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 №7
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«ШАБЛОНИ «Mediator», «Facade», «Bridge», «Template Method» »**
HTTP-сервер

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

Перевірив:
Мягкий М.

Київ-2024

Зміст

Тема:
Шаблони «Mediator», «Facade», «Bridge», «Template Method»

Мета:
Навчитися застосовувати шаблони проєктування «Mediator», «Facade», «Bridge», «Template Method» у процесі розробки програмного забезпечення. Ознайомитися з їхньою концепцією, особливостями реалізації та можливостями використання для створення гнучких і масштабованих рішень.

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. **(Mediator)**

Хід роботи

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

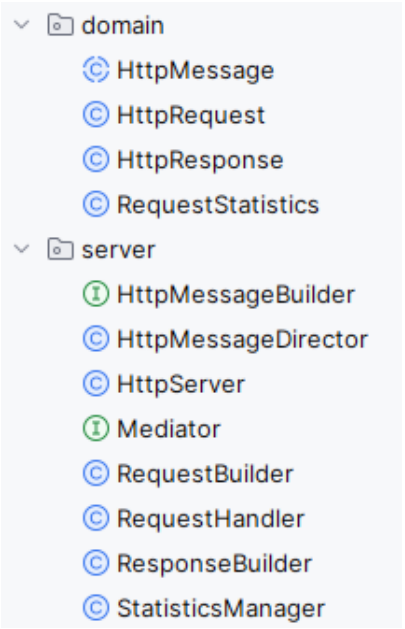

Рис. 1 — Структура класів

У ході виконання лабораторної роботи було реалізовано функціонал 5 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. **HttpServer.** Цей клас є основним компонентом, що відповідає за запуск і зупинку сервера. Він реалізує інтерфейс Mediator і управляє іншими класами системи. Зокрема, він працює з RequestHandler для обробки запитів та з StatisticsManager для збору статистики.
2. **RequestHandler.** Клас, що займається обробкою HTTP-запитів. Він відповідає за прийом запиту, формування відповіді і передачу статистичних даних до StatisticsManager через Mediator. У ньому також реалізовано використання патерну "State", що дозволяє змінювати поведінку обробки запитів в залежності від їхнього стану.
3. **StatisticsManager.** Клас для збору і зберігання статистики про HTTP-запити. Він зберігає дані, такі як час запиту, метод, статус відповіді, шлях та час обробки. StatisticsManager взаємодіє з Mediator для оновлення статистики, і може передавати інформацію до інших компонентів за необхідністю.
4. **Mediator.** Інтерфейс, який визначає взаємодію між різними компонентами системи, забезпечуючи слабку зв'язність. Завдяки цьому інтерфейсу, HttpServer взаємодіє з іншими класами без необхідності прямого посилання на конкретні реалізації, що дозволяє зберігати гнучкість і розширюваність системи.
5. **RequestStatistics.** Клас для зберігання статистичних даних про кожний HTTP-запит. Він включає такі атрибути, як метод запиту, статус відповіді, час обробки та шлях запиту. Ці дані використовуються для подальшого аналізу і зберігаються у StatisticsManager.

