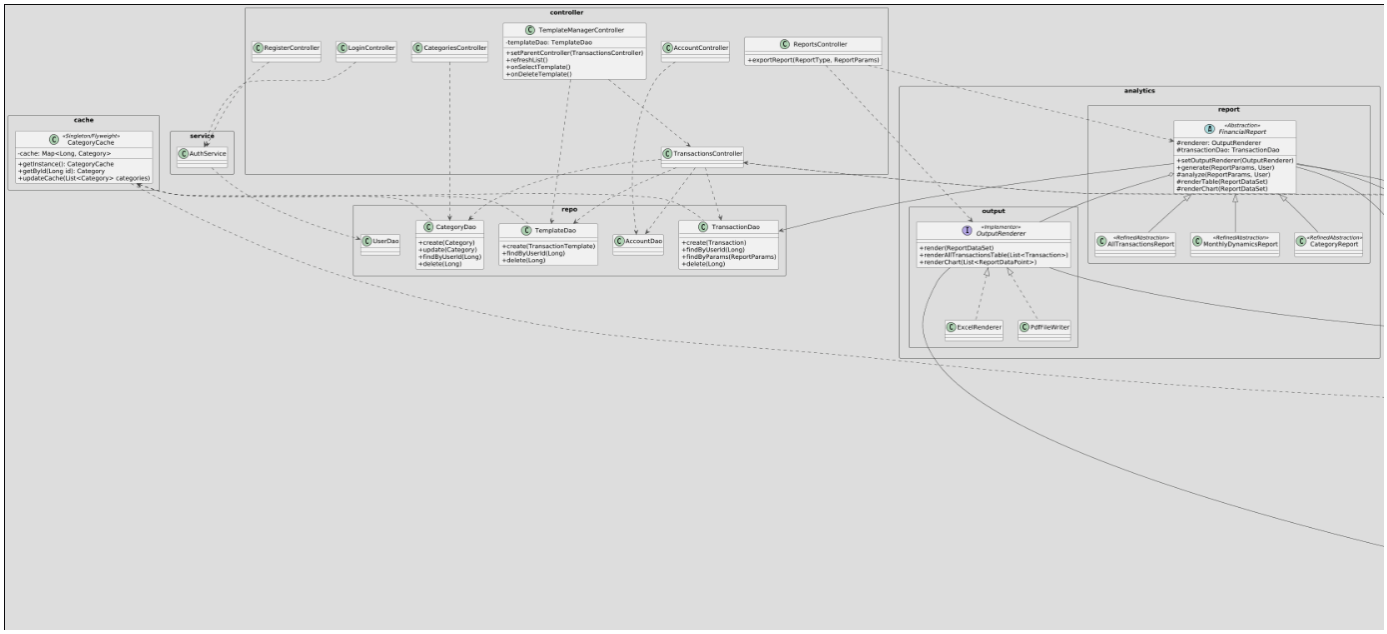


Рисунок 1 – Діаграма класів



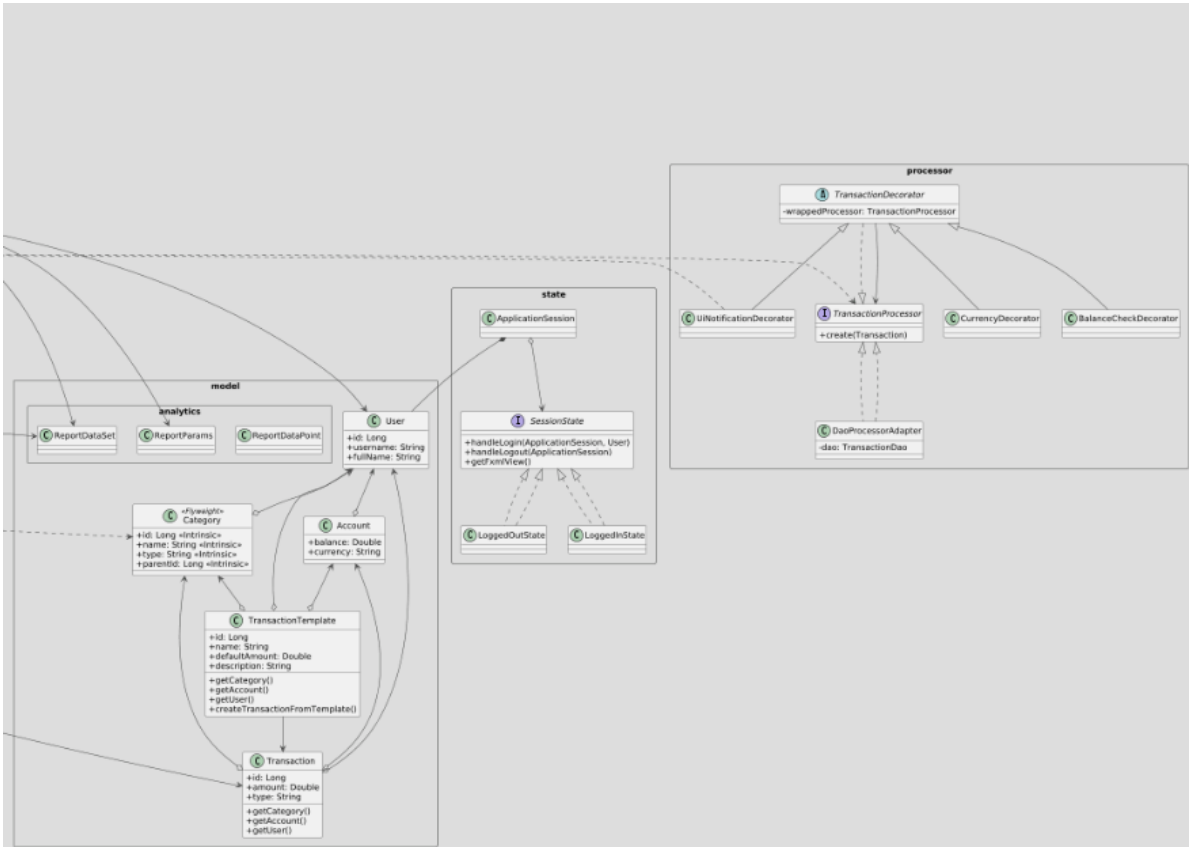


Рисунок 2 – Діаграма класів з патерном Flyweight

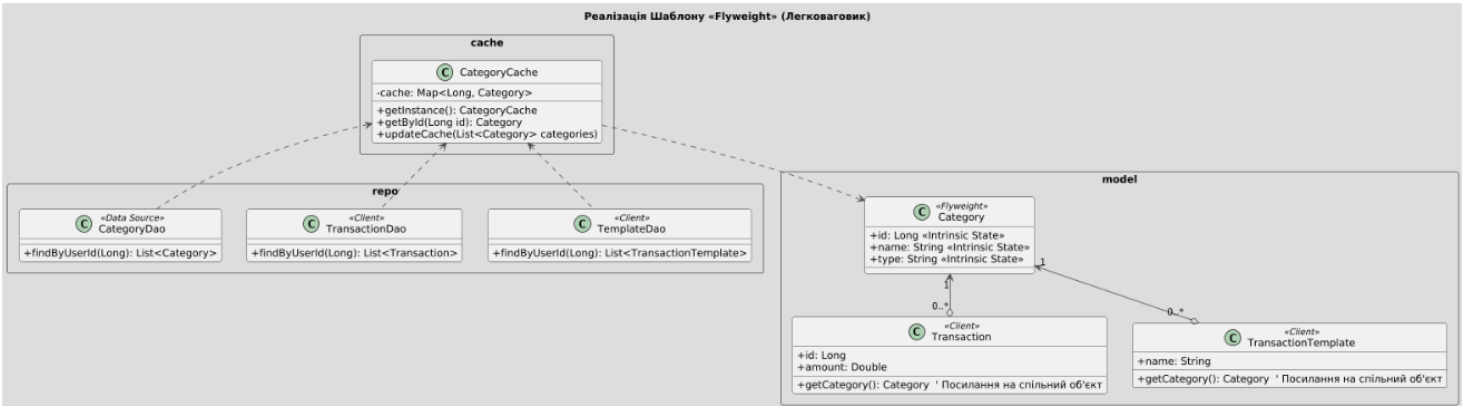


Рисунок 3 – Представлення самого шаблону на діаграмі

2) Реалізація Шаблону «Flyweight» (Легковаговик)

Патерн Flyweight (Легковаговик) належить до структурних патернів проєктування. Його основна мета — мінімізувати використання пам'яті шляхом спільного використання (sharing) максимальної кількості даних між схожими об'єктами. Замість зберігання тисяч копій об'єктів з однаковими даними, Flyweight дозволяє зберігати ці дані лише один раз.

У проєкті управління особистими фінансами шаблон Flyweight був застосований для об'єктів Category (Категорія). У системі може існувати тисячі об'єктів Transaction, але вони зазвичай посилаються на обмежений набір категорій (наприклад, 20-30). Створення нового об'єкта Category для кожної транзакції є неефективним марнотратством пам'яті.

1. Структура Патерну «Flyweight»

