

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. FTP-server»

Виконав:

студент групи ІА-34
Сухоручкін Гліб

Перевірив:

Асистент кафедри ІСТ
Мягкий М.Ю.

Зміст:

1. Вступ
2. Теоретичні відомості
3. Хід роботи
4. Діаграма класів
5. Вихідний код
6. Відповіді на питання до лабораторної роботи
7. Висновки

Вступ

Метою цієї роботи є вивчити структуру шаблонів «Composite», «Flyweight» (Пристосуванець), «Interpreter», «Visitor» та навчитися застосовувати їх в реалізації програмної системи.

Проекторваною системою цієї лабораторної роботи є FTP-server (використовуючи 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.

Теоретичні відомості

Шаблон «Visitor»

Призначення: 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: Ви розробляєте онлайн-корзину інтернет магазину. 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 додати ще логіку розрахунку вартості з врахуванням знижки, то потрібно буде додати ще цю логіку до товарів. А якщо буде ще сезонна знижка, то ми знову будемо додавати 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.

Приклад з життя: Прикладом може служити написання компілятора. Припустимо, існують різні об'єкти в синтаксисі мови програмування: виклики методів і умовні вирази. Компілятор перед генерацією коду повинен обійти всі вирази (і виклики методів, і умовні вирази) і перевірити безпеку типів, після чого згенерувати відповідний код. Відповідно буде два відвідувачі – для перевірки безпеки типів і для генерації коду. У кожного з них буде по 2 методи – для викликів методів і для умовних операцій. Таким чином при необхідності додавання нових кроків компіляції досить буде визначити нового «відвідувача» і викликати його у відповідний час.

