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

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)

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 **Strategy**

Шаблон Strategy є частиною групи шаблонів поведінки та дозволяє визначити набір алгоритмів, інкапсулювати їх і зробити їх взаємозамінними. Це дозволяє змінювати алгоритм роботи об'єкта під час виконання, не змінюючи самого об'єкта.

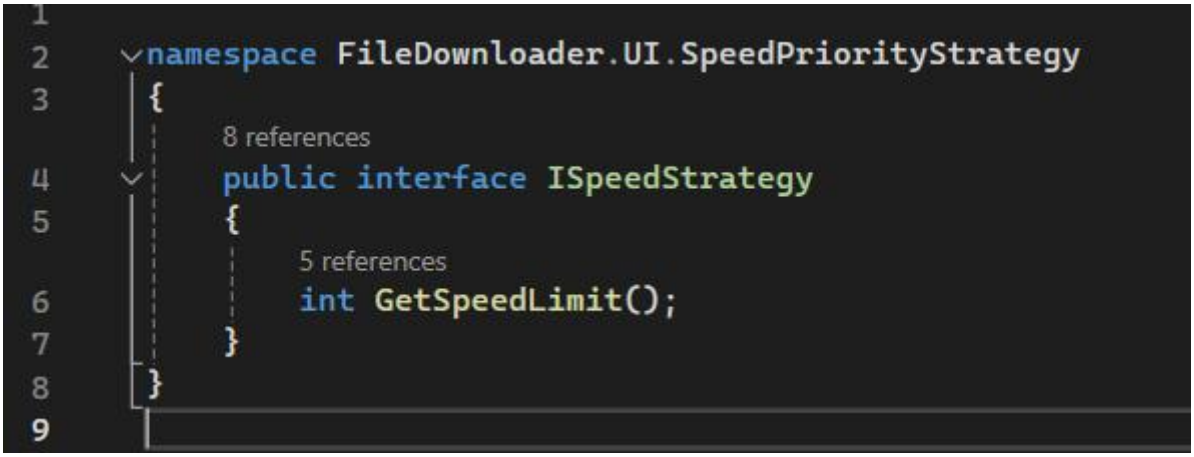
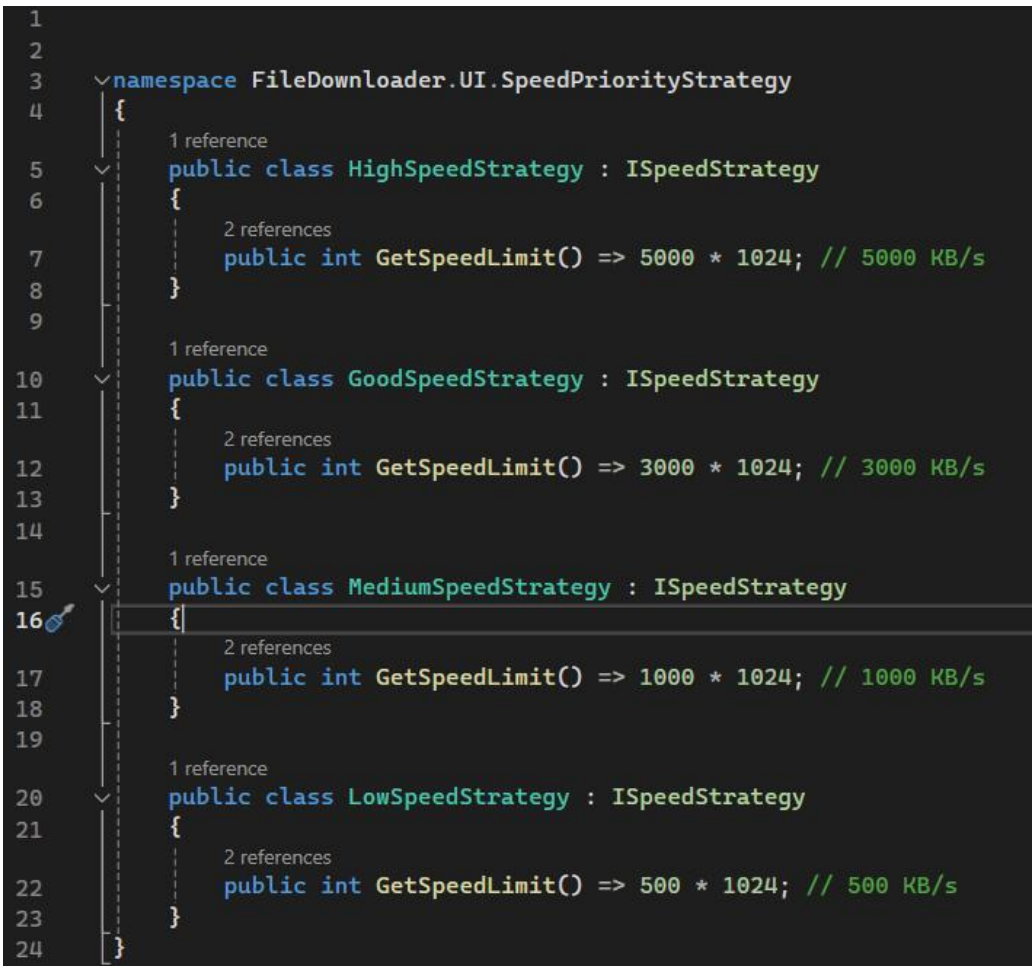