Реалізація патерну базується на розділенні стану об'єкта на внутрішній (спільний) та зовнішній (унікальний), а також на створенні "Фабрики" для керування спільними об'єктами.

1.1. Легковаговик (Flyweight) — Category.java

Це клас, об'єкти якого ми робимо спільними. Щоб бути безпечним для спільного використання, його внутрішній стан (дані, які ми кешуємо) є незмінним (immutable).

- Внутрішній стан (Intrinsic): Це дані, які є спільними для всіх екземплярів: id, name, type, parentId. Вони завантажуються з БД і не змінюються протягом життєвого циклу програми.
- Зовнішній стан (Extrinsic): Це контекст, у якому використовується Легковаговик. У нашому випадку, це сам об'єкт Transaction або TransactionTemplate, який містить посилання на спільний об'єкт Category.

1.2. Фабрика Легковаговиків (Flyweight Factory) — CategoryCache.java

Це центральний компонент патерну. Клас CategoryCache реалізований як Singleton і відповідає за створення та керування унікальними екземплярами Category.

- Кешування: CategoryCache містить Map<Long, Category>, де ключ — це category.id, а значення — єдиний спільний екземпляр (Flyweight) цієї категорії.
- Методи:
    - getById(Long id): Клієнти (наприклад, TransactionDao) викликають цей метод, щоб отримати об'єкт Category. Фабрика перевіряє, чи об'єкт вже є в кеші.
    - put(Category category) / updateCache(List<Category> categories): Використовуються CategoryDao для наповнення або оновлення кешу при завантаженні даних з БД.

1.3. Клієнти (Clients) — CategoryDao та TransactionDao

Клієнти — це класи, які використовують Легковаговики.

- CategoryDao: Виступає як постачальник для Фабрики. Після завантаження категорій з БД (findByUserId()) він негайно передає список у CategoryCache.updateCache(list), щоб наповнити кеш актуальними даними.
- TransactionDao / TemplateDao: Виступають як споживачі. Коли ці DAO завантажують транзакції або шаблони з БД, вони зчитують лише category_id. Замість створення нового об'єкта Category для кожної транзакції, вони викликають CategoryCache.getById(catId), отримуючи посилання на єдиний спільний екземпляр Category з кешу.

2. Принцип Роботи та Переваги

Зв'язок між компонентами виглядає так: TransactionDao (Клієнт) запитує об'єкт у CategoryCache (Фабрика), який повертає раніше збережений Category (Легковаговик).

