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 №6

Технології розроблення програмного забезпечення

ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

FTP-server

Виконав
студент групи ІА–22:
Білокур Євгеній
Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

Короткі теоретичні відомості 3
Реалізувати не менше 3-х класів відповідно до обраної теми 8
Структура класів 8
Опис класів 9
Реалізувати один з розглянутих шаблонів за обраною темою 10
Опис шаблону 10
Діаграма класів 11
Висновки та код 12

Тема: Шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Мета: Ознайомитися з шаблонами проектування «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та набути практичних навичок їх застосування. Реалізувати частину функціоналу програми за допомогою одного з розглянутих шаблонів для досягнення конкретних функціональних можливостей та забезпечення ефективної взаємодії між класами.

Хід роботи

..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

Принципи проектування SOLID

Принципи SOLID — це набір з п’яти принципів, які допомагають розробникам створювати зручні для підтримки, гнучкі та масштабовані об'єкти в об'єктноорієнтованих системах. Ось основні з них:

1. **Принцип єдиного обов'язку (SRP)**
   Кожен клас повинен мати лише одну причину для змін, тобто він повинен виконувати лише одну задачу. Це допомагає знизити складність коду та підвищити його підтримуваність. Якщо клас має кілька обов'язків, це ускладнює його модифікацію та тестування.
2. **Принцип відкритості/закритості (OCP)**
   Програмні компоненти повинні бути відкритими для розширення, але закритими для змін. Це дозволяє змінювати поведінку системи через розширення без необхідності змінювати існуючий код, що зменшує ризик введення помилок при внесенні змін.
3. **Принцип підстановки Барбари Лісков (LSP)**
   Об’єкти підкласу повинні заміняти об'єкти базового класу без порушення коректності програми. Підкласи не повинні змінювати очікувану поведінку батьківських класів, що забезпечує безпеку і надійність при використанні спадкування.

4. **Принцип розділення інтерфейсу (ISP)**
   Інтерфейси повинні бути спеціалізованими, тобто кожен інтерфейс повинен відповідати за одну конкретну задачу. Це дозволяє спростити взаємодію з системою, оскільки клієнт має справу тільки з тими методами, які йому потрібні.
5. **Принцип інверсії залежностей (DIP)**
   Модулі верхнього рівня не повинні залежати від модулів нижнього рівня, а обидва повинні залежати від абстракцій. Деталі реалізації не повинні впливати на абстракції, що дозволяє спрощувати тестування та змінювати реалізацію без впливу на основну логіку системи.

Ці принципи допомагають створювати чистий, зрозумілий і підтримуваний код, знижуючи ймовірність помилок і забезпечуючи легке розширення системи без порушення її стабільності.

Шаблон "Abstract Factory" (Абстрактна фабрика)

Призначення:
Шаблон "Abstract Factory" забезпечує створення сімейства пов'язаних або залежних об'єктів без вказівки їх конкретних класів. Це дозволяє клієнту працювати з об'єктами, не знаючи їх конкретної реалізації.

Проблема:
Необхідно створити різні типи об'єктів, які належать до певної категорії (наприклад, для різних операційних систем або стилів інтерфейсів), але потрібно уникнути прямого створення конкретних класів, щоб забезпечити гнучкість і розширюваність.

Рішення:
Створення абстрактної фабрики, яка визначає інтерфейси для створення об'єктів, але дозволяє конкретним фабрикам створювати свої власні реалізації. Клієнт працює з абстрактними фабриками, що дозволяє йому створювати об'єкти певної сім'ї без знання їх конкретної реалізації.

Приклад:
У програмуванні для різних операційних систем можна створити окремі фабрики для Windows і Linux, кожна з яких буде створювати інтерфейси і компоненти, специфічні для цієї системи.

Переваги:

- Покращує розширюваність системи.
- Знижує залежності між конкретними класами.

- Дозволяє легко додавати нові сімейства об'єктів без зміни існуючого коду.

Недоліки:

- Може збільшити складність коду через додаткові класи та інтерфейси.
- Може привести до надмірної абстракції, яка ускладнює розуміння коду.

Шаблон "Factory Method" (Фабричний метод)

Призначення:
Шаблон "Factory Method" дозволяє створювати об'єкти без вказівки їх конкретного класу. Цей шаблон дозволяє підкласи змінювати тип створюваних об'єктів.

