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 5
З дисципліни: «Теорія розробки програмного забезпечення»
Web-browser
Pattern Chain of Responsibility

Виконав:

студент групи ІА-14
Калінін Я.В

Перевірив:
Мягкий М.Ю

Київ 2023

Мета: Реалізувати шаблон проектування «Chain of Responsibility» відповідно до заданої теми :

Web-browser (proxy, chain of responsibility, factory method, template method, visitor, p2p) Веб-браузер повинен мати можливість зробити наступне: мати адресний рядок для введення адреси сайту, переміщатися і відображати структуру html документа, переглядати підключений javascript та css файли, перегляд всіх підключених ресурсів (зображень), коректна обробка відповідей з сервера (коди відповідей HTTP) - переходи при перенаправленнях, відображення сторінок 404 і 502/503.

Теоритичні відомості :

Ланцюжок обов'язків — це поведінковий патерн проектування, що дає змогу передавати запити послідовно ланцюжком обробників. Кожен наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком.

Ланцюжок обов'язків базується на тому, щоб перетворити окремі поведінки на об'єкти. У нашому випадку кожна перевірка переїде до окремого класу з одним методом виконання. Дані запиту, що перевіряється, передаватимуться до методу як аргументи.

Interface ResponseHandler

```
1  package com.example.demo.handler;
2
   4 implementations  Qwertua
3  public interface ResponseHandler {
       1 usage  1 implementation  Qwertua
4      void setNextHandler(ResponseHandler nextHandler);
5
       1 usage  3 implementations  Qwertua
6      boolean handleResponse(int statusCode);
7  }
```

setNextHandler: Цей метод встановлює наступний обробник (handler) в ланцюжку обробників помилок. Кожен обробник у ланцюжку володіє посиланням на наступний обробник. Якщо поточний обробник не може обробити запит, він передає його наступному обробнику у ланцюжку.

handleResponse: Цей метод обробляє статусний код відповіді і повертає true, якщо він успішно оброблений поточним обробником, або передає запит наступному обробнику і повертає false, якщо поточний обробник не може обробити цей код.

```java
package com.example.demo.handler;

3 inheritors  Qwertua
public abstract class BaseResponseHandler implements ResponseHandler {
    3 usages
    private ResponseHandler nextHandler;

    1 usage  Qwertua
    @Override
    public void setNextHandler(ResponseHandler nextHandler) {
        this.nextHandler = nextHandler;
    }

    3 usages  Qwertua
    protected boolean callNextHandler(int statusCode) {
        if (nextHandler != null) {
            return nextHandler.handleResponse(statusCode);
        }
        return false;
    }
}
```

setNextHandler Цей метод встановлює наступний обробник у ланцюжку. Він приймає об'єкт типу ResponseHandler, який буде наступним обробником після поточного.

