



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: «Патерни проєктування»
«2. HTTP-сервер»

Виконала:
студентка групи - IA-34
Мартинюк Т.В.

Перевірив:
Мягкий Михайло
Юрійович

Київ 2025

Тема: Патерни проєктування.

Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

Тема проєкту: HTTP-сервер (state, builder, factory method, mediator, composite, p2p) 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.

Теоретичні відомості:

Mediator

Патерн Mediator служить «диригентом», який координує взаємодію між різними компонентами системи без їх прямого зв'язку один з одним. У випадку обробки HTTP-запитів Mediator може керувати послідовністю дій: парсингом, викликом обробників, збереженням даних у різних репозиторіях та логуванням, забезпечуючи централізоване управління і зменшуючи залежності між компонентами. Це робить систему більш гнучкою та легко модифікованою.

Facade

Патерн Facade створює спрощений інтерфейс для складної підсистеми. Замість того, щоб окремі частини системи (наприклад, логіка збереження в різні репозиторії та логування) взаємодіяли безпосередньо, Facade пропонує один інтерфейс для виконання усіх цих дій. Це дозволяє приховати внутрішню складність і зробити використання підсистеми простішим і зрозумілішим для клієнтського коду.

Bridge

Патерн Bridge розділяє абстракцію і реалізацію, дозволяючи їм змінюватися незалежно. У контексті системи обробки HTTP-запитів це може бути корисним, якщо потрібно підтримувати різні способи обробки запитів або

збереження даних, не змінюючи інтерфейс, який використовує бізнес-логіка. Bridge дозволяє легко додавати нові варіанти реалізації без порушення існуючого коду.

Template Method

Патерн Template Method визначає скелет алгоритму у базовому класі, залишаючи окремі кроки для реалізації у підкласах. Це зручно для обробки HTTP-запитів, коли послідовність дій загалом однакова, але окремі етапи (наприклад, специфічна обробка тіла запиту або логування) можуть відрізнятися. Template Method дозволяє уникнути дублювання коду і забезпечує узгодженість у виконанні основного алгоритму.

Завдання:

- Ознайомитись з короткими теоретичними відомостями.
- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
- Реалізувати один з розглянутих шаблонів за обраною темою.
- Реалізувати не менше 3-х класів відповідно до обраної теми.
- Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

Хід роботи:

Проблема: Потрібен клас, який керував би повним життєвим циклом HTTP-запиту: від парсингу та визначення обробника до збереження даних у різних репозиторіях та відправки відповіді клієнту. Раніше цю функцію виконував клас Middleware, який концентрував у собі всю логіку обробки запиту. Це порушувало принцип єдиної відповідальності, робило код залежним від багатьох реалізацій і ускладнювало тестування, підтримку та зміну логіки обробки запитів.

Рішення: Запровадження патерну Mediator через клас RequestMediator централізує управління життєвим циклом запиту. RequestMediator координує послідовність дій (парсинг, обробка маршрутів, збереження даних, логування, відправка відповіді), усуваючи прямі залежності між компонентами. Це спрощує Middleware, який тепер лише надає залежності, та робить код гнучкішим і легшим для тестування.

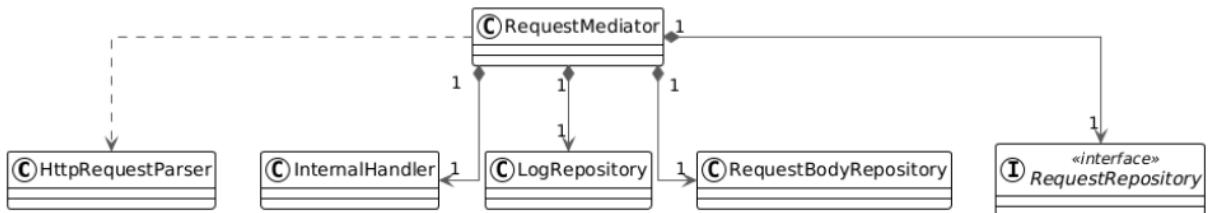


Рисунок 1 – Реалізація патерну проєктування «Mediator»

Висновок:

