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: «Вступ до патернів проєктування»

«2. HTTP-сервер»

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

Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Основи проектування розгортання.

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» 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

1. Singleton

Шаблон Singleton належить до категорії породжувальних шаблонів і забезпечує існування лише одного екземпляра певного класу в системі. Його головна мета — гарантувати, що доступ до цього об’єкта здійснюється централізовано, через єдину глобальну точку. Конструктор класу робиться приватним, щоб запобігти створенню нових об’єктів, а доступ до єдиного екземпляра забезпечується через статичний метод (зазвичай getInstance()), який створює об’єкт лише під час першого виклику. Такий підхід дозволяє уникнути дублювання ресурсів, забезпечити узгодженість даних і спростити контроль над спільними об’єктами в програмі.

2. Iterator

Шаблон Iterator належить до поведінкових шаблонів і визначає стандартний спосіб послідовного доступу до елементів колекції без необхідності розкривати її внутрішню структуру. Основна ідея полягає в тому, щоб відокремити механізм обходу колекції від самої колекції, надавши спеціальний об’єкт — ітератор. Він зазвичай реалізує методи перевірки наявності наступного елемента та отримання поточного. Це спрощує роботу з різними структурами даних і робить код більш універсальним, оскільки клієнтський код не залежить від конкретного типу колекції.

3. **Proxy**

Шаблон Proxy також належить до структурних шаблонів і передбачає створення об’єкта-посередника, який виступає «замісником» реального об’єкта. Проксі контролює доступ до цього об’єкта, перехоплюючи запити від клієнта. Завдяки цьому можна реалізувати додаткову поведінку — наприклад, керування правами доступу, оптимізацію ресурсів, логування або відкладене створення об’єктів. Важливо, що проксі має той самий інтерфейс, що й об’єкт, який він заміщає, тому клієнт не помічає різниці між роботою з реальним об’єктом і з його замісником.

4. **State**

Шаблон State належить до поведінкових шаблонів і використовується для керування зміною поведінки об’єкта залежно від його внутрішнього стану. Замість того щоб застосовувати численні умовні оператори для перевірки стану, кожен стан оформлюється як окремий клас, який реалізує спільний інтерфейс. Контекст (основний об’єкт) зберігає посилання на поточний стан і делегує йому виконання дій. Під час зміни стану контекст просто замінює об’єкт стану, що робить систему більш гнучкою, розширюваною та зручною для підтримки. Такий підхід спрощує логіку керування й дозволяє легко додавати нові стани без змін у вже існуючих класах.

5. **Strategy**

Шаблон Strategy також є поведінковим і передбачає визначення сімейства взаємозамінних алгоритмів, кожен з яких інкапсульований у власному класі. Контекст, який використовує стратегію, має спільний інтерфейс для вибору потрібного алгоритму під час виконання програми. Такий підхід дозволяє змінювати спосіб виконання певної дії без зміни структури контексту. Шаблон зменшує кількість умовних конструкцій, робить систему відкритою для розширення та зручною для налаштування поведінки в різних ситуаціях.

Хід роботи

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.

Проблема: Для отримання запитів від клієнтів модуль використовується модуль HTTPRequest handler. Проте, логічно що під час старту, поки сервер не запустився HTTPRequest handler не повинен приймати запити від клієнтів, а після команди stop, поки сервер зупиняється, вже не приймав запити від клієнтів, але відповіді, якщо вони будуть готові, відправив.

Рішення: Ми явно можемо виділити два стани для об'єкта: stopped та running. Тому для вирішення краще за все підходить патерн State. Визначаємо загальний інтерфейс ServerState з методами, які по різному працюють в різних станах.

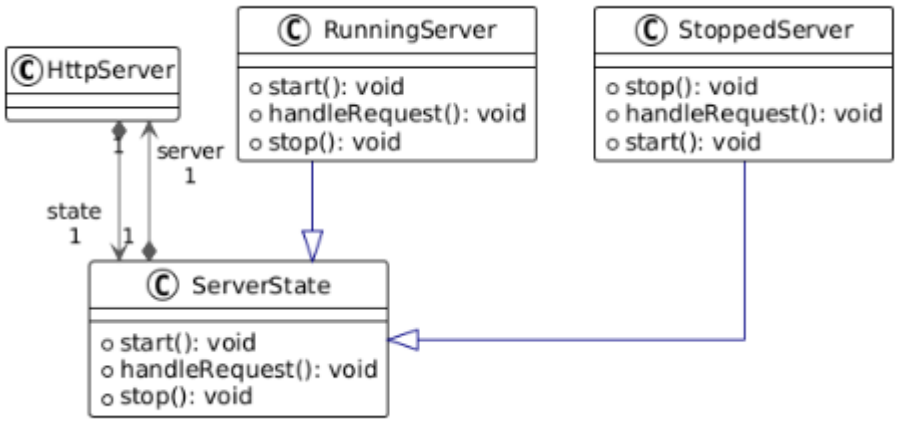


Рисунок 1 – реалізація класів за шаблоном "State"

Висновок: У процесі роботи HTTP-сервера виникає потреба змінювати його поведінку залежно від поточного стану – під час запуску, зупинки або активної роботи. Наприклад, модуль HTTPRequestHandler не повинен приймати нові запити, коли сервер ще не запущений або вже зупиняється, але має можливість завершити відправлення готових відповідей.

