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: «Шаблони «SINGLETON»,
«ITERATOR», «PROXY», «STATE»,
«STRATEGY»»

Варіант №26

Виконав:
студент групи ІА-23
Мозоль В.О

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Тема.
Шаблони «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»

Мета.
Метою даної лабораторної роботи є ознайомлення з шаблонами проєктування, зокрема з шаблоном "Iterator", та їх практичне застосування при розробці програмного забезпечення.

Завдання.
1 . Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми

Обрана тема.
26 Download manager (iterator, command, observer, template method, composite, p2p)
Інструмент для скачування файлів з інтернету по протоколах http або https з можливістю продовження завантаження в зупиненому місці, розподілу швидкостей активним завантаженням, ведення статистики завантажень, інтеграції в основні браузери (firefox, opera, internet explorer, chrome).

Короткі теоретичні відомості.

Шаблони проєктування (або патерни) — це перевірені практикою рішення загальних проблем проєктування програмного забезпечення. Вони описують стандартні підходи, які можна застосувати в різних ситуаціях, щоб вирішити певні проблеми, зберігаючи при цьому структуру системи зрозумілою та підтримуваною. Шаблони проєктування мають загальновживані назви, що дозволяє розробникам легко обговорювати та впроваджувати відповідні підходи. Приклади шаблонів включають Singleton, Factory, Observer, Command, Iterator тощо.

Основні переваги шаблонів проєктування:

- Зменшення часу та зусиль на створення архітектури.
- Гнучкість та адаптованість системи до змін.
- Полегшення підтримки та розвитку системи.
- Стійкість системи до змін та спрощення інтеграції з іншими системами.

Iterator — це приклад шаблону поведінки, який надає спосіб поелементного проходження по колекції об'єктів без розкриття її внутрішньої структури. Основна ідея шаблону в тому, щоб винести функцію ітерації в окремий клас, дозволяючи колекції зосередитися на зберіганні даних. Це досягається через методи First(), Next(), IsDone, та CurrentItem, які дозволяють ініціювати та здійснювати ітерацію по елементах.

Iterator дозволяє створити різні способи обходу колекції без зміни самої колекції. Це дає змогу реалізувати обхід не тільки у порядку, а й, наприклад, у зворотньому порядку, за парними або непарними індексами тощо.

Основні переваги та недоліки шаблону Iterator

Переваги:

- Уніфікація способу обходу для всіх колекцій.
- Можливість реалізації різних способів обходу.
- Спрощення коду зберігання даних.

Недоліки:

- Може бути зайвим для простих колекцій, де підходить звичайний цикл.

Хід роботи.