У ході лабораторної роботи було успішно застосовано патерн Mediator для організації обробки HTTP-запитів. Клас RequestMediator виступає центральним посередником, який повністю керує життєвим циклом запиту, координуючи роботу компонентів: парсинг вхідного потоку, виконання роутингу через InternalHandler, збереження тіла запиту в RequestBodyRepository, збереження метаданих у RequestRepository та логування продуктивності в LogRepository. Кожен компонент виконує лише свою задачу і не залежить напряму від інших, а вся взаємодія відбувається виключно через RequestMediator. Це дозволило усунути порушення принципу єдиної відповідальності, позбутися сильної зв'язаності між класами, спростити тестування та підтримку коду, а також зробити послідовність обробки запиту легко змінюваною. Застосування патерну Mediator значно покращило архітектуру серверу, забезпечило слабку зв'язаність компонентів і продемонструвало практичну цінність цього структурного патерну проєктування в реальному проєкті.

Лістинг коду програми:

```
package server;
import database.repos.LogRepository;
import database.repos.RequestBodyRepository;
import database.repos.RequestRepository;
import database.repos.StatisticsRepository;

public abstract class Middleware {

    protected final RequestRepository requestRepository;
    protected final RequestBodyRepository requestBodyRepository;
    protected final LogRepository logsRepository;
    protected final StatisticsRepository statisticsRepository;

    public Middleware(RequestRepository requestRepository,
                      RequestBodyRepository requestBodyRepository,
                      LogRepository logsRepository,
                      StatisticsRepository statisticsRepository) {

        this.requestRepository = requestRepository;
        this.requestBodyRepository = requestBodyRepository;
        this.logsRepository = logsRepository;
        this.statisticsRepository = statisticsRepository;
    }

    public abstract Handler createHandler();
}
```

```
package server;

import http.HttpRequest;
import http.HttpResponse;
import http.HttpRequestParser;
import database.repos.RequestBodyRepository;
import database.repos.RequestRepository;
import database.repos.LogRepository;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.sql.SQLException;

public class RequestMediator {

    private final RequestRepository requestRepository;
    private final RequestBodyRepository requestBodyRepository;
    private final LogRepository logsRepository;
    private final InternalHandler routeHandler;

    public RequestMediator(
        RequestRepository requestRepository,
        RequestBodyRepository requestBodyRepository,
        LogRepository logsRepository,
        InternalHandler routeHandler) {
```

```

        this.requestRepository = requestRepository;
        this.requestBodyRepository = requestBodyRepository;
        this.logsRepository = logsRepository;
        this.routeHandler = routeHandler;
    }

    public void mediateRequest(InputStream inputStream, OutputStream outputStream)
        throws IOException, SQLException {
        long startTime = System.currentTimeMillis();

        HttpRequestParser parser = new HttpRequestParser();
        HttpRequest request = parser.parse(inputStream);

        HttpResponse response = routeHandler.handle(request);

        long duration = System.currentTimeMillis() - startTime;

        Integer bodyId = null;
        if (request.getBody() != null && !request.getBody().isEmpty()) {
            bodyId = requestBodyRepository.saveBody(request.getBody());
        }

        int requestId = requestRepository.save(request, bodyId);

        logsRepository.log(requestId, response.getStatusCode(), duration);

        response.send(outputStream);
    }
}

```

package server;

```

import http.RouteComponent;
import database.repos.LogRepository;
import database.repos.RequestBodyRepository;
import database.repos.RequestRepository;
import serverConfigs.ConfigHandler;
import database.repos.StatisticsRepository;
import statistics.Logger;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.sql.SQLException;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

