Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки

Технології розроблення програмного забезпечення

Лабораторна робота №4

ШАБЛОНИ «SINGLETON», «ITERATOR»,
«PROXY», «STATE», «STRATEGY»

Виконала
студентка групи ІА–24:
Кармазіна А. В.

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Завдання 3
Тема: 26 Download manager 3
Теоретичні відомості 3
Хід роботи 4
Реалізація шаблону Iterator 4
Висновок 6

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.

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) (https://github.com/Karmazinanastya/TRPZ_Karmazina_labs/tree/main/FileDownloader)

Теоретичні відомості

Шаблон **Singleton** (Одинак) забезпечує існування лише одного екземпляра класу та надає глобальну точку доступу до нього. Його мета — контроль доступу до єдиного екземпляра, наприклад, для управління базою даних чи журналом. Реалізується через приватний конструктор і статичний метод доступу, часто з потокобезпекою.

Iterator (Ітератор) забезпечує спосіб послідовного доступу до елементів колекції без розкриття її внутрішньої структури. Його використовують для обходу складних структур даних, таких як масиви чи дерева. Основна перевага — відокремлення логіки обходу від самої колекції через методи next() і hasNext().

Proxy (Замісник) створює сурогатний об’єкт, який контролює доступ до реального. Цей шаблон дозволяє додати функціонал, наприклад, кешування або перевірку доступу перед викликом об’єкта. Часто використовується для лінивого завантаження ресурсів, наприклад, зображень або даних, і реалізується як обгортка навколо основного об'єкта.

State (Стан) дозволяє об'єкту змінювати свою поведінку залежно від внутрішнього стану. Це спрощує логіку умовних операторів, які впливають на поведінку об'єкта. Наприклад, банкомат може перебувати в станах "готовий", "немає грошей" або "очікує введення", і кожен з них реалізується окремим класом.

Strategy (Стратегія) визначає сімейство алгоритмів, інкапсулює кожен із них і робить їх взаємозамінними. Цей шаблон дозволяє змінювати алгоритм роботи об'єкта під час виконання програми. Прикладом може бути вибір різних методів сортування (швидке чи бульбашкове сортування). Алгоритми реалізуються через інтерфейс або абстрактний клас, який описує загальні методи.

Хід роботи

Реалізація шаблону **Iterator**

Шаблон Iterator в цьому проєкті використовується для ітерації через список об'єктів історії завантажень. Цей шаблон надає стандартний спосіб доступу до елементів колекції без необхідності розкривати її внутрішню реалізацію. У коді це реалізовано через інтерфейс IIterator<T> та клас HistoryIterator.

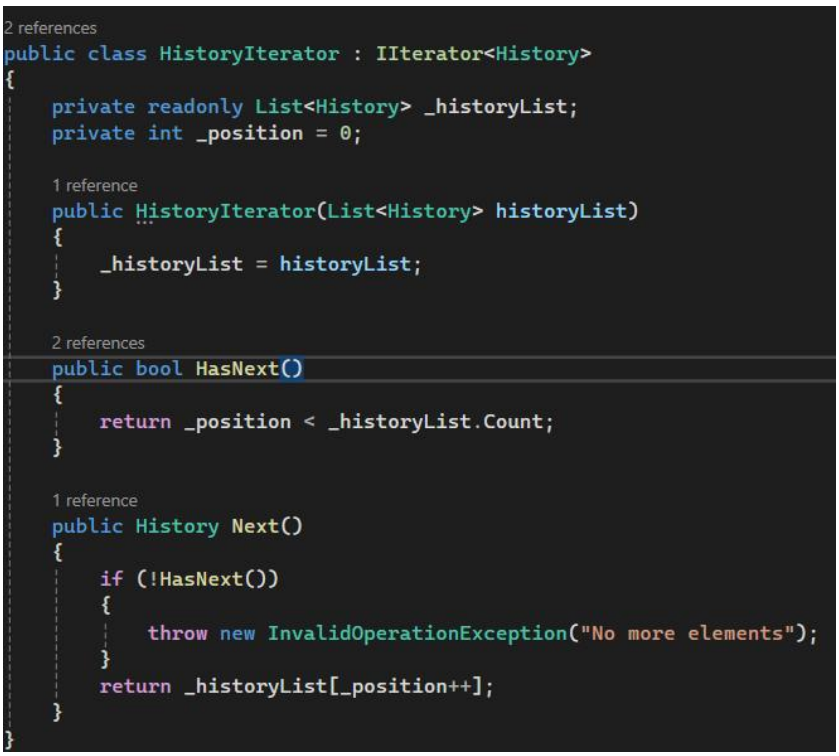
```
namespace FileDownloader.Data.Models
{
    2 references
    public interface IIterator<T>
    {
        2 references
        bool HasNext();
        1 reference
        T Next();
    }
```

Рис. 1 - Інтерфейс IIterator<T>

Це базовий інтерфейс, який визначає два основні методи для ітератора:

HasNext() — перевіряє, чи є наступний елемент у колекції.

Next() — повертає наступний елемент у колекції.



```
2 references
public class HistoryIterator : IIterator<History>
{
    private readonly List<History> _historyList;
    private int _position = 0;

    1 reference
    public HistoryIterator(List<History> historyList)
    {
        _historyList = historyList;
    }

    2 references
    public bool HasNext()
    {
        return _position < _historyList.Count;
    }

    1 reference
    public History Next()
    {
        if (!HasNext())
        {
            throw new InvalidOperationException("No more elements");
        }
        return _historyList[_position++];
    }
}
```

Рис. 2 - Клас HistoryIterator

Цей клас реалізує інтерфейс IIterator<History> і забезпечує доступ до елементів колекції історії завантажень.
Ключові властивості та методи:

_historyList — список об'єктів історії.

_position — індекс поточного елемента в колекції.

HasNext() — перевіряє, чи залишилися невідвідані елементи в списку.

Next() — повертає наступний елемент і збільшує позицію

Висновок

Шаблон Iterator є оптимальним рішенням для роботи з колекцією історії завантажень у цьому проєкті з кількох причин. По-перше, він інкапсулює внутрішню структуру колекції, надаючи єдиний, стандартний спосіб доступу до її елементів. Це дозволяє змінювати реалізацію колекції без зміни коду, який працює з нею. По-друге, користувачі шаблону можуть ітерувати колекцією без необхідності займатися управлінням індексами або перевірками меж, що спрощує код. Крім того, шаблон дозволяє легко додавати нові стратегії ітерації, такі як фільтрація чи зворотна ітерація, без зміни самих колекцій, що забезпечує високу розширюваність. Останнє, але не менш важливе, — ітератор дозволяє розділити логіку ітерації та обробки даних, що зберігає принцип єдиного обов’язку в програмі та робить код легким для тестування та підтримки. Завдяки цим перевагам шаблон Iterator забезпечує більшу гнучкість і підтримуваність, що є важливими у масштабованих проєктах