Рис. 1 - Інтерфейс ISpeedStrategy

Цей інтерфейс визначає метод GetSpeedLimit(), який буде реалізувати кожна стратегія швидкості. Це дозволяє використовувати різні стратегії для обмеження швидкості завантаження.

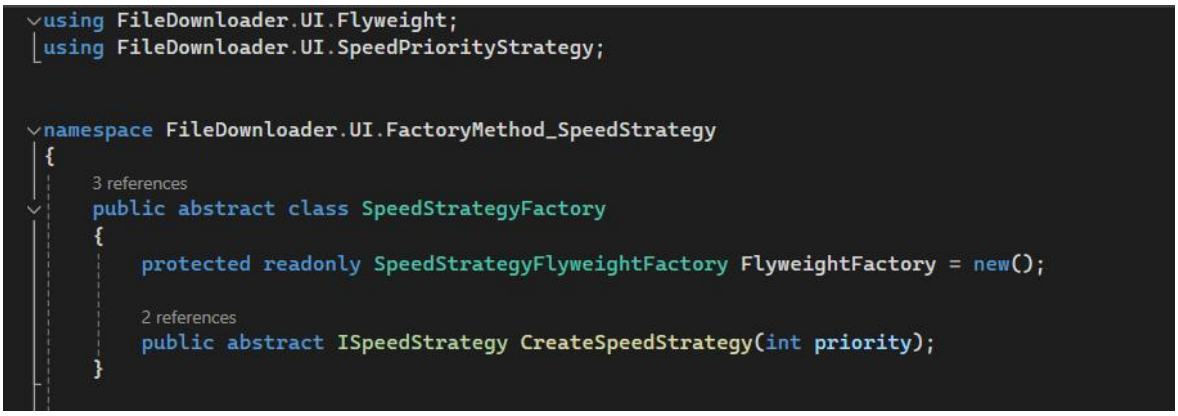
Класи, що реалізують цей інтерфейс, визначають різні стратегії для ліміту швидкості.



```
namespace FileDownloader.UI.SpeedPriorityStrategy
{
    1 reference
    public class HighSpeedStrategy : ISpeedStrategy
    {
        2 references
        public int GetSpeedLimit() => 5000 * 1024; // 5000 KB/s
    }

    1 reference
    public class GoodSpeedStrategy : ISpeedStrategy
    {
        2 references
        public int GetSpeedLimit() => 3000 * 1024; // 3000 KB/s
    }

    1 reference
    public class MediumSpeedStrategy : ISpeedStrategy
    {
        2 references
        public int GetSpeedLimit() => 1000 * 1024; // 1000 KB/s
    }

    1 reference
    public class LowSpeedStrategy : ISpeedStrategy
    {
        2 references
        public int GetSpeedLimit() => 500 * 1024; // 500 KB/s
    }
}
```