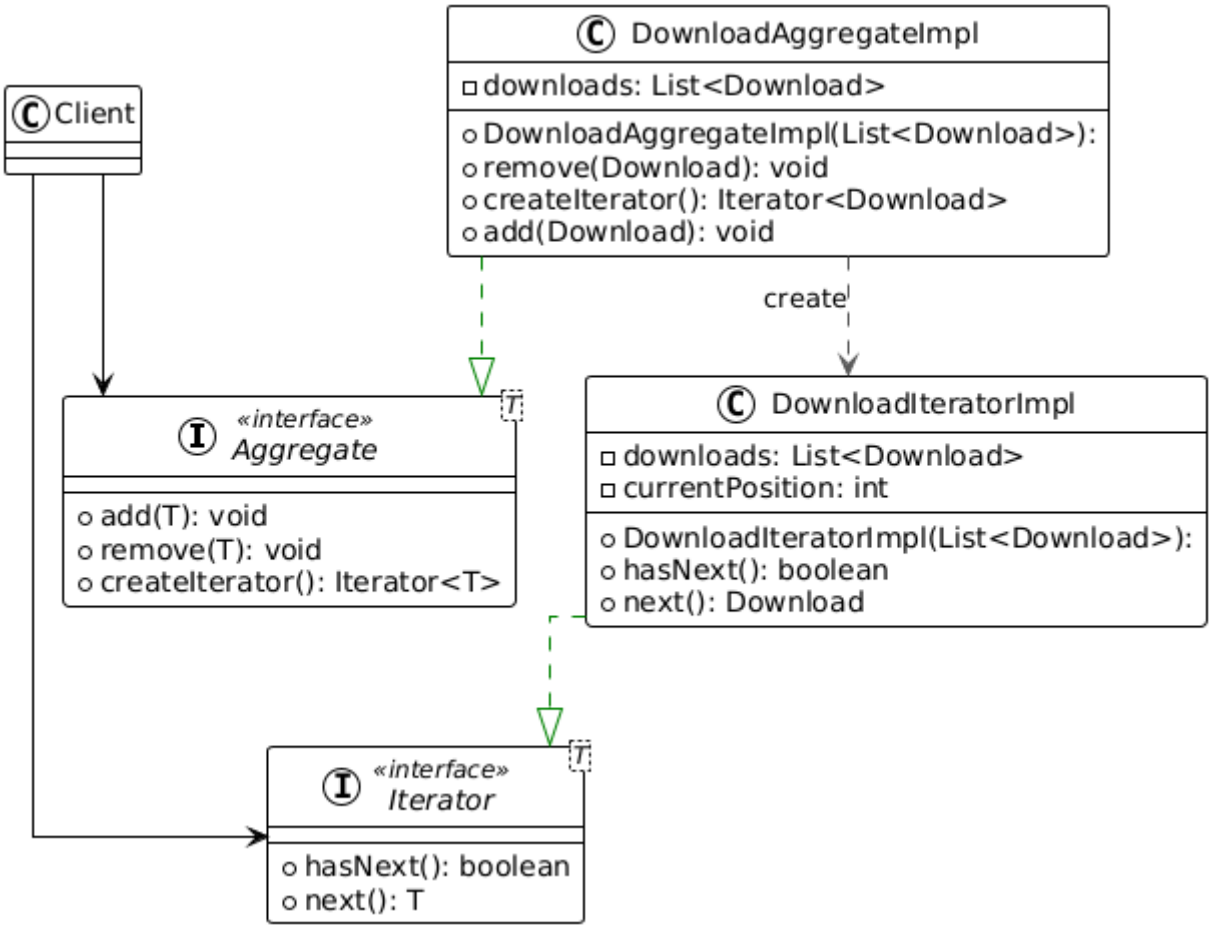

Рисунок №1 – UML діаграма шаблону Ітератор

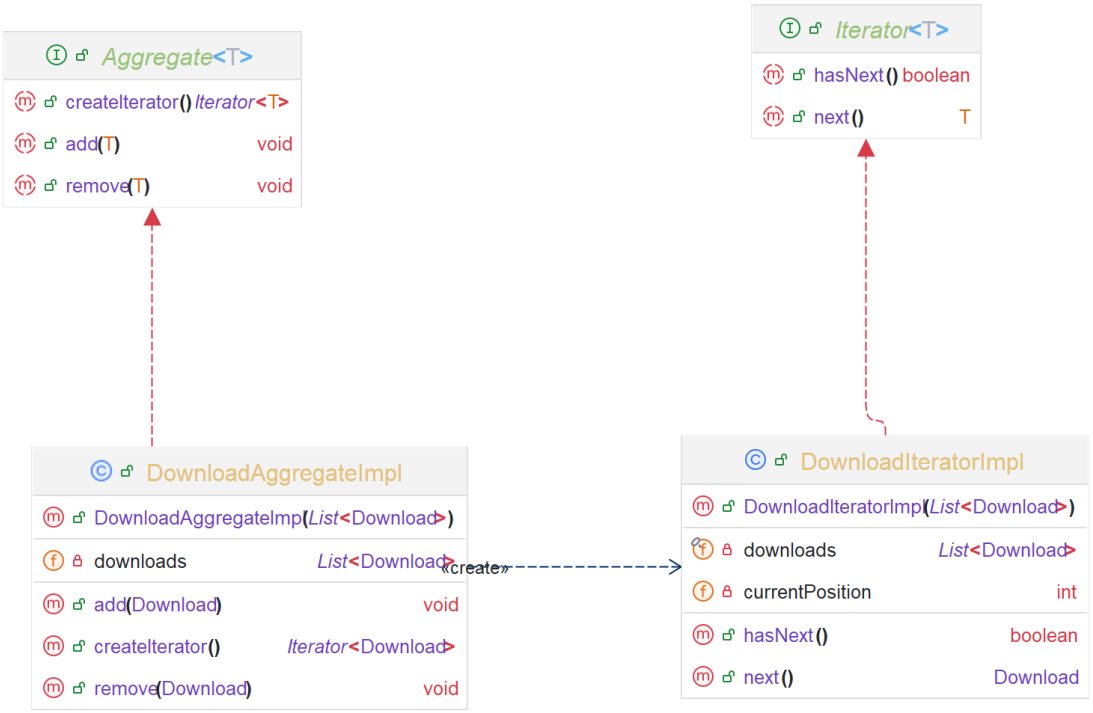

Рисунок №2 – Діаграма класів , згенерована IDE, реалізації шаблону Iterator

На другому зображенні ми бачимо інтерфейс Aggregate та його реалізацію DownloadAggregateImpl. Цей клас містить список завантажень List<Download> та надає методи для управління цим списком, такі як add(T), remove(T) та createIterator().