Хід роботи

Діаграма класів

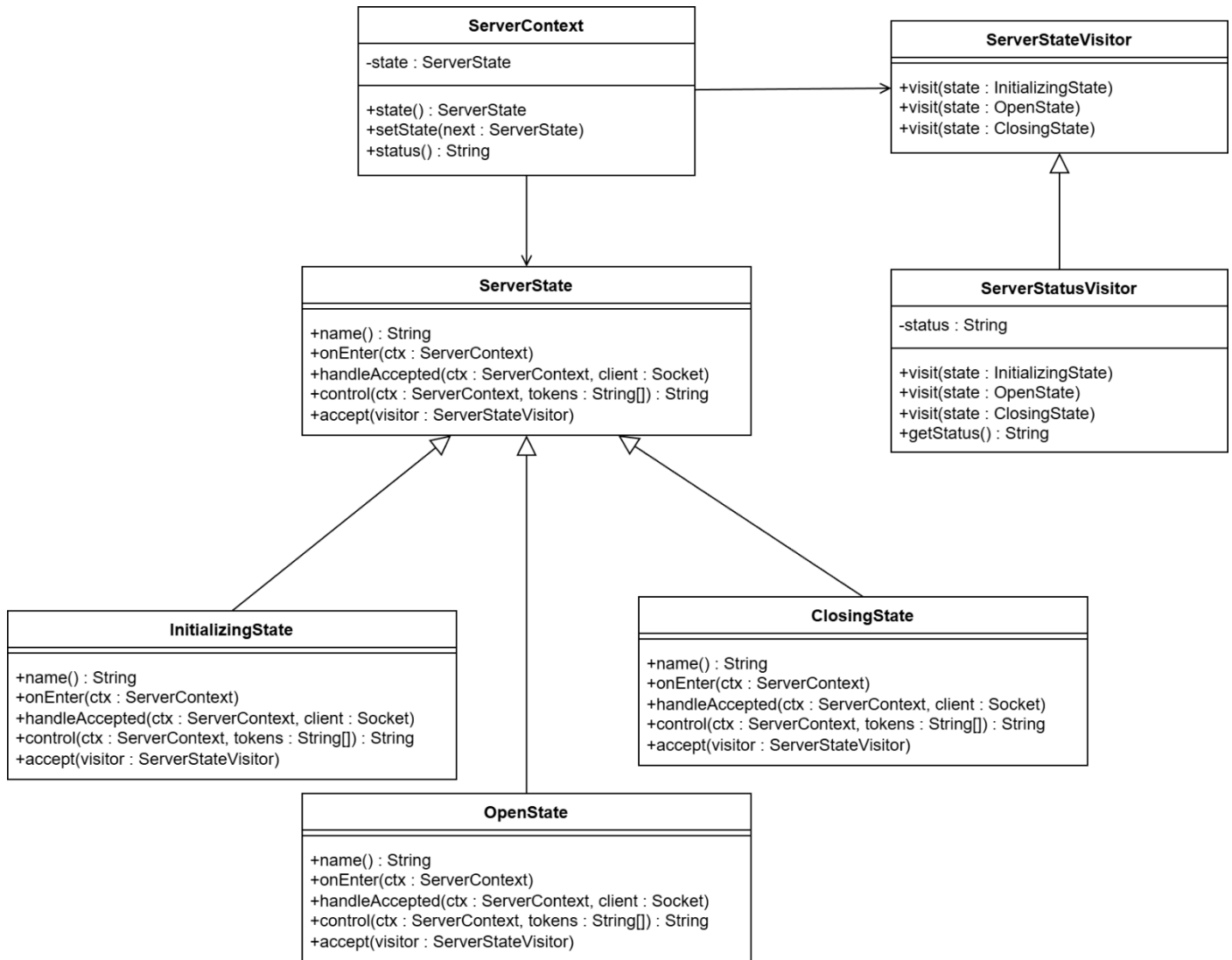


Рис 1. Патерн «Visitor» у FTP-сервері.

Пояснення діаграми класів

ServerContext – контекст всього сервера й клієнт патерну Visitor. Містить поле `-state : ServerState` з поточним станом, дає до нього доступ через `state()` і дозволяє його змінювати через `setState(next : ServerState)`. Метод `status()` не запитує стан напряму, а делегує побудову текстового статусу відвідувачу: створює **ServerStatusVisitor**, викликає `state.accept(visitor)` і потім бере результат через `getStatus()`. Таким чином, логіка «як перетворювати стан у рядок» винесена з контексту в окремий об'єкт-відвідувач.

ServerState – інтерфейс «елемента», над яким працює Visitor. Описує спільний контракт для всіх станів сервера: `name()` повертає ім'я стану; `onEnter(ctx)` виконує ініціалізацію при вході в стан; `handleAccepted(ctx, client)` обробляє нове підключення; `control(ctx, tokens)` – команди керування станом. Ключовий для патерну метод `accept(visitor : ServerStateVisitor)` – «точка входу» відвідувача, через яку конкретний стан передає себе у відповідний `visit(...)`.

InitializingState / **OpenState** / **ClosingState** – конкретні «елементи» (ConcreteElement) патерну Visitor і водночас конкретні стани сервера. Усі наслідують **ServerState**,

реалізують поведінку для своїх фаз (що робити при підключенні, які команди керування приймати) і перевизначають `accept(visitor : ServerStateVisitor)`, де викликають відповідний метод відвідувача (`visitor.visit(this)`). Завдяки цьому нові операції над станами можна додавати через нові реалізації `ServerStateVisitor`, не змінюючи самі стани.

ServerStateVisitor – інтерфейс «відвідувача». Оголошує перевантажені методи `visit(state : InitializingState)`, `visit(state : OpenState)`, `visit(state : ClosingState)` – по одному для кожного конкретного стану. Саме так Visitor патерн досягає подвійної диспетчеризації: конкретний тип стану визначається під час виклику `accept`, а конкретний тип відвідувача – під час створення реалізації цього інтерфейсу.

ServerStatusVisitor – конкретний відвідувач (`ConcreteVisitor`). Містить поле `-status : String`, у яке записує результат обробки стану. У кожному з методів `visit(...)` він, спираючись на конкретний тип стану, формує відповідний текстовий статус (наприклад, "STATE OPEN" або "STATE INITIALIZING"), а метод `getStatus()` повертає побудований рядок. Таким чином, задача «як описати поточний стан сервера текстом» інкапсульована у `Visitor` й відділена і від `ServerContext`, і від самих класів станів.

