МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Лабораторна робота №6

Тема**: «**Патерн проектування Замісник»

Дисципліна: «Ефективність та якість архітектурних рішень інформаційних систем»

Виконав: студент групи ІКМ-М224а
Нестеренко Владислав Валентинович

Перевірив: Асистент кафедри
Хорошун Андрій Сергійович

Харків - 2024р.

Мета: Здобути навички з реалізацією паттерна проектування Замісник.

Завдання:

1. Ознайомитися з патерном Замісник.
2. Виконати всі пункти лабораторної роботи
3. Надати звіт про виконану роботу

Хід роботи:

1. Є система у якій реалізовано процес завантаження файлів. (interface - **Downloader**, який має метод **download**).
2. В системі також є класс **SimpleDownloader** який є реалізацією інтерфейсу Downloader і відповідає за завантаження файлів.
3. Необхідно додати до системі механізм кешування завантаженних данних не додаючи змін до початкового стану класса **SimpleDownloader**

Необхідно створити структуру класів та методів яка буде демонструвати реалізацію патерну Замісник і буде вирішувати описанне завдання.

Також необхідно навести приклад клієнтського коду де буде реалізовано умовний рендерінг обох типів сторінок у кількох типах.

У рамках виконання лабораторної роботи не потрібно описувати деталі реалізації самих методів! Достатньо вказати сам метод та параметри який він приймає та повертає.

Виконання:

Замісник — це структурний патерн проектування, що дає змогу підставляти замість реальних об’єктів спеціальні об’єкти-замінники. Ці об’єкти перехоплюють виклики до оригінального об’єкта, дозволяючи зробити щось до чи після передачі виклику оригіналові.

Реалізація на Java:

Інтерфейс Downloader - визначає метод download, який приймає URL як параметр і повертає масив байтів..

```
//Інтерфейс, що визначає метод завантаження файлів.
public interface Downloader {
    byte[] download(String url);
}
```

Клас SimpleDownloader: реалізує метод download для завантаження файлів.

```
public class SimpleDownloader implements Downloader {
    @Override
    public byte[] download(String url) {
        // Логіка завантаження файлу з URL
        return new byte[0];
```

```
    }
}
```

Клас CachedDownloader: реалізує кешування даних. Якщо дані вже завантажені (присутні в кеші), вони повертаються з кешу, інакше виконується завантаження через SimpleDownloader.

```
import java.util.HashMap;
import java.util.Map;

public class CachedDownloader implements Downloader {
    private Downloader downloader;
    private Map<String, byte[]> cache;

    public CachedDownloader(Downloader downloader) {
        this.downloader = downloader;
        this.cache = new HashMap<>();
    }

    @Override
    public byte[] download(String url) {
        if (cache.containsKey(url)) {
            return cache.get(url);
        }
        byte[] data = downloader.download(url);
        cache.put(url, data);
        return data;
    }
}
```

Клієнтський код: демонструє використання як простого завантажувача, так і завантажувача з кешуванням.

```
public class Main {
    public static void main(String[] args) {
        Downloader simpleDownloader = new SimpleDownloader();
        Downloader cachedDownloader = new CachedDownloader(simpleDownloader);

        // Використання простого завантажувача
        byte[] data1 = simpleDownloader.download("http://example.com/file1");

        // Використання завантажувача з кешуванням
        byte[] data2 = cachedDownloader.download("http://example.com/file2");
        byte[] data3 = cachedDownloader.download("http://example.com/file2");
// Використає кеш

        // Умовний рендерінг сторінок
        renderPage(simpleDownloader);
        renderPage(cachedDownloader);
    }

    public static void renderPage(Downloader downloader) {
        // Логіка рендерінгу сторінки використовуючи завантажувач
        // downloader.download("http://example.com/somepage");
    }
}
```

Висновки

Під час виконання даної лабораторної роботи я здобув навички з реалізації патерна проектування «Замісник».

Я створив структуру класів та методів, яка демонструє застосування цього патерна для додавання механізму кешування завантажених даних без внесення змін до існуючого класу завантажувача. Зокрема, я реалізував інтерфейс Downloader, який містить метод download, та два класи: SimpleDownloader для простого завантаження файлів і CachedDownloader для завантаження файлів з використанням кешування. Клас CachedDownloader реалізує кешування даних, перевіряючи наявність даних у кеші перед виконанням завантаження, і використовує SimpleDownloader для фактичного завантаження файлів. Також я розробив приклад клієнтського коду, який демонструє використання як простого завантажувача, так і завантажувача з кешуванням, а також умовний рендерінг сторінок, що використовують обидва типи завантажувачів.