2. Реалізувати один з розглянутих шаблонів за обраною темою

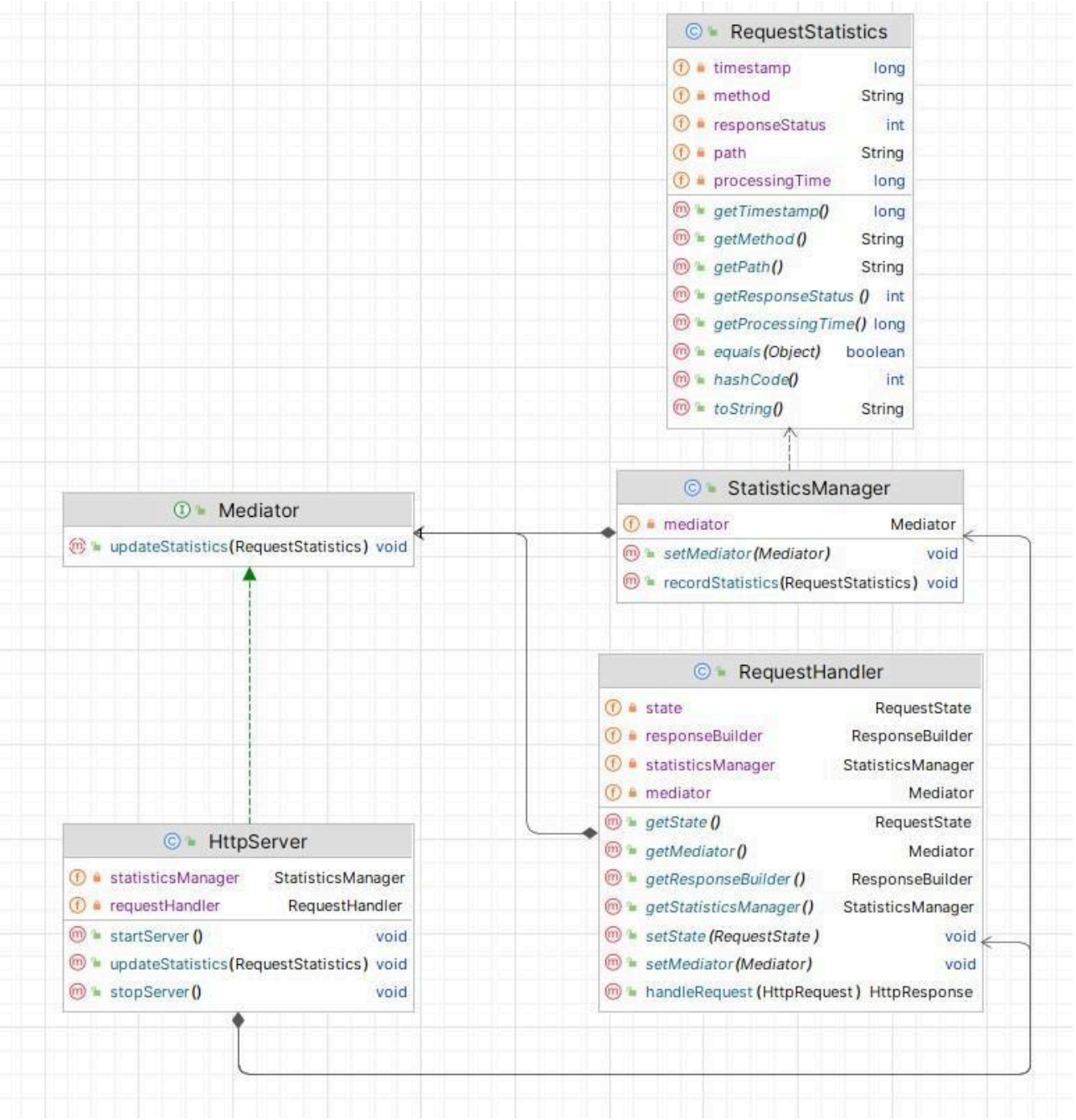

Рис. 2 — Реалізація патерну

У контексті проекту HTTP-сервер, патерн Mediator використовується для організації комунікації між різними компонентами сервера без безпосереднього зв'язку між ними. Це дозволяє розділити функціональність сервера на окремі модулі, кожен з яких виконує свою задачу, при цьому взаємодія між ними здійснюється через посередника.

У складних системах, де кілька компонентів повинні взаємодіяти, може виникнути ситуація, коли компоненти обмінюються даними безпосередньо один з одним. Це може призвести до складних і заплутаних залежностей між класами, що ускладнює підтримку та тестування коду. У нашому випадку, клас HttpServer міг би безпосередньо взаємодіяти з RequestHandler та StatisticsManager, що спричинило б тісну зв'язність і ускладнило б розширення або модифікацію системи в майбутньому.