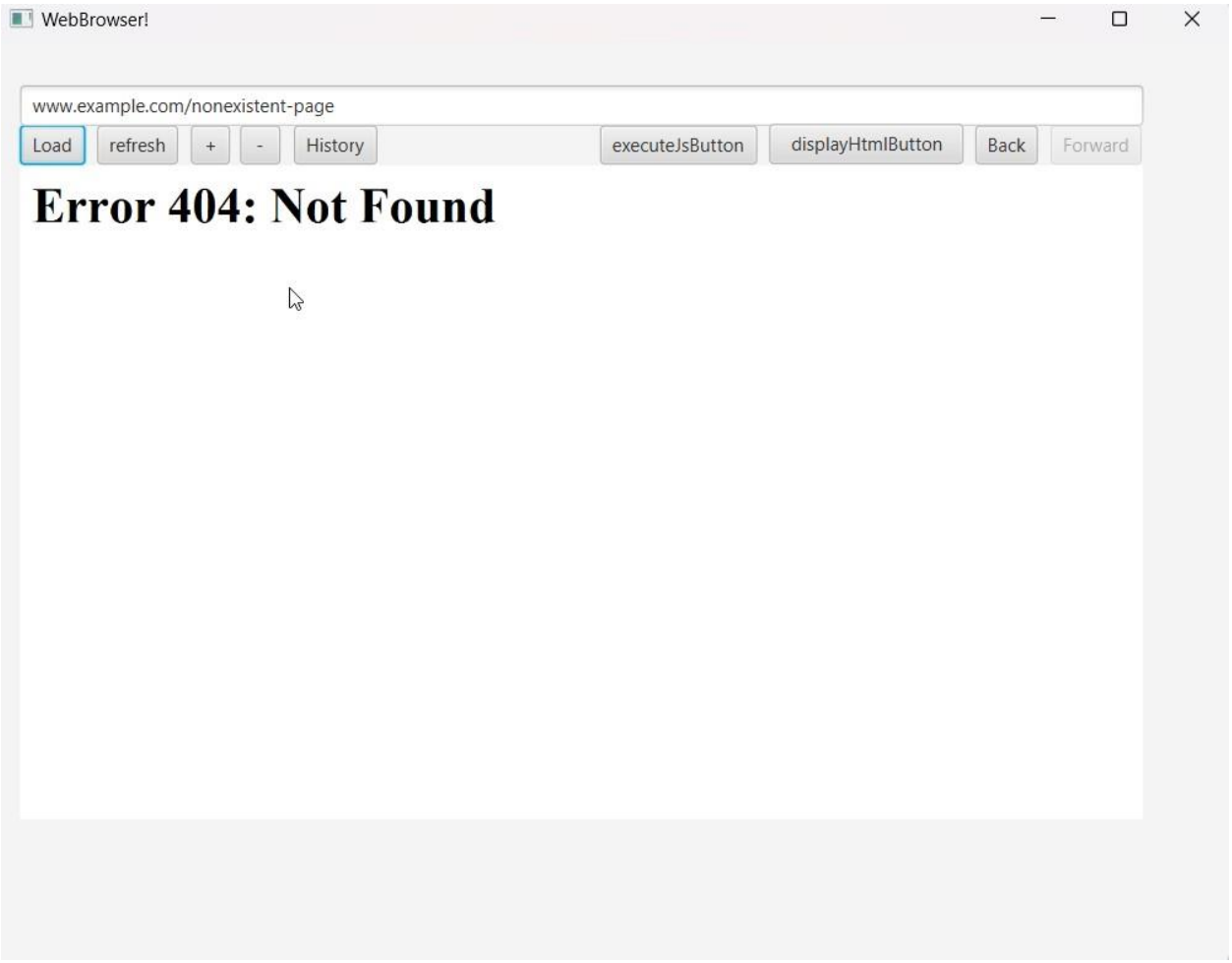
callNextHandler Цей метод викликає обробку наступного обробника в ланцюжку. Якщо наступний обробник успішно обробляє статусний код, метод повертає true, вказуючи на успішне оброблення. Якщо немає наступного обробника або він не може обробити код, метод повертає false.