Далі на цьому зображенні представлений інтерфейс Iterator та його реалізація DownloadIteratorImpl. Цей клас містить посилання на список завантажень та поточну позицію в цьому списку. Він надає методи для перебору елементів, такі як hasNext() та next().

На першому зображенні ми бачимо UML-діаграму, яка наочно демонструє взаємозв'язок між класами Aggregate, Iterator та їх конкретними реалізаціями. Тут ми можемо побачити, що DownloadAggregateImpl реалізує інтерфейс Aggregate, а DownloadIteratorImpl реалізує інтерфейс Iterator. Також показано, що DownloadAggregateImpl має метод createIterator(), який повертає об'єкт Iterator.

Це дозволяє приховати деталі реалізації колекції від клієнтського коду та надає уніфікований спосіб для перебору її елементів.

Код реалізації шаблону можна переглянути у GitHub репозиторії у папці DownloadManager або у Додатку А.

Робота паттерну.

Для демонстрації роботи паттерну використаємо заглушки завантажень (Додаток А – Заглушки).

Будемо ітератором проходитись по колекції завантажень виводити інформацію про завантаження в консоль та починати завантажувати файл по url який зберігається у моделі завантаження (Додаток А – Код для тестування Ітератору).

Результат виконання коду:

```
Download id: 1001 url: https://file-examples.com/wp-content/storage/2017/02/file-sample_1MB.docx size: 1524.5 downloaded:
Download id: 1002 url: https://file-examples.com/wp-content/storage/2018/04/file_example_AVI_1920_2_3MG.avi size: 2048.75 downloaded:
Download id: 1003 url: https://drive.usercontent.google.com/u/0/uc?id=1vCi-Q1KBUJwiD54_DRqWIuzl4JqiNe09&export=download size: 15.25 downloaded:
******
Download completed: D:/DownloadTest\file_example_AVI_1920_2_3MG.avi
Download completed: D:/DownloadTest\file-sample_1MB.docx
Download completed: D:/DownloadTest\uc
All downloads completed.
```

Рисунок №3 – Робота паттерну Ітератор

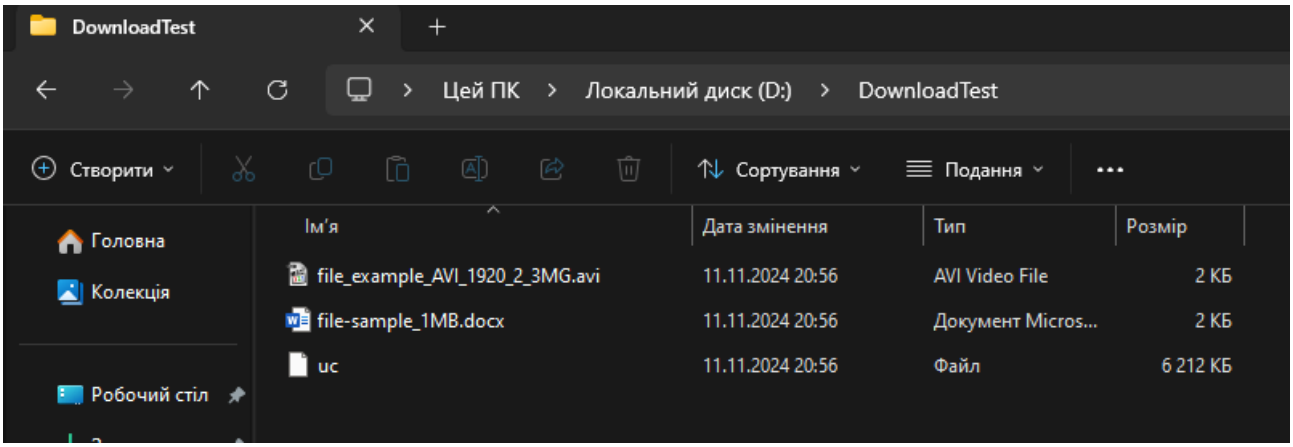

Рисунок №4– Наявність завантажених файлів

Висновки.