Проблема:
Необхідно створити об'єкти, але точний тип об'єкта не може бути визначений до моменту виконання програми, і створення різних об'єктів має бути централізовано контрольованим.

Рішення:
Створення абстрактного класу з фабричним методом, який делегує створення конкретного об'єкта підкласам. Кожен підклас реалізує фабричний метод, що відповідає за створення конкретних об'єктів.

Приклад:
У бібліотеці для обробки зображень можна створити фабричний метод для створення різних типів фільтрів (наприклад, фільтри для чорно-білих або кольорових зображень), який буде визначати тип фільтра залежно від контексту.

Переваги:

- Спрощує код клієнта, оскільки він не потребує знань про точні типи об'єктів.
- Легко додаються нові типи об'єктів, не змінюючи існуючий код.

Недоліки:

- Може призвести до створення великої кількості підкласів.
- Вимагає підтримки додаткової абстракції, що може ускладнити код.

Шаблон "Memento" (Мементо)

Призначення:
Шаблон "Memento" дозволяє зберігати та відновлювати стан об'єкта без порушення інкапсуляції. Це корисно для реалізації функцій "відміни" в програмах.

Проблема:
Необхідно зберігати стан об'єкта для подальшого відновлення без прямого доступу до його внутрішніх даних.

Рішення:
Мементо дозволяє зберігати знімок внутрішнього стану об'єкта в окремому об'єкті (мементо) і пізніше відновлювати цей стан, не порушуючи принцип інкапсуляції.

Приклад:
У текстовому редакторі можна зберігати історію змін і використовувати мементо для реалізації функції "відміна", щоб повернутися до попереднього стану документа.

Переваги:

- Забезпечує безпечне збереження і відновлення стану об'єкта.
- Покращує гнучкість у реалізації функцій, таких як "відміна" та "повтор".

Недоліки:

- Зберігання стану може займати багато пам'яті, якщо об'єкти мають великий стан.
- Ускладнює код через необхідність створення мементо для кожного об'єкта.

Шаблон "Observer" (Спостерігач)

Призначення:
Шаблон "Observer" дозволяє створювати залежність один до багатьох, де зміни в одному об'єкті автоматично сповіщають усіх його спостерігачів.

Проблема:
Необхідно оновлювати кілька об'єктів одночасно, коли змінюється стан одного з них, без безпосереднього зв'язку між ними.

Рішення:
Створення спостерігача, який підписується на зміни в іншому об'єкті. Коли об'єкт змінюється, спостерігачі отримують повідомлення про ці зміни.

Приклад:
В інтерфейсі користувача може бути використаний шаблон "Observer", щоб оновлювати різні елементи інтерфейсу, коли змінюється дані, наприклад, оновлення балансу користувача в реальному часі.

Переваги:

- Спрощує управління залежностями між об'єктами.
- Покращує масштабованість, оскільки нові спостерігачі можуть бути додані без зміни коду спостережуваного об'єкта.

Недоліки:

- Може призвести до надмірної кількості оновлень, якщо спостерігачів багато.
- Ускладнює тестування через велику кількість взаємодій.

Шаблон "Decorator" (Декоратор)

Призначення:
Шаблон "Decorator" дозволяє динамічно додавати нову поведінку об'єкту, не змінюючи його структуру. Це дає можливість додавати нові можливості об'єкта в процесі виконання.

Проблема:
Необхідно розширювати функціональність об'єкта, але без створення великої кількості підкласів і порушення принципу відкритості/закритості.

Рішення:
Створення декоратора, який обгортає існуючий об'єкт і додає нову поведінку. Декоратори можуть бути складені, що дозволяє динамічно комбінувати різні додаткові функціональності.

Приклад:
У графічному редакторі можна додавати нові фільтри або ефекти до зображень, використовуючи декоратори для розширення базових можливостей без зміни існуючого коду.

Переваги:

- Дозволяє гнучко додавати нову функціональність.

- Спрощує модифікацію об'єктів без змін у їхній базовій реалізації.

Недоліки:

- Може призвести до надмірної кількості декораторів, що ускладнює розуміння системи.
- Вимагає додаткових витрат на виконання через обгортання об'єктів.