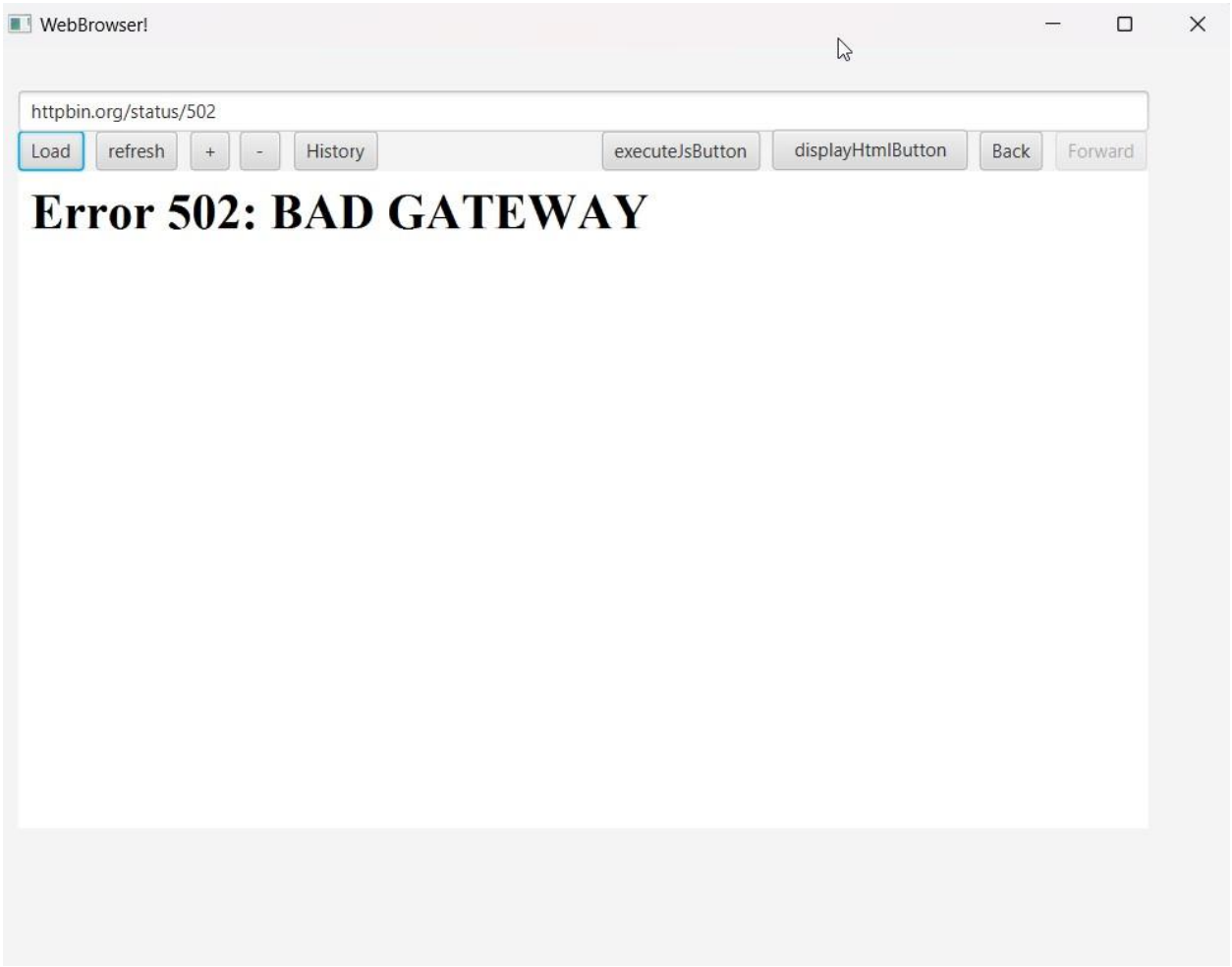
```java
package com.example.demo.handler;
 Qwertua *
public class BadGatewayHandler extends BaseResponseHandler {
    1 usage  Qwertua *
    @Override
    public boolean handleResponse(int statusCode) {
        if (statusCode == 502) {

            /*  Обробка помилки 502 */
            return true;
        } else {
            return callNextHandler(statusCode);
        }
    }
}