В результаті виконання лабораторної роботи було реалізовано шаблон проєктування "Iterator" для керування колекцією завантажень. Було продемонстровано, як за допомогою ітератора можна проходити по колекції завантажень, отримувати інформацію про кожне завантаження та починати процес завантаження файлу.

Додаток А.

Interface Aggregate<T>:

```
package com.project.downloadmanager.util.iterator;


public interface Aggregate<T> {
    public void add(T t);
    public void remove(T t);
    public Iterator<T> createIterator();
}
```

Interface Iterator<T>:

```
package com.project.downloadmanager.util.iterator;

public interface Iterator<T> {
    public boolean hasNext();
    public T next();
}
```

Class DownloadAggregateImpl:

```
package com.project.downloadmanager.util.iterator.impl;


import com.project.downloadmanager.model.Download;
import com.project.downloadmanager.util.iterator.Aggregate;
import com.project.downloadmanager.util.iterator.Iterator;

import java.util.List;

public class DownloadAggregateImpl implements Aggregate<Download> {

    private List<Download> downloads;

    public DownloadAggregateImpl(List<Download> downloads) {
        this.downloads = downloads;
    }

    @Override
    public void add(Download download) {
        downloads.add(download);
    }

    @Override
    public void remove(Download download) {
        downloads.remove(download);
    }

    @Override
    public Iterator<Download> createIterator() {
        return new DownloadIteratorImpl(downloads);
    }
}
```

Class DownloadIteratorImpl:

```
package com.project.downloadmanager.util.iterator.impl;


import com.project.downloadmanager.model.Download;
import com.project.downloadmanager.util.iterator.Iterator;

import java.util.List;
import java.util.NoSuchElementException;

public class DownloadIteratorImpl implements Iterator<Download> {

    private int currentPosition = 0;
    private final List<Download> downloads;
```

```
    public DownloadIteratorImpl(List<Download> downloads) {
        this.downloads = downloads;
    }

    @Override
    public boolean hasNext() {
        return currentPosition < downloads.size();
    }

    @Override
    public Download next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        return downloads.get(currentPosition++);
    }
}
```

Код для тестування Ітератору:

```
public void downloadStart() {

    List<Thread> downloadThreads = new ArrayList<>();

    Aggregate<Download> aggregate = new DownloadAggregateImpl(new
DownloadRepository().findAll());
    Iterator<Download> downloadIterator = aggregate.createIterator();
    while (downloadIterator.hasNext()) {
        Download download = downloadIterator.next();
        System.out.println(download.toString());

        Download downloadTask = new Download(download.getUrl());

        Thread downloadThread = new Thread(downloadTask);
        downloadThreads.add(downloadThread);
        downloadThread.start();
    }
    System.out.println("******");

    for (Thread thread : downloadThreads) {
        try {
            thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    System.out.println("All downloads completed.");
}
```

Заглушки:

```
@Override
public List<Download> findAll() {
    List<Download> downloads = new ArrayList<>();

    Download download1 = new Download(
            1001L,
            "https://file-examples.com/wp-content/storage/2017/02/file-sample_1MB.docx",
            123L,
            1524.50,
            new java.util.Date(System.currentTimeMillis() - 3600000),
            DownloadStatus.COMPLETED,
            new java.util.Date(System.currentTimeMillis())
    );
    Download download2 = new Download(
            1002L,
            "https://file-examples.com/wp-
content/storage/2018/04/file_example_AVI_1920_2_3MG.avi",
            456L,
            2048.75,
            new java.util.Date(System.currentTimeMillis() - 7200000),
            DownloadStatus.DOWNLOADING,
```

```
            null
    );
    Download download3 = new Download(
            1003L,
            "https://drive.usercontent.google.com/u/0/uc?id=1vCi-
Q1KBUJwiD54_DRqWIuzl4JqiNe09&export=download",
            789L,
            15.25,
            new java.util.Date(System.currentTimeMillis() - 900000),
            DownloadStatus.ERROR,
            new Date(System.currentTimeMillis() - 600000)
    );

    downloads.add(download1);
    downloads.add(download2);
    downloads.add(download3);
    return downloads;
}
```

Config.properties:

Місце де вказується директорія для збереження завантажень:

```
downloadDirectory= D:/DownloadTest
```