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 №**4**
з предмету «Технології розроблення програмного забезпечення»
на тему «**Шаблони “Singleton”, “Iterator”, “Proxy”, “State”, “Strategy”**»

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 та їхньої взаємодії 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 проектування (design pattern) — це повторюване рішення для типових проблем проєктування в об'єктно-орієнтованому програмуванні. Шаблони проектування надають готову структуру або набір кроків для вирішення задач, які постають перед програмістами під час розробки програмного забезпечення. Вони не є готовим кодом, а скоріше орієнтиром, який можна адаптувати до конкретних потреб проєкту. Завдяки шаблонам проектування розробники можуть створювати більш гнучкі та підтримувані рішення, уникати поширених помилок і робити код зрозумілішим.

Шаблони проектування допомагають розробникам:

Покращити якість коду: Вони надають перевірені рішення для типових проблем, що дозволяє уникати помилок.

Зробити код легшим для підтримки та масштабування: Правильно вибраний шаблон структурує код, роблячи його простішим для змін і доповнень.

Забезпечити гнучкість: Використання шаблонів дозволяє створювати розширюваний код, який легко адаптується до нових вимог.

Стандартизувати підхід: Це полегшує командну роботу, оскільки розробники використовують загальноприйняті способи вирішення задач.

Чим відрізняється шаблон «стратегія» від «стану»?

Обидва шаблони належать до поведінкових і використовують принципи поліморфізму, але мають різні цілі:

Шаблон «стратегія» (Strategy) дозволяє вибирати алгоритм під час виконання програми. Замість того щоб використовувати один метод або набір логіки, шаблон Strategy дозволяє використовувати різні варіанти реалізації, що

4можуть динамічно змінюватися в залежності від умов. Прикладом може бути різне обчислення знижок для різних типів клієнтів.

Шаблон «стан» (State) змінює поведінку об'єкта залежно від його поточного стану. Він корисний, коли об'єкт має обмежену кількість станів, і кожен з них змінює поведінку цього об'єкта. На відміну від «стратегії», «стан» передбачає, що об'єкт змінює свою внутрішню логіку залежно від стану.

В чому полягає ідея шаблона «одинак»? Чому його вважають «анти- шаблоном»? Шаблон «одинак» (Singleton) гарантує, що певний клас матиме лише один екземпляр, який буде доступний з будь-якої точки програми. Це корисно, коли необхідно мати єдиний доступний ресурс, наприклад, конфігурацію програми або з'єднання з базою даних.

Проте, «одинак» часто вважається «анти-шаблоном» через такі проблеми: Погіршення тестованості: Singleton ускладнює написання юніт-тестів,

оскільки його складно ізолювати та підмінити в тестах.

Погіршення гнучкості: Singleton фіксує клас у стані одного екземпляра, що може обмежити можливість його розширення або зміни.

Прихована залежність: Замість того щоб явно передавати залежність у методи чи класи, Singleton стає глобальною залежністю, що порушує принципи чистого коду.

Шаблон «проксі» (Proxy) забезпечує інтерфейс для доступу до іншого об'єкта, часто додаючи додаткову поведінку або контролюючи доступ. Проксі- клас виступає посередником, який обмежує або розширює функціональність оригінального об'єкта. «Проксі» використовують у наступних ситуаціях:

Відкладене створення об'єкта: Proxy створює об'єкт лише тоді, коли він дійсно потрібен (наприклад, завантаження важких ресурсів).

5Контроль доступу: Proxy може перевіряти права доступу перед викликом методів об'єкта.

Логування або кешування: Proxy дозволяє додавати логування, кешування або інші операції, не змінюючи логіку оригінального об'єкта.

Мережеві операції: Proxy може представляти об'єкти, які фізично знаходяться в інших машинах або мережах, дозволяючи їм взаємодіяти через мережу.

Хід роботи::

Реалізація паттерну:

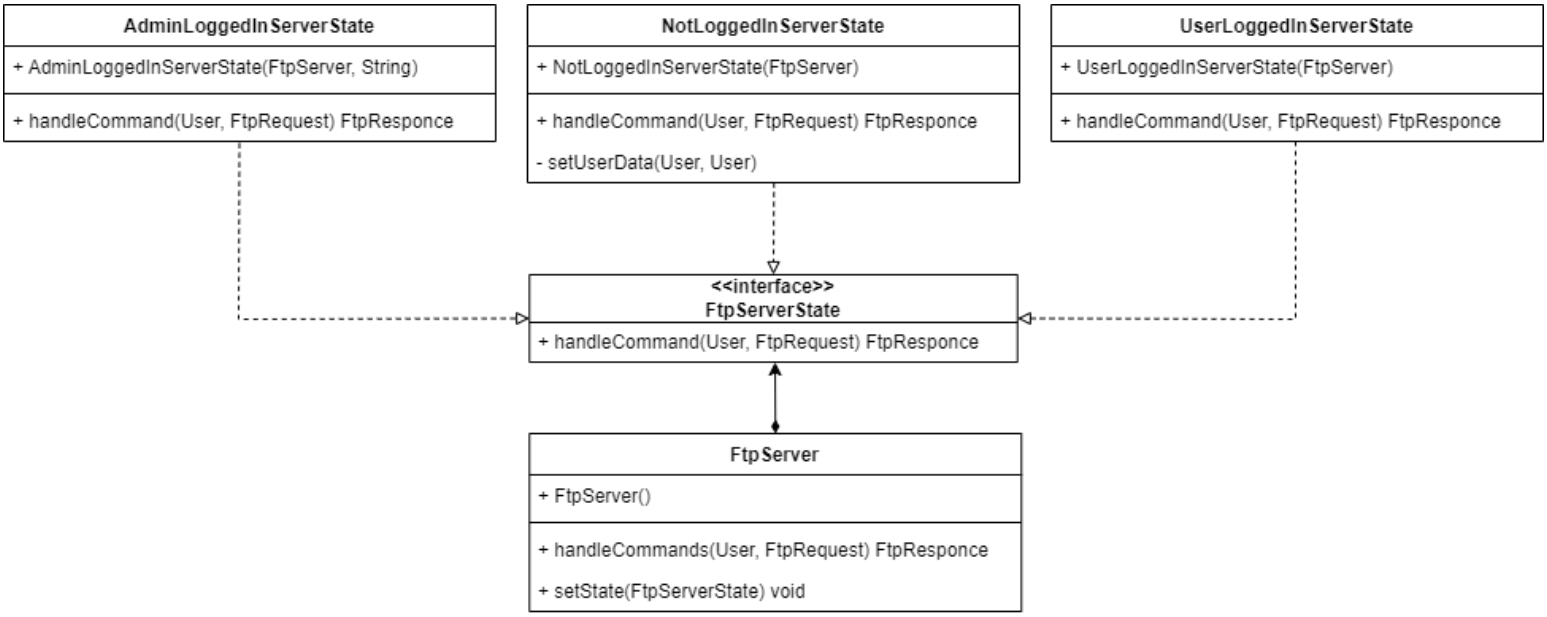

Рисунок 1 – реалізація паттерну State

Опис реалізації паттерну:

Реалізація паттерну полягає в тому, що логіка, яка залежить від стану серверу виноситься в конкретний клас цього стану, що реалізовує загальний для всіх станів інтерфейс FtpServerState. Шаблон **State** дозволяє **FtpServer** динамічно змінювати свою поведінку, базуючись на поточному стані. У цьому випадку сервер змінює поведінку залежно від того, авторизований користувач чи ні, і чи має він права адміністратора.

Опис класів та інтерфейсів:

FtpServerState (інтерфейс)

Інтерфейс, що представляє стан FTP-сервера. Містить метод: handleCommand(User, FtpRequest) FtpResponse — метод для обробки команди, отриманої від користувача. Цей метод повинен бути реалізований у кожному конкретному стані.

AdminLoggedInServerState (конкретний стан)

Реалізує стан, у якому адміністратор авторизований на сервері.

Методи: AdminLoggedInServerState(FtpServer, String) — конструктор стану адміністратора. handleCommand(User, FtpRequest) FtpResponse — обробка команди від адміністратора.

UserLoggedInServerState (конкретний стан)

Реалізує стан, у якому звичайний користувач авторизований на сервері.

Методи: UserLoggedInServerState(FtpServer) — конструктор стану користувача. handleCommand(User, FtpRequest) FtpResponse — обробка команди від авторизованого користувача.

NotLoggedInServerState (конкретний стан)

Реалізує стан, у якому користувач не авторизований.

Методи: NotLoggedInServerState(FtpServer) — конструктор стану для неавторизованого користувача. handleCommand(User, FtpRequest) FtpResponse — обробка команди від неавторизованого користувача. setUserData(User, User) — приватний метод для встановлення даних користувача.

FtpServer

Клас, що представляє FTP-сервер. Він зберігає поточний стан і делегує обробку команд у поточному стану.

Методи: FtpServer() — конструктор сервера. handleCommands(User, FtpRequest) FtpResponse — метод для обробки команд, який делегує виконання поточному стану. setState(FtpServerState) void — метод для зміни стану сервера.

Логіка роботи:

Залежно від стану (AdminLoggedInServerState, UserLoggedInServerState або NotLoggedInServerState), метод handleCommand обробляє команди по-різному.

FtpServer може змінювати стан через метод setState, що дозволяє системі змінювати поведінку залежно від стану користувача (адміністратор, користувач або неавторизований).

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 State. Існуючий код було оновлено для використання відповідного паттерну.