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**
з предмету «Технології розроблення програмного забезпечення»
на тему «**Шаблони «Composite», «Flyweight», «Interpreter», «Visitor»**»

Виконав
студент групи ІА-23:
Воронюк Є. В.

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Мета: реалізація частини функціоналу робочої програми у вигляді класів та їхньої взаємодії із застосуванням одного із розглянутих шаблонів проектування.

Завдання 1: ознайомитися з короткими теоретичними відомостями.
Завдання 2: реалізувати частину функціоналу робочої програми у вигляді класів і їх взаємодій для досягнення конкретних функціональних можливостей.
Завдання 3: застосування одного з розглянутих шаблонів при реалізації програми.

Тема для виконання:

22 - FTP-server (state, builder, memento, template method, visitor, client- server)

FTP-сервер повинен вміти коректно обробляти і відправляти відповіді по протоколу FTP, з можливістю створення користувачів (з паролями) і доступних їм папок, розподілу прав за стандартною схемою (rwe), ведення статистики з'єднань, обмеження максимальної кількості підключень і максимальної швидкості поширення глобально і окремо для кожного облікового запису.

Короткі теоретичні відомості:

Шаблон «Composite»

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.

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.

Проблема: патерн ''Composite'' (Компонувальник) має сенс тільки тоді, коли основна модель вашої програми може бути структурована у вигляді дерева.

Наприклад, є два об'єкти: Продукт і Коробка. Коробка може містити кілька Продуктів і інших Коробок поменше. Ті, в свою чергу, теж містять або Продукти, або Коробки і так далі.

Тепер припустимо, ваші Продукти харчування й Коробки можуть бути частиною замовлень. Кожне замовлення може містити як прості Продукти без упаковки, так і складові Коробки. Ваше завдання полягає в тому, щоб дізнатися ціну всього замовлення.

Якщо вирішувати завдання в лоб, то вам буде потрібно відкрити всі коробки замовлення, перебрати всі продукти і порахувати їх сумарну вартість. Але це занадто клопітно, так як типи коробок і їх вміст можуть бути вам невідомі. Крім того, наперед невідомо і кількість рівнів вкладеності коробок, тому перебрати коробки простим циклом вийде.

Рішення: компонувальник пропонує розглядати Продукт і Коробку через єдиний інтерфейс із загальним методом отримання вартості.

Продукт просто поверне свою ціну. Коробка запитає ціну кожного предмета всередині себе і поверне суму результатів. Якщо одним з внутрішніх предметів виявиться коробка поменше, вона теж буде перебирати свій вміст, і так далі, поки не будуть пораховані всі складові частини.

Для вас, клієнта, головне, що тепер не потрібно нічого знати про структуру замовлень. Ви викликаєте метод отримання ціни, він повертає цифру, а ви не тонете в горах картону і скотча.

Переваги та недоліки:

+ Спрощує архітектуру клієнта при роботі зі складним деревом компонентів.
+ Полегшує додавання нових видів компонентів.
- Створює занадто загальний дизайн класів.

Шаблон «Flyweight»

Шаблон використовується для зменшення кількості об'єктів в додатку шляхом поділу цих об'єктів між ділянками додатку. Flyweight являє собою поділюваний об'єкт, здається, ніби для кожної літери існує окремий об'єкт. Насправді фізично об'єкт всього один, існує лише безліч посилань на нього.

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).

Проблема: на дозвіллі ви вирішили написати невелику гру, в якій гравці переміщуються по карті та стріляють один в одного. Фішкою гри повинна була стати реалістична система частинок. Кулі, снаряди, уламки від вибухів — все це повинно реалістично літати та гарно виглядати. Кожна частинка у грі представлена власним об'єктом, що має безліч даних. У певний момент, коли побоїще на екрані досягає кульмінації, оперативна пам'ять комп'ютера вже не може вмістити нові об'єкти частинок, і програма «вилітає».