Реалізувати не менше 3-х класів відповідно до обраної теми

Структура класів

Структура проекту з реалізованими класами зображена на рисунку 1

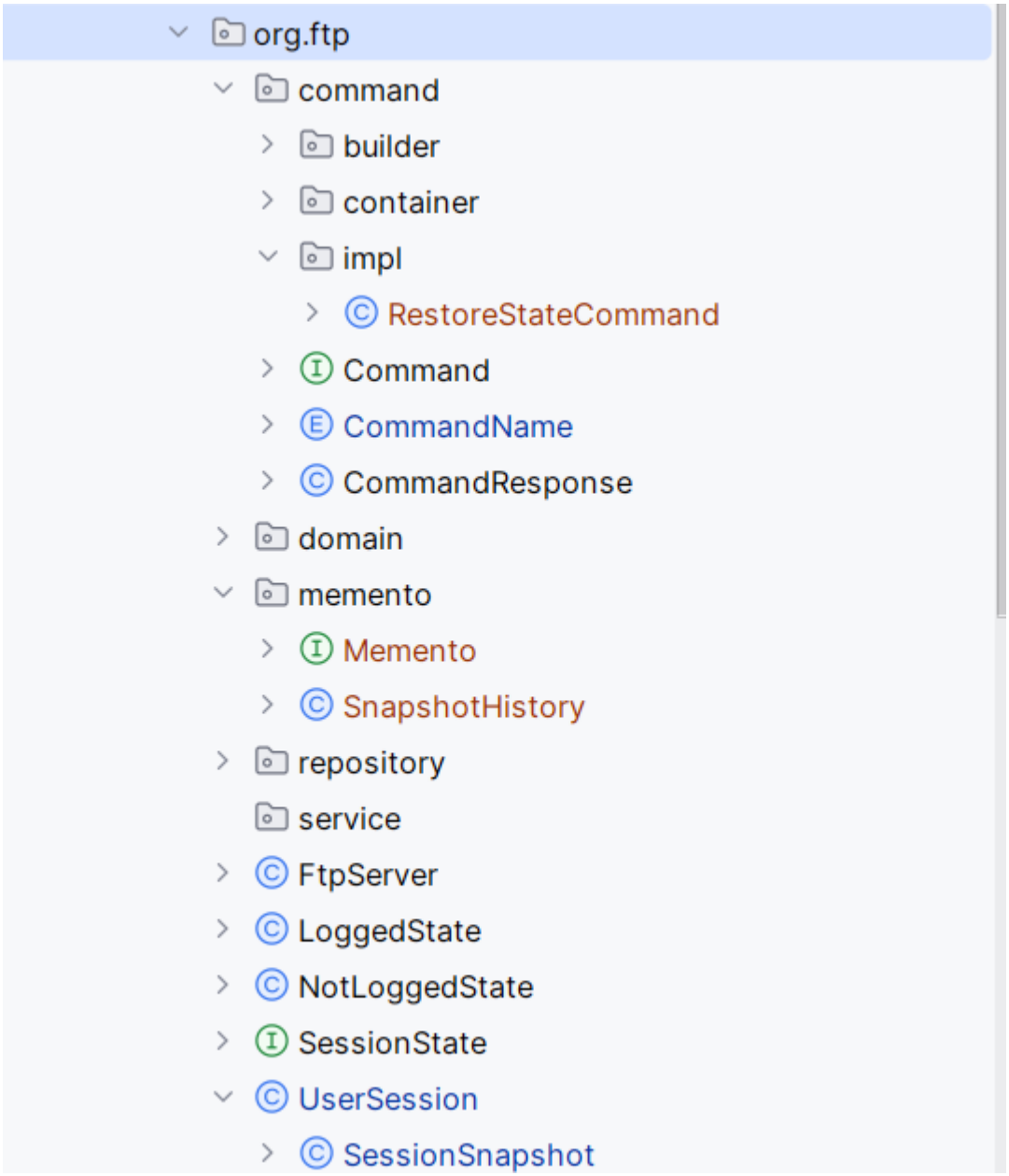

Рисунок 1 – Структура реалізованих класів

Опис класів

Memento (Інтерфейс)

Інтерфейс Memento визначає методи для отримання стану об’єкта, які будуть збережені у знімку. Він має два методи:

- getCurrDirectory(): повертає поточний каталог сесії.
- getState(): повертає стан сесії.