Як працює Flyweight у вашому проєкті:

1. При запуску програми або відкритті екрана категорій CategoryDao.findByUserId() завантажує всі категорії з БД.
2. CategoryDao передає цей список у CategoryCache.updateCache(). Кеш (Map) наповнюється 50 унікальними об'єктами Category.
3. Користувач відкриває звіт, і TransactionDao.findByUserId() завантажує 10 000 транзакцій.

4. Під час обробки ResultSet, TransactionDao для кожної транзакції викликає CategoryCache.getById().
5. Результат: Усі 10 000 об'єктів Transaction у пам'яті посилаються лише на ті 50 об'єктів Category, які зберігаються в кеші.

Основні переваги використання цього шаблону:

- Значна економія пам'яті:
    - До (Без Flyweight): 10 000 транзакцій = 10 000 об'єктів Transaction + 10 000 об'єктів Category.
    - Після (З Flyweight): 10 000 транзакцій = 10 000 об'єктів Transaction + лише 50 об'єктів Category.
    - Це критично важливо для десктопного застосунку, який працює з великими обсягами даних.
- Підвищення продуктивності:
    - Усунено необхідність у повільному пошуку по списку (allCategories.stream().filter(...)) всередині циклу ResultSet у TransactionDao.
    - Замість цього використовується миттєвий пошук у хеш-мапі (CategoryCache.getById()), що значно прискорює завантаження та обробку транзакцій.
- Безпека (Immutable State):
    - Оскільки об'єкти Category тепер незмінні (immutable), їх безпечно використовувати спільно у багатьох потоках або об'єктах (Transaction, TransactionTemplate) без ризику, що зміна в одному місці випадково пошкодить дані в іншому.

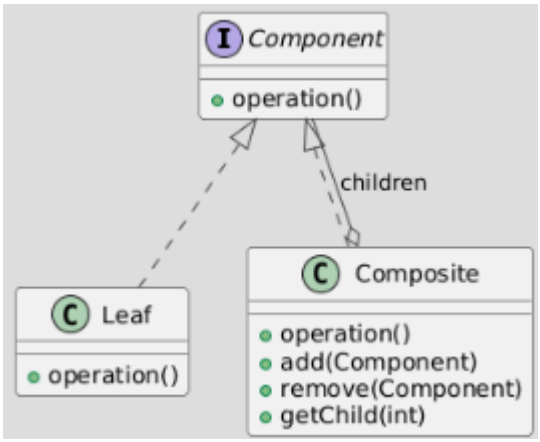
Таким чином, шаблон Flyweight оптимізував використання пам'яті та швидкість доступу до даних, зробивши систему більш ефективною при роботі з великими наборами транзакцій.

Відповіді на контрольні запитання:

1. Яке призначення шаблону «Композит»?

Компонувальник — це структурний патерн проектування, що дає змогу згрупувати декілька об’єктів у деревоподібну структуру, а потім працювати з нею так, ніби це одиничний об’єкт.

2. Нарисуйте структуру шаблону «Композит».



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

Шаблон «Композит» складається з трьох класів/інтерфейсів:

1. Component: Абстракція (інтерфейс/базовий клас), що визначає спільну поведінку для всіх елементів.
2. Leaf (Листок): Простий елемент, що не може мати дочірніх. Реалізує поведінку Component.
3. Composite (Композит): Складний елемент, що містить колекцію інших об'єктів Component (Листків або інших Композитів). Реалізує поведінку Component, делегуючи запити своїм дочірнім елементам.