Вихідний код

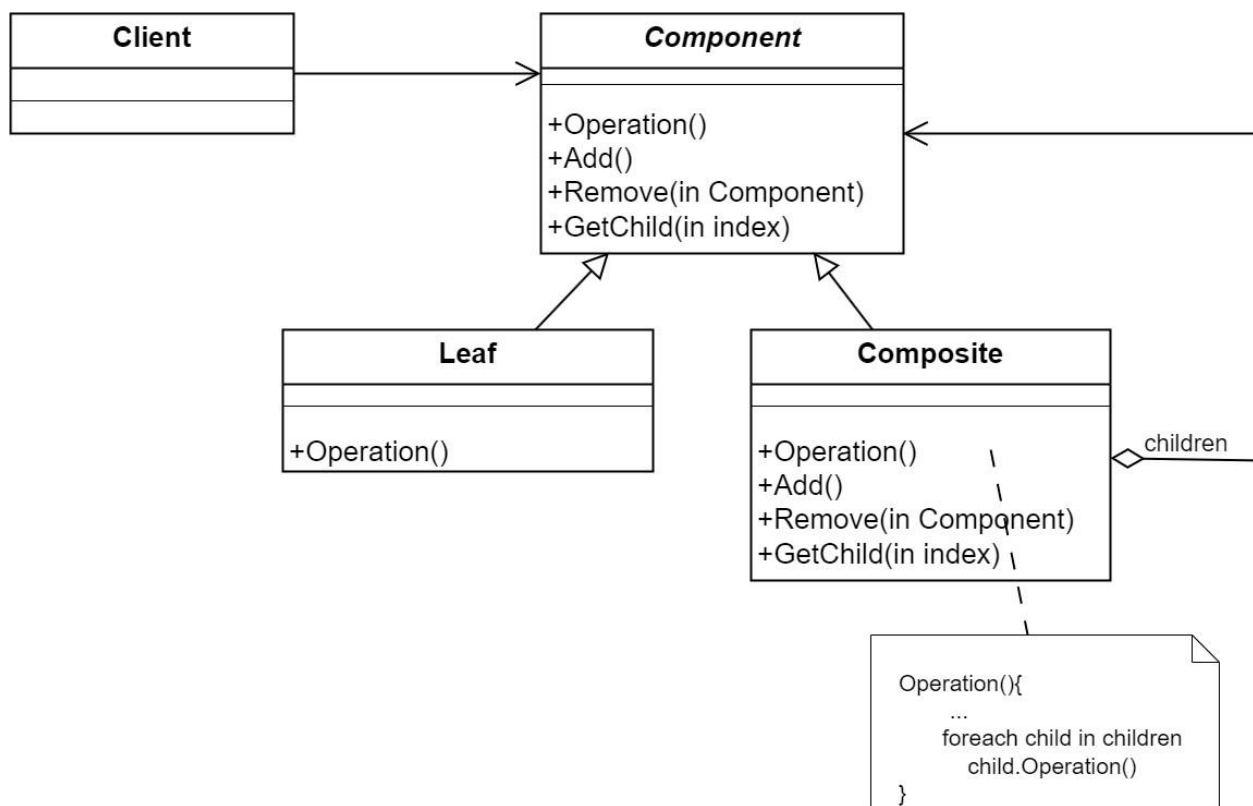
<https://github.com/GlebTiKsTRPZ/la8>

Відповіді на питання до лабораторної роботи

1. Яке призначення шаблону «Композит»?

Використовувати деревоподібну структуру «частина–ціле», щоб однаково обробляти окремі об'єкти й складені об'єкти з вкладеністю.

2. Нарисуйте структуру шаблону «Композит».