Така логіка безпосередньо пов’язана зі зміною станів об’єкта, тому використання шаблону State є найбільш доцільним рішенням. Він дозволяє уникнути громіздких умовних конструкцій у коді, розділивши поведінку на окремі класи-стани – Running та Stopped. Кожен стан реалізує власний варіант роботи методів через спільний інтерфейс ServerState, що робить архітектуру сервера гнучкою, масштабованою та зрозумілою.

```
package src.main;


import java.net.Socket;

public interface ServerState {
    void start(Server server);
    void stop(Server server);
    void handleClient(Server server, Socket client);
}


package src.main;

import java.net.ServerSocket;
import java.net.Socket;

public class StoppedServer implements ServerState {

    @Override
    public void start(Server server) {

        try {
            ServerSocket socket = new ServerSocket(server.getPort());
            server.setServerSocket(socket);

            System.out.println("Сервер працює на порті:" + server.getPort());

            server.setRunning(true);
            server.setState(new RunningServer());

            while (server.isRunning()) {
                Socket client = socket.accept();
                System.out.println("Клієнт під'єднаний: " + client.getInetAddress());
                server.handleClient(client);
            }

        } catch (Exception e) {
            if (server.isRunning()) {
                System.out.println("Помилка під час запуску сервера: " + e.getMessage());
            }
        } finally {
            System.out.println("Сокет сервера закрито.");
        }
    }

    @Override
    public void stop(Server server) {
        System.out.println("Сервер уже зупинений");
    }

    @Override
    public void handleClient(Server server, Socket client) {
```

```
        System.out.println("Сервер не може приймати запити, адже він зупинений");
    }
}


package src.main;

import java.net.Socket;

public class RunningServer implements ServerState {

    @Override
    public void start(Server server) {
        System.out.println("Сервер уже працює");
    }

    @Override
    public void stop(Server server) {
        System.out.println("Зупиняємо сервер...");
        server.setRunning(false);

        try {
            if (server.getServerSocket() != null) {
                server.getServerSocket().close();
            }
        } catch (Exception ignored) {}

        server.setState(new StoppedServer());
    }

    @Override
    public void handleClient(Server server, Socket client) {
        new Thread(() -> server.processClient(client)).start();
    }
}

package src.main;

import java.io.*;
import java.net.*;
import java.util.HashMap;
import java.util.Map;

public class Server {

    private ServerState state;
    private boolean running;
    private int port;
    private ServerSocket serverSocket;

    public Server(int port) {
        this.port = port;
        this.state = new StoppedServer();
        this.running = false;
```

```
    }

    public void setState(ServerState state) {
        this.state = state;
    }

    public ServerState getState() {
        return state;
    }

    public boolean isRunning() {