Реалізація

HttpServer реалізує інтерфейс Mediator і виступає в ролі посередника між компонентами. Він отримує та обробляє запити через RequestHandler, а також управляє оновленням статистики через StatisticsManager.
RequestHandler та StatisticsManager не взаємодіють безпосередньо між собою, а замість цього передають необхідні дані через HttpServer. Таким чином, коли RequestHandler обробляє запит, він може передати статистичні дані в HttpServer, який через Mediator оновить статистику в StatisticsManager.
Взаємодія через Mediator дозволяє мінімізувати кількість залежностей між класами, спрощуючи підтримку та розширення сервера, оскільки зміни в одному класі не впливають безпосередньо на інші компоненти.

Переваги використання Mediator:

1. Зниження зв'язності: Компоненти не мають безпосередніх залежностей один від одного, що дозволяє легше змінювати або замінювати окремі частини системи без порушення роботи інших.
2. Масштабованість: Коли система розширюється, наприклад, додаються нові обробники запитів або статистичні модулі, можна легко інтегрувати їх, не змінюючи існуючі класи.
3. Централізоване управління: Всі взаємодії між компонентами проходять через посередника, що дозволяє централізовано керувати процесами і зберігати єдину точку контролю.
4. Зручність тестування: Завдяки зниженій зв'язності, тестування окремих компонентів стає простішим, оскільки можна замінити взаємодії через Mediator на фіктивні або замінні сервіси для перевірки їхньої роботи.

```java
package org.example.server;

import org.example.domain.HttpRequest;
import org.example.domain.RequestStatistics;

public interface Mediator { 3 usages 1 implementation
    void updateStatistics(RequestStatistics statistics); 1 usage 1 implementation
}
```

Рис. 3 — Реалізація інтерфейсу Mediator

Короткий опис: Mediator виступає як посередник для управління або оновлення об'єкта RequestStatistics, що містить дані про статистику запитів.

Метод void updateStatistics(RequestStatistics statistics) приймає об'єкт типу RequestStatistics і оновлює його відповідно до заданих умов.

```java
package org.example.server;

import lombok.Setter;
import org.example.domain.HttpRequest;
import org.example.domain.RequestStatistics;

public class StatisticsManager { 3 usages
    @Setter
    private Mediator mediator;
    public void recordStatistics(RequestStatistics statistics) { 1 usage
        // TODO
    }
}
```

Рис. 4 — Реалізація StatisticsManager

Короткий опис: StatisticsManager слугує як компонент для запису статистичних даних запитів (типу RequestStatistics). Використовує об'єкт Mediator для передачі або оновлення статистики.

```java
package org.example.server;

import org.example.domain.HttpRequest;
import org.example.domain.RequestStatistics;

public class HttpServer implements Mediator {  no usages
    private RequestHandler requestHandler;  no usages
    private StatisticsManager statisticsManager;  1 usage

    @Override  1 usage
    public void updateStatistics(RequestStatistics statistics) { statisticsManager.recordStatistics(statistics); }

    public void startServer() { System.out.println("Server started. Listening for requests..."); }

    public void stopServer() { System.out.println("Server stopped."); }
}
```

Рис. 5 — Реалізація HttpServer

Короткий опис: Клас HttpServer реалізує інтерфейс Mediator і відповідає за обробку HTTP-запитів, управління статистикою запитів і керування життєвим циклом сервера. Реалізує патерн Mediator, координуючи взаємодію між компонентами StatisticsManager та іншими частинами системи.

Сервер є точкою входу для керування HTTP-запитами та оновленням статистики.

Посилання на код проекту: github

Висновки:

У рамках лабораторної роботи ми застосували патерн Mediator для централізованого управління взаємодією між компонентами сервера. Це дозволило зменшити залежності між класами, спростивши розширення та підтримку системи. Патерн забезпечив гнучкість і масштабованість, централізуючи логіку обміну даними між компонентами, що полегшило розробку та налагодження сервера