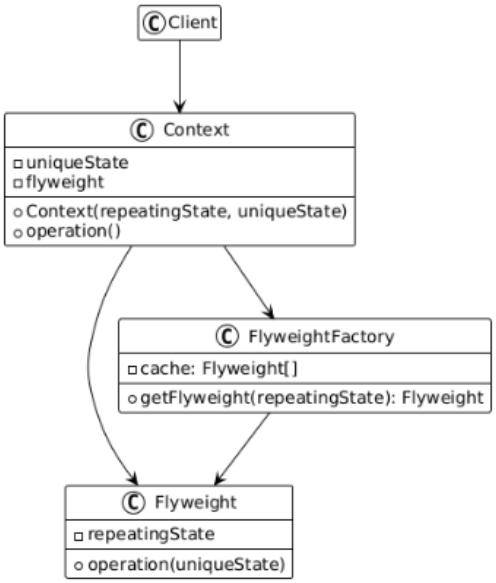
Взаємодія: Обидва (Leaf та Composite) успадковують/реалізують Component, що дозволяє клієнту працювати з ними однаково. Composite використовує композицію для побудови рекурсивної деревовидної структури.

4. Яке призначення шаблону «Легковаговик»?

Легковаговик — це структурний патерн проектування, що дає змогу вмістити більшу кількість об’єктів у відведеній оперативній пам’яті. Легковаговик заощаджує пам’ять, розподіляючи спільний стан об’єктів між собою, замість

зберігання однакових даних у кожному об’єкті.

5. Нарисуйте структуру шаблону «Легковаговик».



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

Шаблон «Легковаговик» (Flyweight) включає чотири основні класи/інтерфейси:

1. Flyweight (Легковаговик): Інтерфейс або абстрактний клас, який оголошує метод(и) для отримання зовнішнього стану (ExtrinsicState) і виконання операцій.
2. ConcreteFlyweight (Конкретний Легковаговик): Клас, що реалізує інтерфейс Flyweight. Він містить внутрішній стан (IntrinsicState), який є спільним і не залежить від контексту.
3. UnsharedConcreteFlyweight (Неспільний Конкретний Легковаговик): Спеціальний клас, який реалізує Flyweight, але не є спільним між об'єктами. Його використовують для великих або унікальних компонентів у структурі.
4. FlyweightFactory (Фабрика Легковаговиків): Відповідає за створення та управління об'єктами Flyweight. Вона перевіряє, чи існує вже запитуваний спільний об'єкт, і повертає його, або створює новий, якщо він відсутній.
5. Client (Клієнт): Об'єкт, який створює або отримує об'єкти Flyweight через фабрику і передає зовнішній стан (ExtrinsicState) як аргумент під час виклику операцій.

Взаємодія спрямована на мінімізацію використання пам'яті через розділення стану на спільний та унікальний:

- Фабрика як точка доступу: Client ніколи не створює ConcreteFlyweight безпосередньо. Він запитує його у FlyweightFactory, передаючи ключ (наприклад, тип об'єкта).
- Спільне використання стану (Intrinsic State): FlyweightFactory використовує мапу або іншу структуру даних для зберігання та повернення існуючих об'єктів ConcreteFlyweight. Внутрішній стан (IntrinsicState) залишається незмінним і спільним.

- Унікальний стан (Extrinsic State): Client зберігає унікальний (зовнішній) стан, який передається об'єкту Flyweight як параметр методу, коли викликається операція.
- Зворотний зв'язок: Об'єкт Flyweight використовує свій спільний IntrinsicState у поєднанні з переданим ExtrinsicState для виконання повної операції.