        return running;
    }

    public void setRunning(boolean running) {
        this.running = running;
    }

    public int getPort() {
        return port;
    }

    public void start() {
        state.start(this);
    }

    public void stop() {
        state.stop(this);
    }

    public void handleClient(Socket client) {
        state.handleClient(this, client);
    }


    public void processClient(Socket client) {
        try (BufferedReader reader = new BufferedReader(new
InputStreamReader(client.getInputStream()));
             OutputStream output = client.getOutputStream()) {

            String requestLine = reader.readLine();
            System.out.println("Отриманий запит: " + requestLine);

            String body = "Тіло запиту";

            Map<String, String> headers = new HashMap<>();
            headers.put("Content-Type", "text/plain");
            headers.put("Content-Length", String.valueOf(body.getBytes().length));

            HttpResponse response = HttpResponseDirector.Ok(body, headers);

            output.write(response.toHttpString().getBytes());
            output.flush();
```

```
            System.out.println("Відповідь надіслана.");

        } catch (IOException e) {
            System.out.println("Помилка під час оброки запиту: " + e.getMessage());
        } finally {
            try {
                client.close();
                System.out.println("Client connection closed.");
            } catch (IOException ignored) {}
        }
    }

    public ServerSocket getServerSocket() {
        return serverSocket;
    }

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

Відповіді на контрольні питання:

1. Що таке шаблон проєктування?

   Шаблон проєктування – це типове, перевірене рішення для частої архітектурної проблеми, яка виникає під час розробки програмного забезпечення. Він не є готовим кодом, а радше узагальненою схемою або концепцією, яку можна адаптувати під конкретну задачу.

2. Навіщо використовувати шаблони проєктування?

   Шаблони проєктування використовують для:

   підвищення гнучкості та масштабованості системи;

   зменшення складності коду завдяки перевіреним підходам;

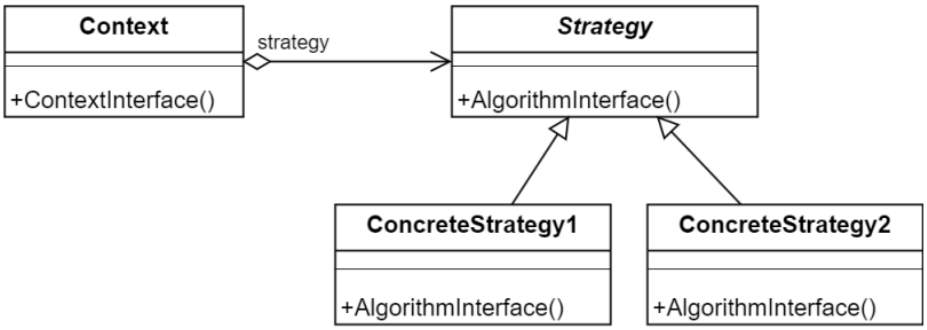
   покращення повторного використання компонентів;

   спрощення комунікації між розробниками, оскільки назва шаблону одразу описує архітектурну ідею.

3. Яке призначення шаблону «Стратегія»?

   Шаблон «Strategy» (Стратегія) дозволяє змінювати деякий алгоритм поведінки об'єкта іншим алгоритмом, що досягає ту ж мету іншим способом. Прикладом можуть служити алгоритми сортування: кожен алгоритм має власну реалізацію і визначений в окремому класі; вони можуть бути взаємозамінними в об'єкті, який їх використовує.

4. Структура шаблону проєктування «Стратегія»



5. Які класи входять в шаблон стратегія та яка між ними взаємодія?

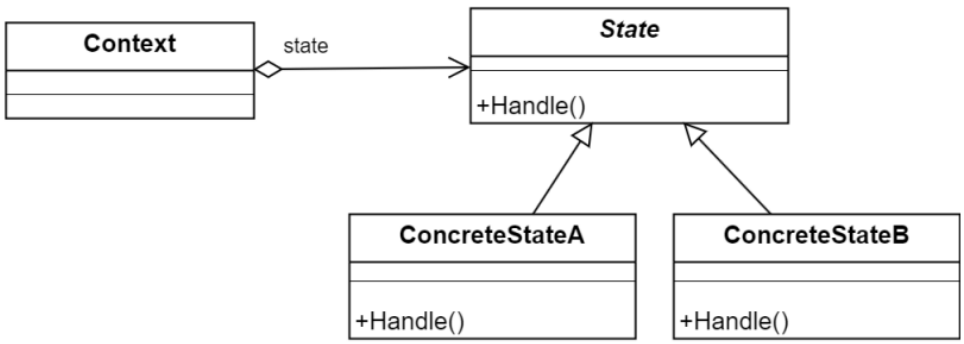
- Context – зберігає посилання на поточну стратегію і викликає її методи.
- Strategy (інтерфейс) – оголошує загальний метод для всіх алгоритмів.
- ConcreteStrategyA/B – реалізують різні алгоритми.

Взаємодія: Context делегує виконання алгоритму об’єкту стратегії, не знаючи, яку саме реалізацію він використовує.