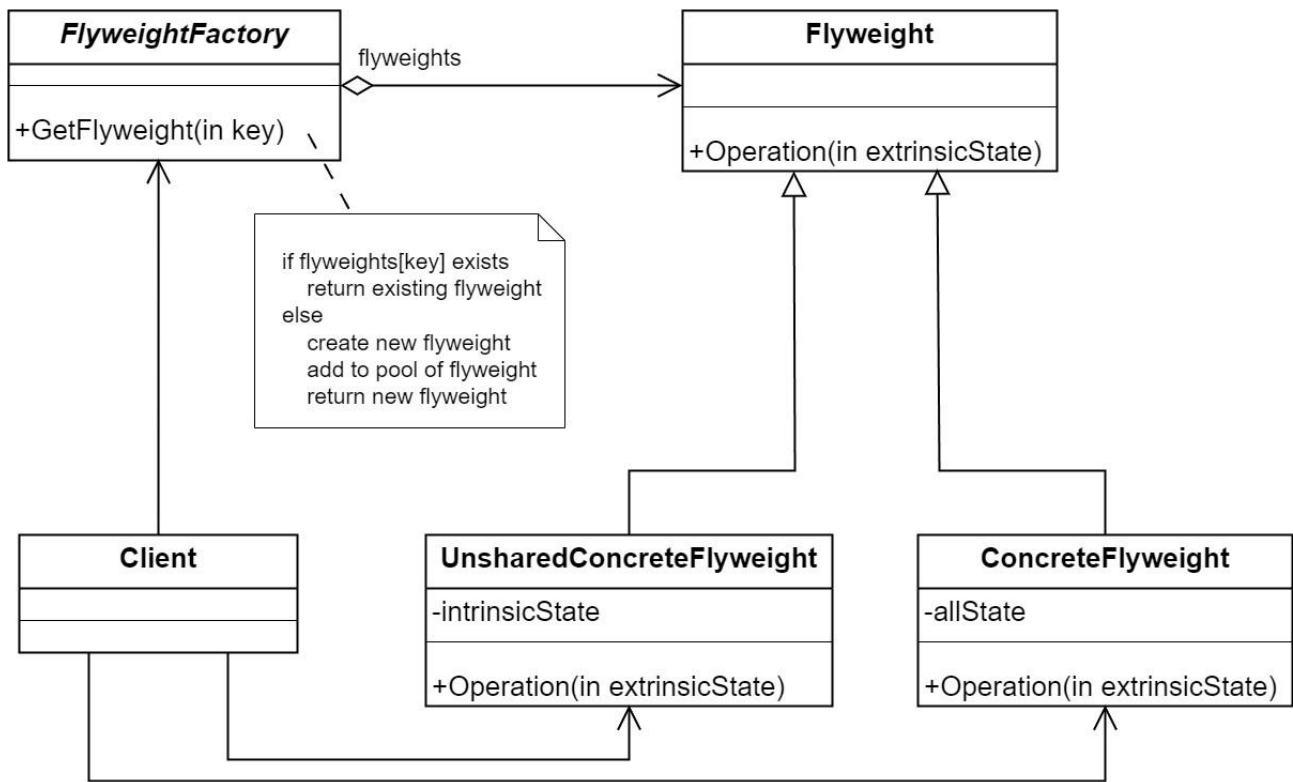
3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

Є спільний інтерфейс `ITask`, від якого наслідуються `Feature`, `Userstory` і `Task`; `Feature` та `Userstory` – складні об'єкти, що містять колекції `ITask`, а `Task` – «лист» без дочірніх; усі реалізують метод `GetEstimatedPoints()`, а клієнтський код працює тільки з `ITask` і викликає цей метод, не знаючи, чи є в об'єкта діти.

4. Яке призначення шаблону «Легковаговик»?

Зменшити кількість об'єктів у програмі, розділяючи один поділюваний об'єкт між багатьма місцями, коли потрібно багато однакових сутностей, і таким чином економити пам'ять.

5. Нарисуйте структуру шаблону «Легковаговик».



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

Є поділюваний об'єкт-легковаговик з внутрішнім станом (спільні дані, напр. код букви) і численні об'єкти контексту, які тримають зовнішній стан (рядок, стовпчик тощо) і при виклику операцій передають його легковаговику; один легковаговик таким чином використовується багатьма контекстами.