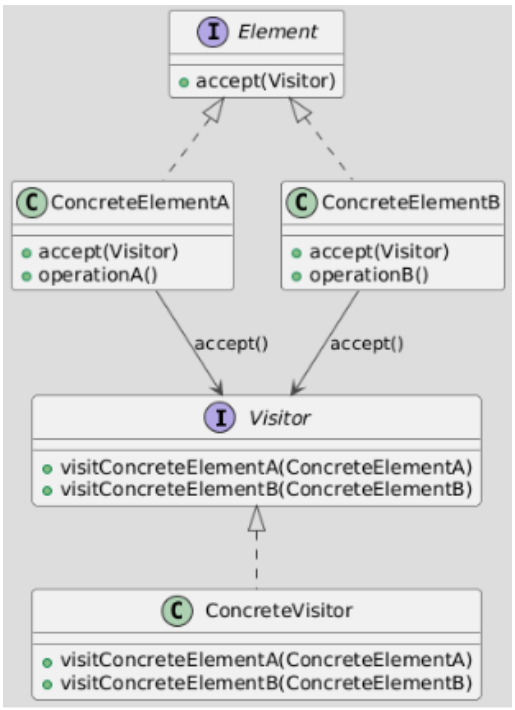
7. Яке призначення шаблону «Інтерпретатор»?

Шаблон Інтерпретатор (Interpreter) визначає граматику мови і спосіб її інтерпретації для обчислення виразів. Використовується для побудови інтерпретаторів простих мов або виразів.

8. Яке призначення шаблону «Відвідувач»?

Відвідувач — це поведінковий патерн проектування, що дає змогу додавати до програми нові операції, не змінюючи класи об’єктів, над якими ці операції можуть виконуватися.

9. Нарисуйте структуру шаблону «Відвідувач».



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

Шаблон «Відвідувач» включає п'ять основних класів/інтерфейсів:

1. Visitor (Відвідувач): Інтерфейс, який оголошує метод visit() для кожного конкретного класу Елемента.

2. ConcreteVisitor (Конкретний Відвідувач): Клас, який реалізує інтерфейс Visitor. Він містить алгоритм або логіку, яку потрібно виконати для відповідного ConcreteElement.
3. Element (Елемент): Інтерфейс, який оголошує метод accept(Visitor) для прийому відвідувача.
4. ConcreteElement (Конкретний Елемент): Клас, що реалізує інтерфейс Element. У методі accept(Visitor) він викликає відповідний метод visit() об'єкта Visitor, передаючи себе як аргумент (visitor.visitConcreteElementA(this)).
5. ObjectStructure (Структура Об'єктів): Колекція, яка перелічує елементи (зазвичай, список або дерево) і може мати метод для перебору та застосування Visitor до всіх своїх Element.

Головна мета — відокремити алгоритм від структури об'єктів, над якою він працює.

1. Запуск: Client ініціює процес, створюючи ConcreteVisitor і передаючи його структурі ObjectStructure.
2. Прийняття: ObjectStructure (або Client) викликає метод accept(Visitor) на кожному ConcreteElement.
3. Подвійна Диспетчеризація: Усередині методу accept() об'єкт ConcreteElement викликає відповідний метод visit() на переданому об'єкті Visitor, передаючи себе (this) як параметр.
4. Виконання: ConcreteVisitor виконує логіку, використовуючи специфічний тип ConcreteElement, який йому передали. Це дозволяє легко додавати нові операції (нових ConcreteVisitor), не змінюючи класи Element.

Висновок:

У ході лабораторної роботи було вивчено структуру та принципи функціонування шаблонів проєктування Composite, Flyweight (Пристосуванець), Interpreter та Visitor. Опрацьовано їх призначення, ключові компоненти та взаємодію між класами, а також особливості застосування цих патернів у різних програмних ситуаціях. Крім того, набуті практичні навички побудови UML-діаграм та роботи зі структурними і поведінковими патернами проектування.

У практичній частині було реалізовано патерн Flyweight, що дозволяє оптимізувати використання пам’яті при роботі з великою кількістю однотипних об’єктів, виділяючи спільні внутрішні дані та уникаючи дублювання стану. Використання цього патерну забезпечило підвищення ефективності та продуктивності системи, спростило керування ресурсами та створило можливість гнучкого масштабування програмного забезпечення. Такий підхід дозволяє економити пам’ять без втрати функціональності і полегшує інтеграцію нових компонентів системи.