```

```
public class Server {

    private boolean running;
    private ServerSocket serverSocket;

    private final Logger logger = new Logger();

    private ThreadPoolExecutor executorService;
    private final ConfigHandler configHandler;
    private RouteComponent routes;
    private Middleware internalMiddleware;
    private RequestRepository requestRepository;
    private RequestBodyRepository requestBodyRepository;
    private LogRepository logsRepository;
    private StatisticsRepository statisticsRepository;

    public Server() {
        this.configHandler = new ConfigHandler();
    }

    public void setRepositories(
        RequestRepository requestRepository,
        RequestBodyRepository requestBodyRepository,
        LogRepository logsRepository,
        StatisticsRepository statisticsRepository) {

        this.requestRepository = requestRepository;
        this.requestBodyRepository = requestBodyRepository;
        this.logsRepository = logsRepository;
        this.statisticsRepository = statisticsRepository;
    }

    public void start() {
        try {
            if (this.routes == null) {
                throw new IllegalStateException("Some routes must be added before starting the server.");
            }

            running = true;
            logger.info("Starting HTTP server... ");

            serverSocket = new ServerSocket(configHandler.getPort());

            executorService = new ThreadPoolExecutor(
                configHandler.getNumberOfThreads(),
                configHandler.getNumberOfThreads(),
                0L, TimeUnit.MILLISECONDS,
                new LinkedBlockingQueue<>()
            );
        }

        logger.info("Http Server started on port " + configHandler.getPort());

        while (running) {
            try {

```

```

        Socket client = serverSocket.accept();
        executorService.submit(() -> handleRequest(client));
    } catch (IOException e) {
        if (running) {
            logger.warning("Error accepting client: " + e.getMessage());
        }
    }
}

} catch (IOException e) {
    logger.severe("Error starting server: " + e.getMessage());
}
}

private void handleRequest(Socket socket) {
    try (InputStream in = socket.getInputStream();
         OutputStream out = socket.getOutputStream()) {

        RequestMediator mediator = new RequestMediator(
            requestRepository,
            requestBodyRepository,
            logsRepository,
            new InternalHandler(routes)
        );

        mediator.mediateRequest(in, out);

    } catch (IOException | SQLException e) {
        logger.severe("Error handling request: " + e.getMessage());
        e.printStackTrace();
    } finally {
        try {
            socket.close();
        } catch (IOException e) {
            logger.warning("Error closing socket: " + e.getMessage());
        }
    }
}

public void setRoutes(RouteComponent component) {
    this.routes = component;
}

public void stop() {
    try {
        logger.info("Stopping HTTP server...");
        serverSocket.close();
        executorService.shutdown();
        logger.info("HTTP server stopped.");
    }
    catch (IOException e) {
        logger.severe("Error stopping HTTP server: " +
                     e.getMessage());
    }
}

```

```

};

public ServerSocket getServerSocket() {
    return serverSocket;
}

public void setServerSocket(ServerSocket serverSocket) {
    this.serverSocket = serverSocket;
}

}

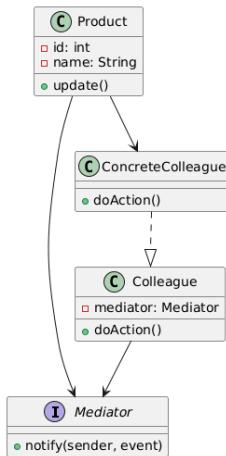
```

Контрольні питання:

1. Яке призначення шаблону «Посередник»?

Організовує взаємодію між об'єктами через центральний об'єкт, зменшуючи прямі залежності між ними.

2. Нарисуйте структуру шаблону «Посередник».



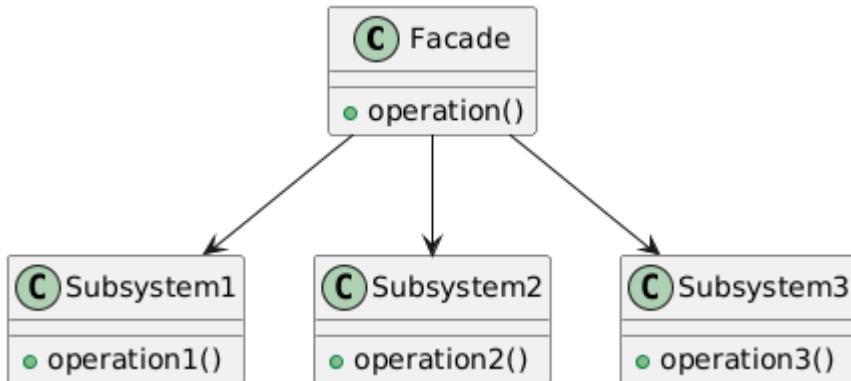
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

- Mediator (інтерфейс) — визначає методи взаємодії.
- ConcreteColleague — конкретна реалізація колеги.
- Colleague — об'єкти, які взаємодіють через медіатора.
- Взаємодія: колеги не знають один про одного, лише про медіатора; медіатор координує їх дії.