6. Яке призначення шаблону «Стан»?

   Шаблон «State» (Стан) дозволяє змінювати логіку роботи об'єктів у випадку зміни їх внутрішнього стану. Реалізація даного шаблону полягає в наступному: пов'язані зі станом поля, властивості, методи і дії виносяться в окремий загальний інтерфейс (State); кожен стан являє собою окремий клас (ConcreteStateA, ConcreteStateB), які реалізують загальний інтерфейс

7. Структура шаблону «Стан»



8. Які класи входять в шаблон стан та яка між ними взаємодія?

   Context – головний об’єкт, що має поточний стан і делегує йому дії.

State (інтерфейс) – визначає методи для всіх можливих станів.

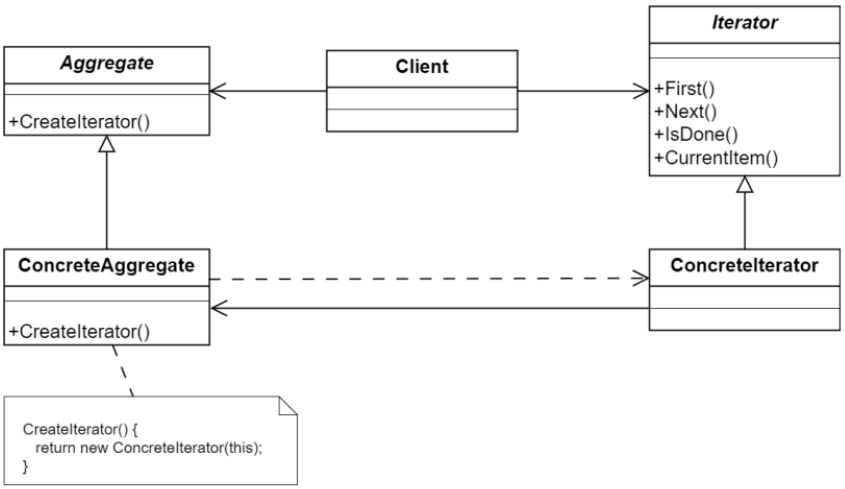
ConcreteStateA/B – реалізують поведінку, характерну для певного стану.

Взаємодія: Context викликає метод поточного стану, а стан може змінювати контекст на інший стан.

9. Яке призначення шаблону «Ітератор»?

   «Iterator» (Ітератор) являє собою шаблон реалізації об'єкта доступу до набору (колекції, агрегату) елементів без розкриття внутрішніх механізмів реалізації. Ітератор виносить функціональність перебору колекції елементів з самої колекції, таким чином досягається розподіл обов'язків: колекція відповідає за зберігання даних, ітератор – за прохід по колекції.

10. Структура шаблону проєктування «Ітератор»



11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?

    Iterator (інтерфейс) – визначає методи next() і hasNext().

    ConcreteIterator – реалізує логіку обходу елементів конкретної колекції.

    Aggregate (інтерфейс колекції) – створює ітератор.

    ConcreteAggregate – повертає конкретний ітератор.

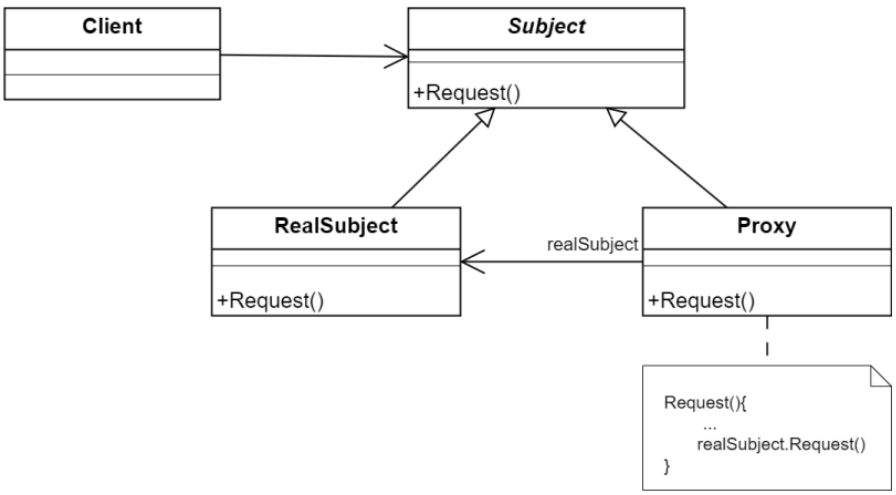
Взаємодія: клієнт працює з ітератором, не знаючи структури колекції.

12. «Singleton» (Одинак) являє собою клас в термінах ООП, який може мати не більше одного об'єкта (звідси і назва «одинак») Насправді, кількість об'єктів можна задати (тобто не можна створити більш n об'єктів даного

класу). Даний об'єкт найчастіше зберігається як статичне поле в самому класі.

13. Його часто називають анти-шаблоном, бо він порушує принцип інкапсуляції і створює глобальний стан, ускладнює тестування, оскільки важко ізолювати об’єкт, може призвести до прихованих залежностей у коді, погано масштабується в багатопоточних або розподілених системах.
14. Яке призначення шаблону «Проксі»?

    «Proxy» (Проксі) – об'єкти є об'єктами-заглушками або двійниками/замінниками для об'єктів конкретного типу. Зазвичай, проксі об'єкти вносять додатковий функціонал або спрощують взаємодію з реальними об'єктами. Проксі об'єкти використовувалися в більш ранніх версіях інтернетбраузерів, наприклад, для відображення картинки: поки картинка завантажується, користувачеві відображається «заглушка» картинки.
15. Структура патерну «Проксі»



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?
    - Subject (інтерфейс) – визначає спільний інтерфейс для Proxy та RealSubject.
    - RealSubject – об’єкт, до якого здійснюється доступ.
    - Proxy – зберігає посилання на RealSubject і контролює виклики до нього.

Взаємодія: клієнт викликає методи Proxy, який вирішує, коли і як передавати запит до реального об’єкта.