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 дисципліни «Технології розроблення програмного забезпечення»
Тема: **«Шаблони SINGLETON, ITERATOR, PROXY, STATE, STRATEGY»**
HTTP-сервер

Виконав:
Студент групи ІА-22
Білецький С. В.

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:
Шаблони «singleton», «iterator», «proxy», «state», «strategy»

Мета:
Навчитися використовувати шаблони проєктування «Singleton», «Iterator», «Proxy», «State» та «Strategy» для розробки програмного забезпечення. Ознайомитися з принципами їхньої роботи та особливостями застосування в різних ситуаціях. Дослідити, як ці патерни допомагають створювати гнучкі та масштабовані програмні рішення.

Завдання:
HTTP-сервер. Сервер повинен мати можливість розпізнавати вхідні запити і формувати коректні відповіді (згідно протоколу HTTP), надавати сторінки chtml (html сторінки з додаванням найпростіших C# конструкцій на розсуд студента), вести статистику вхідних запитів, обробку запитів у багатопотоковому/подієвому режимах. **(State)**

Хід роботи

1. Реалізувати не менше 3-х класів відповідно до обраної теми

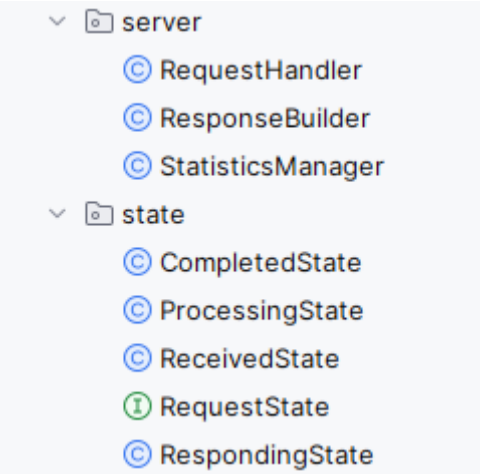

Рис. 1 — Структура класів

У ході виконання лабораторної роботи було реалізовано функціонал 6 класів, що зображені на рис. 1 (певні класи реалізовані частково та мають потенційні можливості для доповнення у зв’язку з тим, що система ще не повністю реалізована). Детальний опис кожного з класів:

1. **RequestState (Інтерфейс).** Цей інтерфейс визначає спільну поведінку для всіх класів станів. Всі стани повинні реалізувати метод handleRequest(),

який описує, як обробляти запит у відповідному стані. Це дозволяє змінювати поведінку обробки запиту залежно від поточного стану об'єкта.

2. **ReceivedState.** Цей клас представляє стан, коли сервер тільки отримав запит і ще не почав його обробляти. У цьому стані сервер лише готується до подальших дій. Це початковий стан для кожного запиту. Після того, як сервер визначив, що запит отримано, відразу ж відбувається перехід до наступного стану для обробки.
3. **ProcessingState.** Цей клас відповідає за обробку запиту після того, як він був отриманий. У цьому стані сервер займається логікою, необхідною для обробки запиту (наприклад, отримання необхідних даних, виконання бізнес-логіки). Цей стан можна використовувати для інтеграції з іншими модулями, такими як база даних або логіка генерації контенту. Перехід до наступного стану здійснюється лише після завершення обробки запиту.
4. **RespondingState.** Цей клас відповідає за формування відповіді для запиту після його обробки. Тут сервер генерує відповідь і готується до відправлення її клієнту. Тут формується і перевіряється сама відповідь, що буде надіслана клієнту. Відповідь може включати інші дані, такі як статистика чи помилки, які потрібно врахувати у відповіді.
5. **CompletedState.** Цей клас позначає, що сервер завершив обробку запиту. У цьому стані сервер записує статистику про запит і його обробку, щоб мати змогу потім аналізувати та виводити інформацію про ефективність роботи. Цей стан служить для завершення обробки запиту та збереження результатів. Він визначає завершення циклу обробки запиту, і після цього сервер готовий до обробки нового запиту.
6. **RequestHandler.** Клас RequestHandler виступає контекстом для патерну State і координує зміну станів під час обробки запиту. Він зберігає поточний стан та викликає відповідні дії для обробки запиту. Також відповідає за синхронізацію всіх станів і гарантує, що запит буде оброблений поетапно, відповідно до визначених станів. Завдяки цьому класу можна легко додавати нові стани для обробки запитів або змінювати існуючі стани без зміни основної логіки роботи сервера.