Рішення: якщо уважно подивитися на клас частинок, то можна помітити, що колір і спрайт займають найбільше пам'яті. Більше того, ці поля зберігаються в кожному об'єкті, хоча фактично їхні значення є однаковими для більшості частинок.
Інший стан об'єктів — координати, вектор руху й швидкість відрізняються для всіх частинок. Таким чином, ці поля можна розглядати як контекст, у якому використовується частинка, а колір і спрайт — це дані, що не змінюються в часі.

Переваги та недоліки:
+ Заощаджує оперативну пам'ять.
- Витрачає процесорний час на пошук/обчислення контексту.
- Ускладнює код програми внаслідок введення безлічі додаткових класів.

Шаблон «Interpreter»

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.

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

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

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

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.

Шаблон «Visitor»

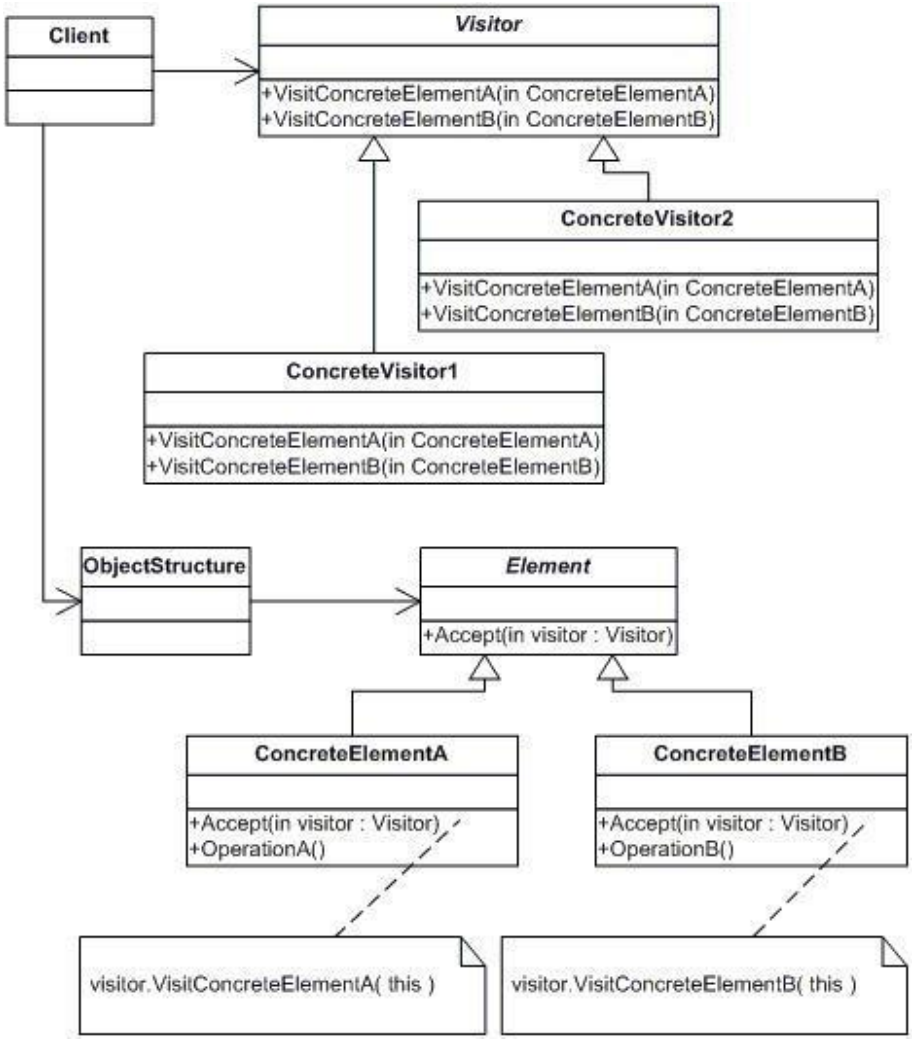

Рисунок 1 – загальна структура шаблону «Visitor»

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).

Даний шаблон дозволяє групувати однотипні операції, що застосовуються над різнотипними об'єктами.