Цей інтерфейс визначає абстракцію для збереження конкретного стану об’єкта, який необхідно зберегти та відновити.

2. SessionSnapshot (Конкретний Memento)

Клас SessionSnapshot є реалізацією інтерфейсу Memento. Він зберігає два ключові параметри стану сесії:

- Поточний каталог користувача (currDirectory).
- Стан сесії (state).

Цей клас є конкретною реалізацією знімка, який створюється при збереженні стану, і його потім можна використовувати для відновлення сесії до певного стану.

3. SnapshotHistory (Caretaker)

Клас SnapshotHistory відповідає за збереження та відновлення знімків стану. Це роль **Caretaker** в патерні Memento, який займається зберіганням знімків у структурі даних (у даному випадку, у стеці) і може отримати останній знімок за допомогою методу restoreState().

Основні методи:

- saveSnapshot(Memento memento): додає новий знімок до історії.
- restoreState(): відновлює стан з останнього знімка, якщо такий є.

4. UserSession (Originator)

Клас UserSession є основним класом, що представляє користувацьку сесію. Він виконує роль **Originator** в патерні Memento, де зберігаються всі важливі дані сесії, такі як поточний каталог та стан сесії.

Основні функції:

- saveSnapshot(): зберігає поточний стан сесії як знімок у об’єкт SnapshotHistory.
- restoreState(): відновлює стан сесії з останнього знімка.

5. Command (Інтерфейс для команд)

Інтерфейс Command визначає загальний контракт для всіх команд, які можуть бути виконані в рамках сесії. Це абстракція для виконання різних операцій над сесією користувача.

Основний метод:

- execute(UserSession session, String[] args): виконання команди, що працює з сесією.

6. RestoreStateCommand (Команда для відновлення стану)

Клас RestoreStateCommand реалізує інтерфейс Command і виконує команду для відновлення стану сесії. При виклику цієї команди метод restoreState() класу UserSession відновлює попередній стан сесії з останнього знімка.

Команда відповідає кодом успішної відповіді:

- Код 200, повідомлення "State restored successfully" — повідомляється, що стан було успішно відновлено.

Реалізувати один з розглянутих шаблонів за обраною темою

Опис шаблону

Шаблон Memento дозволяє зберігати і відновлювати стан об’єкта без необхідності розкривати його внутрішню структуру. Це дозволяє зберігати історію змін об’єкта і потім відновлювати його до попереднього стану, що є корисним, наприклад, для операцій скасування або відновлення стану після помилок.

Опис основних компонентів шаблону Memento в контексті FTP-сервера

1. **Memento (Інтерфейс)**:
    - Це інтерфейс, який визначає методи для отримання стану об’єкта, який потрібно зберігати.
    - Метою цього інтерфейсу є створення знімків стану об’єкта, які можна пізніше використовувати для відновлення цього стану.
2. **SessionSnapshot (Конкретна реалізація Memento)**:

- Клас SessionSnapshot є реалізацією інтерфейсу Memento. Він містить конкретні дані, такі як поточний каталог і стан сесії.
- Це об’єкт, який зберігає конкретний стан сесії FTP-сервера (наприклад, поточний каталог або інші налаштування, важливі для збереження стану сесії).

3. **SnapshotHistory (Caretaker)**:
   - Клас SnapshotHistory управляє історією знімків. Він зберігає всі знімки стану і дозволяє відновлювати останній збережений стан.
   - Цей клас дозволяє додавати нові знімки та відновлювати стан об’єкта з останнього знімка.
4. **UserSession (Originator)**:
   - Клас UserSession зберігає поточний стан сесії користувача, зокрема поточний каталог і стан сесії.
   - Він створює знімки свого стану та може відновлювати свій стан із збережених знімків, що є важливим для управління сесіями користувачів FTP-сервера.
5. **RestoreStateCommand (Команда для відновлення стану)**:
   - Ця команда дозволяє відновити стан сесії користувача на основі збереженого знімка. Команда викликає метод restoreState() у класі UserSession для відновлення стану.
   - Це дає можливість користувачам скасувати незаплановані зміни або помилки і відновити роботу з попереднього стану.