```

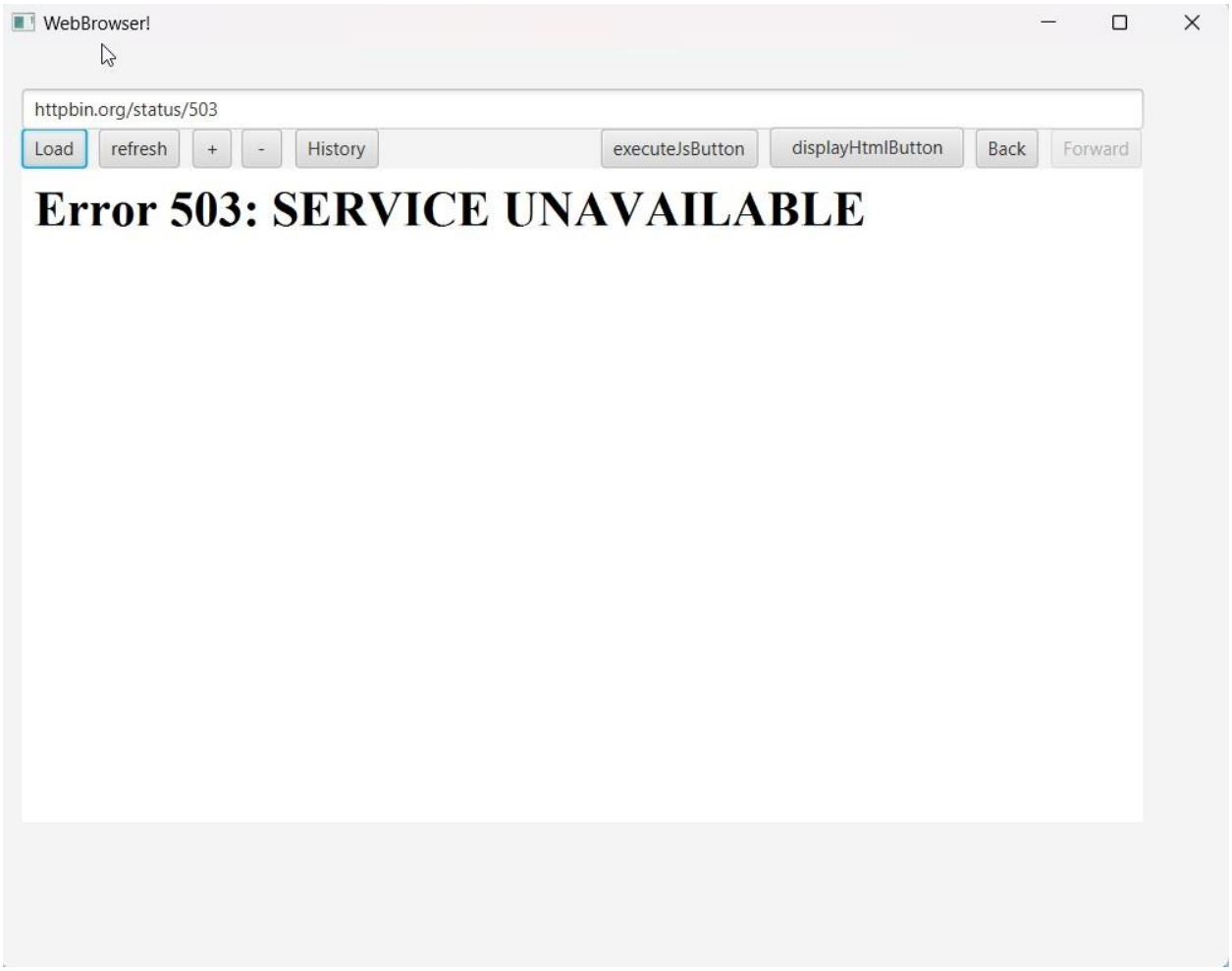
```java
package com.example.demo.handler;
Qwertua *
public class NotFoundHandler extends BaseResponseHandler {
    1 usage  Qwertua *
    @Override
    public boolean handleResponse(int statusCode) {
        if (statusCode == 404) {

          /*   Обробка помилки 404*/
            return true;
        } else {
            return callNextHandler(statusCode);
        }
    }
}
```

```java
package com.example.demo.handler;

Qwertua *
public class ServiceUnavailableHandler extends BaseResponseHandler {
    1 usage  Qwertua *
    @Override
    public boolean handleResponse(int statusCode) {
        if (statusCode == 503) {

           /* Обробка помилки 503*/
            return true;
        } else {
            return callNextHandler(statusCode);
        }
    }
}
```

Результат:


WebBrowser!
www.example.com/nonexistent-page
Load
refresh
+
-
History
executeJsButton
displayHtmlButton
Back
Forward
Error 404: Not Found


WebBrowser!
httpbin.org/status/502
Load
refresh
+
-
History
executeJsButton
displayHtmlButton
Back
Forward
Error 502: BAD GATEWAY



Висновок : у цій лабораторній роботі я детально розглянули шаблон проектування ChainOfResponsibility та реалізував у своєму проекті.