4. Яке призначення шаблону «Фасад»?

Створює спрощений інтерфейс для підсистеми, приховуючи внутрішні деталі і забезпечуючи єдиний доступ.

5. Нарисуйте структуру шаблону «Фасад».



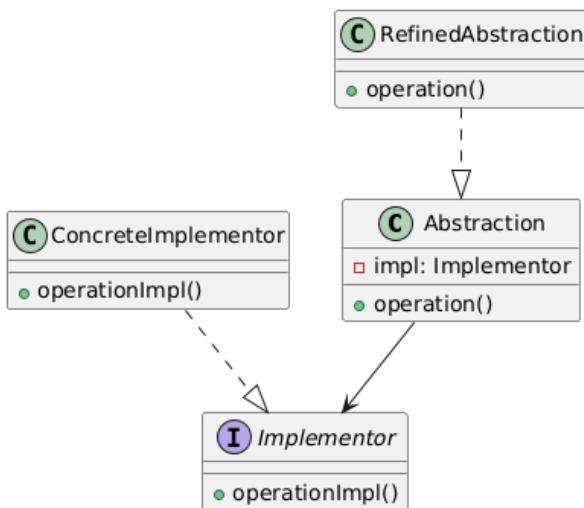
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

- **Facade** – надає спрощений інтерфейс для клієнтів.
- **Subsystem** – внутрішні класи підсистеми.
- **Взаємодія:** клієнт звертається тільки до фасаду, фасад керує підсистемами.

7. Яке призначення шаблону «Міст»?

Розділяє абстракцію і реалізацію, щоб їх можна було змінювати незалежно одна від одної.

8. Нарисуйте структуру шаблону «Міст».



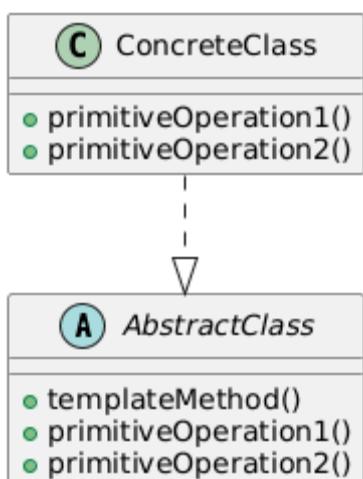
9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

- **Abstraction** – абстрактний інтерфейс.
- **RefinedAbstraction** – розширення абстракції.
- **Implementor** – інтерфейс реалізації.
- **ConcreteImplementor** – конкретна реалізація.
- **Взаємодія:** абстракція делегує роботу реалізації через інтерфейс Implementor; обидві ієрархії незалежні.

10. Яке призначення шаблону «Шаблонний метод»?

Визначає загальний алгоритм у базовому класі, залишаючи частини реалізації підкласам.

11. Нарисуйте структуру шаблону «Шаблонний метод».



12. Які класи входять в шаблон «Шаблонний метод», та яка між ними

взаємодія?

- **AbstractClass** – містить Template Method та абстрактні методи для підкласів.
- **ConcreteClass** – реалізує специфічну логіку окремих кроків алгоритму.
- **Взаємодія:** Template Method викликає абстрактні методи, реалізовані підкласами; підкласи змінюють тільки специфічні кроки.

13. Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

Template Method: визначає алгоритм і делегує частини реалізації підкласам.

Factory Method: визначає інтерфейс для створення об'єкта, а конкретні підкласи вирішують, який саме об'єкт створювати.

Головна різниця: Template Method – про порядок дій алгоритму, Factory Method – про створення об'єктів.

14. Яку функціональність додає шаблон «Міст»?

Забезпечує незалежне змінення абстракцій та реалізацій, спрощує розширення системи без множення підкласів, дає гнучкість при додаванні нових реалізацій або абстракцій