Проблема: ваша команда розробляє програму, що працює з геоданими у вигляді графа. Вузлами графа можуть бути як міста, так інші локації, такі, як пам’ятки, великі підприємства тощо. Кожен вузол має посилання на найближчі до нього вузли. Для кожного типу вузла існує свій власний клас, а кожен вузол представлений окремим об’єктом.

Ваше завдання — зробити експорт цього графа до XML. Справа була б легкою, якщо б ви могли редагувати класи вузлів. У цьому випадку можна було б додати метод експорту до кожного типу вузлів, а потім, перебираючи всі вузли графа, викликати цей метод для кожного вузла. Завдяки поліморфізму, рішення було б елегантним, оскільки ви могли б не прив’язуватися до конкретних класів вузлів.

Але, на жаль, змінити класи вузлів у вас не вийшло. Системний архітектор сказав, що код класів вузлів зараз дуже стабільний, і від нього багато що залежить, а тому він не хоче ризикувати, дозволяючи будь-кому чіпати цей код.

Рішення: патерн пропонує розмістити нову поведінку в окремому класі, замість того, щоб множити її відразу в декількох класах. Об’єкти, з якими повинна бути пов’язана поведінка, не виконуватимуть її самостійно. Замість цього ви будете передавати ці об’єкти до методів відвідувача. Код поведінки, імовірно, повинен відрізнятися для об’єктів різних класів, тому й методів у відвідувача повинно бути декілька.

Хід роботи::

Реалізація патерну:

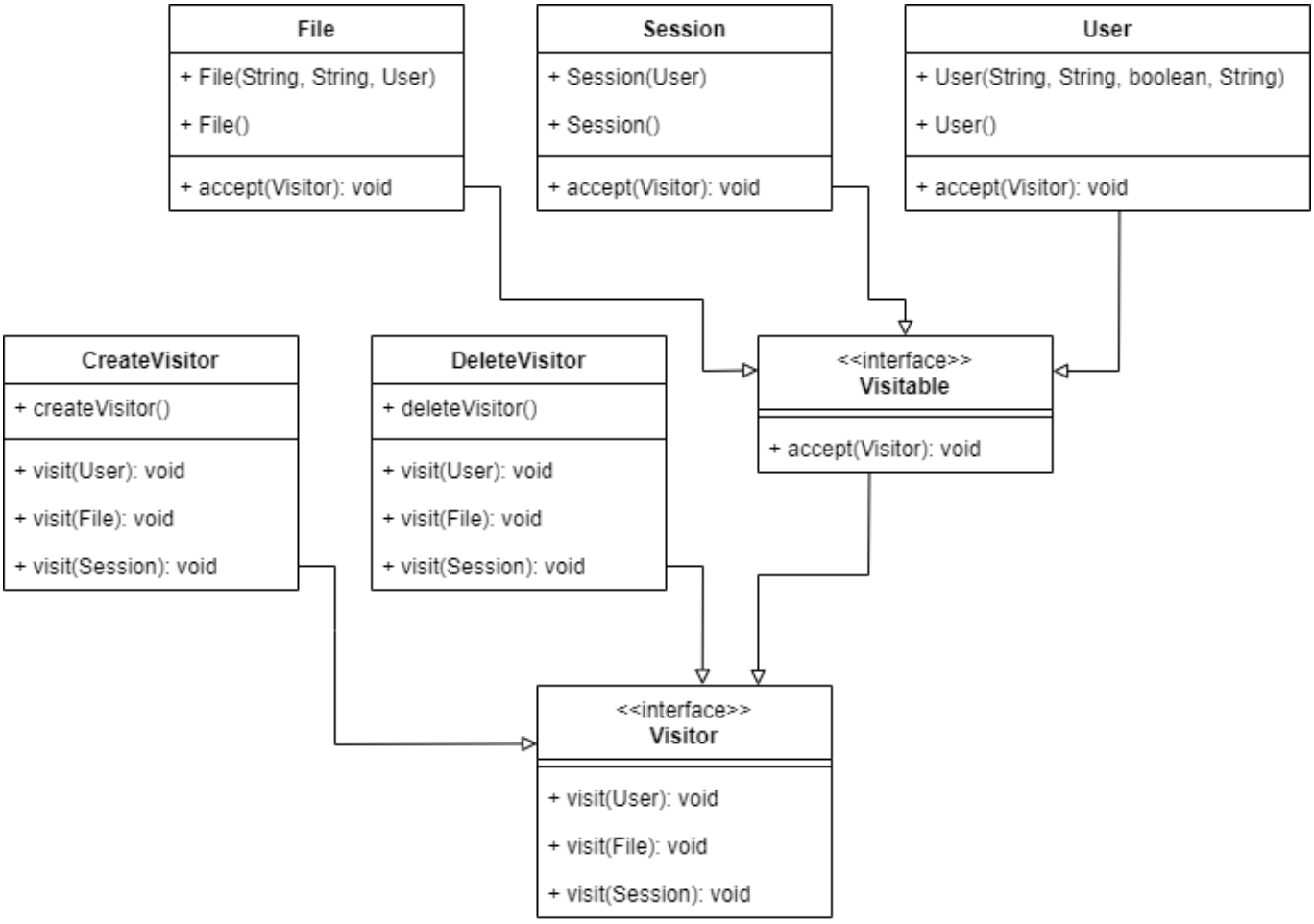

Рисунок 2 – реалізація патерну Visitor

Опис реалізації патерну:

Патерн Visitor використовується для додавання різного функціоналу до класів моделі FTP-серверу, таких як User, Session, File, не змінюючи їхньої структури. Цей паттерн дозволяє додавати нові операції, не модифікуючи класи, що реалізують модель, і робить код більш гнучким для подальших змін. Зокрема, шаблон використовується для додавання операцій створення та видалення. Через реалізацію шаблону відбувається створення користувачів, сесій користувачів, створення (завантаження) файлів; видалення файлів. Можливе розширення для видалення користувачів та записів про сесії.

Опис класів та інтерфейсів:

Visitable(інтерфейс) – містить метод accept(Visitor): void, дозволяє об'єктам приймати Visitor.

Visitor(інтерфейс) - описує інтерфейс для відвідувачів із методами: visit(User): void — обробка об'єкта User, visit(File): void — обробка об'єкта File, visit(Session): void — обробка об'єкта Session.
Конструктори класів дозволяють ініціалізувати об’єкти із параметрами.
File(клас) - реалізує інтерфейс Visitable. Метод у класі: accept(Visitor): void — викликає метод visit(File) у відвідувача.
Session(клас) – реалізує інтерфейс Visitable. Містить метод: accept(Visitor): void — викликає метод visit(Session) у відвідувача.
User (клас) – також реалізує інтерфейс Visitable. Метод у класі: accept(Visitor): void — викликає метод visit(User) у відвідувача.
CreateVisitor містить логіку створення об’єктів, а DeleteVisitor виконує логіку видалення об’єктів.

Логіка роботи:
Об'єкти (File, User, Session) реалізують інтерфейс Visitable, що дозволяє їм "приймати" відвідувачів.
Відвідувачі CreateVisitor і DeleteVisitor реалізують інтерфейс Visitor і надають методи для роботи з кожним типом об'єктів. Таким чином, CreateVisitor додає логіку створення, а DeleteVisitor — логіку видалення об'єктів без зміни самих класів User, File та Session.
Цей шаблон спрощує додавання нових операцій для об'єктів і дотримується принципу Open/Closed – відкритість для розширення та закритість для модифікацій.

Вихідний код системи:
Вихідний код системи розміщено на GitHub: https://github.com/dobrogo-dnia/SDT-labs

Висновок: під час виконання даної лабораторної роботи було реалізовано частину функціоналу робочої програми у вигляді класів та їхньої взаємодії із застосуванням одного із розглянутих шаблонів проектування – а саме шаблону Visitor. Існуючий код було оновлено для використання відповідного патерну.