Проблема, яку вирішує шаблон Memento:

До використання шаблону Memento, керування станом об’єкта FTP-сесії могло б бути складним і заплутаним, особливо в ситуаціях, коли потрібно зберігати і відновлювати кілька різних конфігурацій або налаштувань сесії. Замість того, щоб зберігати великі набори параметрів або використовувати складні механізми для збереження і відновлення стану, можна просто створювати знімки стану і зберігати їх для подальшого використання.

Діаграма класів

Діаграма класів, які реалізовують паттерн Memento зображена на рисунку 2

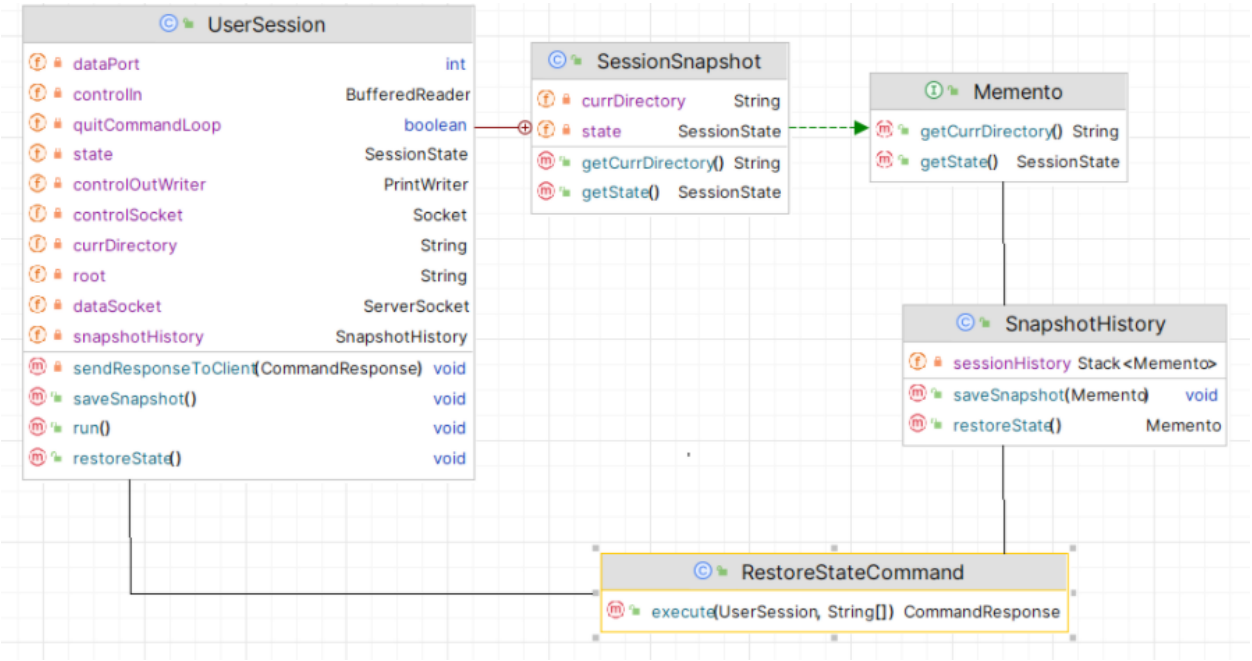

Рисунок 2 – реалізація паттерну Memento

Висновки та код

Код можна знайти за посиланням - https://github.com/B1lok/trpz

Висновок: У ході виконання лабораторної роботи було розглянуто і реалізовано шаблон проектування Memento для управління станом FTP-сесії. Шаблон Memento дозволив ефективно зберігати та відновлювати стан об’єкта без необхідності розкриття його внутрішньої структури. Це значно спростило процес відновлення попередніх станів сесії, що є корисним при необхідності скасування змін або відновлення налаштувань після помилок. Завдяки Memento вдалося реалізувати збереження знімків стану сесії, що зменшило складність коду і дозволило централізовано керувати історією змін без зайвих умовних операторів. У результаті, шаблон Memento підвищив зручність і гнучкість роботи з різними станами FTP-сесії, забезпечивши легкість відновлення попередніх налаштувань за потреби.