2. Реалізувати один з розглянутих шаблонів за обраною темою

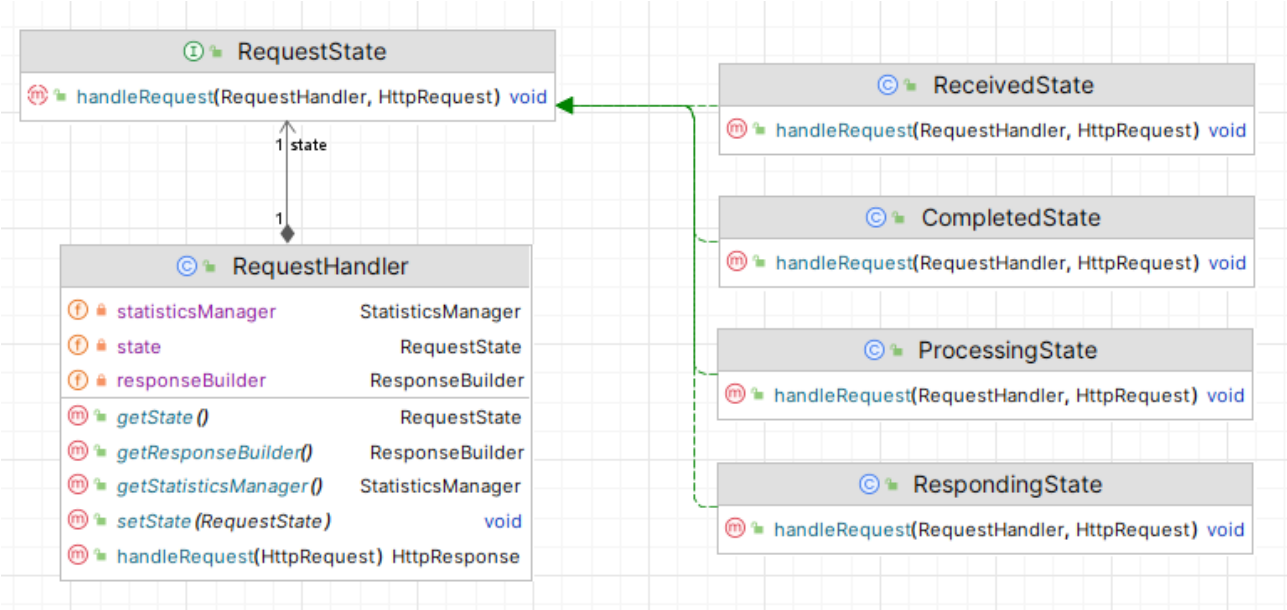

Рис. 2 — Реалізація патерну

У нашому проекті HTTP-сервера патерн State був реалізований для управління різними етапами обробки запиту, дозволяючи змінювати поведінку сервера в залежності від поточного стану запиту. Кожен етап обробки був інкапсульований в окремий клас стану, що надає серверу чітку, покрокову структуру для обробки кожного запиту.

Реалізація

У нашому проекті HTTP-сервера патерн State був реалізований для управління різними етапами обробки запиту, дозволяючи змінювати поведінку сервера в залежності від поточного стану запиту. Кожен етап обробки був інкапсульований в окремий клас стану, що надає серверу чітку, покрокову структуру для обробки кожного запиту.

Клас RequestHandler діє як контекст для патерну State, зберігаючи поточний стан і викликаючи відповідні дії для кожного етапу обробки запиту. Цей клас відповідає за зміну стану на кожному етапі запиту, що забезпечує послідовну обробку від отримання запиту до фіксації статистики про нього.

Переваги використання патерну State

1. Зручність та гнучкість у додаванні нових станів: Кожен стан обробки запиту є окремим класом, що реалізує спільний інтерфейс, і це дозволяє легко додавати нові стани або змінювати існуючі. Наприклад, можна

додати стан для обробки помилок або розширити логіку обробки відповідей без зміни основного коду обробника запитів.
2. Зрозуміла логіка обробки запитів: Використання патерну State розділяє обробку запиту на чіткі кроки, що робить структуру логіки обробки зрозумілішою та легшою для підтримки. Код стає більш модульним, а кожен клас стану має лише одну відповідальність — обробку свого етапу.
3. Поліпшення підтримки багатопотокової обробки: Завдяки чіткому розподілу на етапи обробки, патерн State полегшує підтримку багатопотокової обробки запитів, адже кожен запит може перебувати у своєму стані незалежно від інших. Це дозволяє обробляти декілька запитів одночасно, не заважаючи один одному.
4. Полегшення тестування: Кожен стан має окремий клас, який можна тестувати незалежно від інших, що значно спрощує процес тестування серверної логіки. Наприклад, можна окремо перевірити, як сервер поводиться у стані обробки, незалежно від того, як він обробляє запит у стані відповіді.
5. Гнучкість в управлінні потоками запиту: Патерн State дозволяє легко перемикати поведінку сервера під час обробки запиту. Замість того, щоб прописувати численні умови для перевірки стану запиту, ми просто встановлюємо новий стан і передаємо управління відповідному класу, що полегшує обробку складних умов.

Таким чином, патерн State в нашому проекті допомагає структурувати обробку запитів, забезпечуючи модульність, гнучкість і розширюваність. Ця архітектура не тільки покращує читабельність та підтримку коду, але й дозволяє ефективно обробляти багатопотокові запити в рамках багатозадачного середовища, що є ключовим для високопродуктивних HTTP-серверів.

Висновки:

У ході лабораторної роботи ми ознайомилися з різними шаблонами проєктування, зокрема з State, який застосували для обробки HTTP-запитів. Реалізація патерну State допомогла структурувати процес обробки на етапи, підвищивши модульність і гнучкість коду. Це полегшило підтримку та тестування системи, продемонструвавши ефективність State для побудови керованих і масштабованих серверних застосунків.