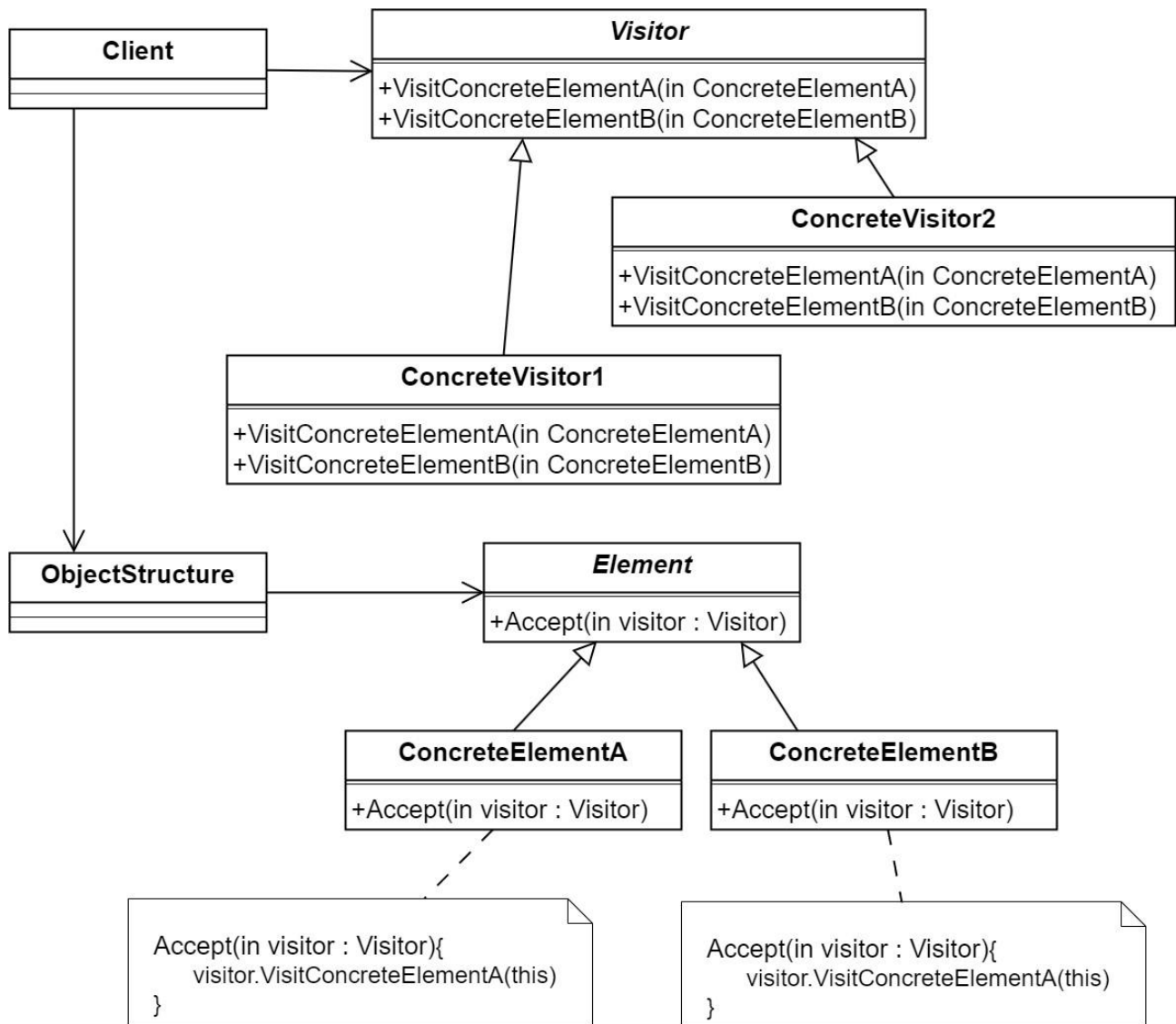
7. Яке призначення шаблону «Інтерпретатор»?

Подати граматику мови й інтерпретатор у вигляді абстрактного синтаксичного дерева з термінальними та нетермінальними виразами, які рекурсивно інтерпретуються в заданому контексті, щоб гнучко змінювати й розширювати правила мови.

8. Яке призначення шаблону «Відвідувач»?

Дозволити задавати нові операції над елементами ієрархії без зміни їхніх класів, винісши логіку обробки в окремі класи-відвідувачі та групуючи однотипні операції над різними типами об'єктів.

9. Нарисуйте структуру шаблону «Відвідувач».



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

Є класи-елементи (у прикладі – різні типи товарів у корзині) і базовий клас/інтерфейс відвідувача з методами для кожного типу елемента; конкретні відвідувачі реалізують різні варіанти логіки (звичайний розрахунок, зі знижкою, сезонні знижки тощо), елементи «знають свій тип», приймають відвідувача й викликають у нього відповідний метод, а логіка при цьому відділена від даних товарів.

Висновок

У цій лабораторній роботі реалізовано патерн «Visitor» для ієрархії станів FTP-сервера: інтерфейс `ServerState` доповнено методом `accept(ServerStateVisitor)`, конкретні стани `InitializingState`, `OpenState` та `ClosingState` виступають елементами, а інтерфейс `ServerStateVisitor` разом із класом `ServerStatusVisitor` інкапсулюють операцію побудови текстового статусу сервера, що дозволило відокремити логіку обробки станів від їх структури й спростило подальше розширення функціональності без зміни коду самих станів.