Рис. 2 - Класи, що реалізують інтерфейс ISpeedStrategy

Кожен з цих класів реалізує метод GetSpeedLimit() і повертає конкретне значення швидкості, яке обмежує завантаження. Залежно від обраної стратегії, буде встановлено різний ліміт швидкості для завантаження файлів.



```
using FileDownloader.UI.Flyweight;
using FileDownloader.UI.SpeedPriorityStrategy;


namespace FileDownloader.UI.FactoryMethod_SpeedStrategy
{
    3 references
    public abstract class SpeedStrategyFactory
    {
        protected readonly SpeedStrategyFlyweightFactory FlyweightFactory = new();

        2 references
        public abstract ISpeedStrategy CreateSpeedStrategy(int priority);
    }
```

Рис. 3 - SpeedStrategyFactory

SpeedStrategyFactory — це абстрактний клас, який визначає метод CreateSpeedStrategy(). Цей метод має створювати об'єкт стратегії на основі пріоритету (наприклад, високий, середній, низький).

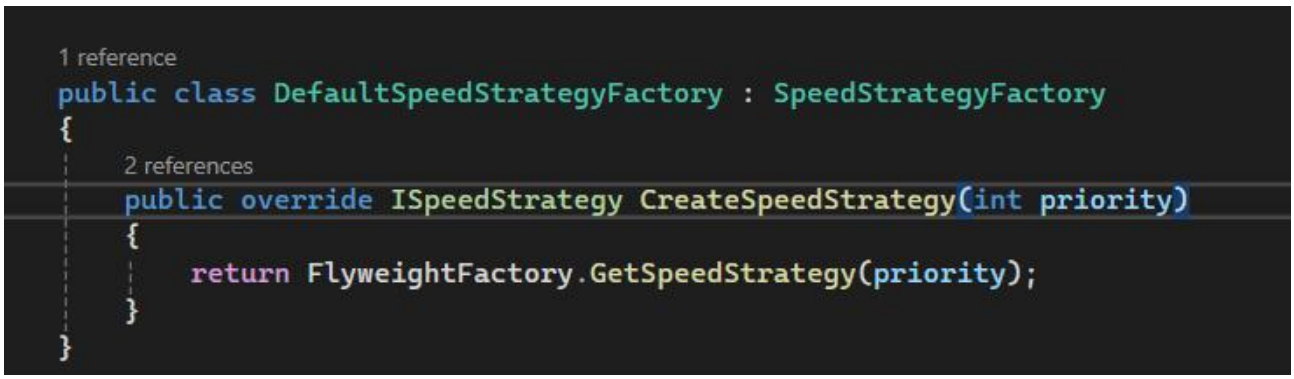


Рис. 4 - Клас DefaultSpeedStrategyFactory

DefaultSpeedStrategyFactory реалізує метод CreateSpeedStrategy(), який створює стратегію, звертаючись до фабрики SpeedStrategyFlyweightFactory. Цей клас забезпечує вибір стратегії на основі переданого пріоритету.

Рис. 5 - Фабрика SpeedStrategyFlyweightFactory

SpeedStrategyFlyweightFactory є фактичною фабрикою, яка створює й кешує стратегії швидкості в залежності від пріоритету. Використовуючи патерн Flyweight, цей клас забезпечує повторне використання стратегії, що дозволяє знизити витрати пам'яті, якщо одна й та сама стратегія знову використовується з тим самим пріоритетом.

```
1 reference
public async Task DownloadFileAsync(
    string url,
    string filePath,
    int priority,
    ProgressBar progressBar,
    Label progressLabel,
    CancellationToken cancellationToken)
{
    if (string.IsNullOrEmpty(url) || !Uri.IsWellFormedUriString(url, UriKind.RelativeOrAbsolute))
    {
        throw new ArgumentException("Invalid URL");
    }

    var speedStrategy = _strategyFactory.CreateSpeedStrategy(priority);
    int maxBytesPerSecond = speedStrategy.GetSpeedLimit();
```

Рис. 6 - Використання стратегії в класі DownloadFacade

У класі DownloadFacade створюється стратегія на основі пріоритету завантаження, переданого в параметрах. Потім ця стратегія використовується для визначення ліміту швидкості завантаження. Це дозволяє змінювати ліміт швидкості завантаження без зміни логіки самого завантаження, лише змінюючи стратегію.

Висновок

Шаблон Strategy дозволяє змінювати алгоритм ліміту швидкості для завантаження файлів, не змінюючи інших частин коду. Це досягається за допомогою інтерфейсу ISpeedStrategy, конкретних класів стратегій, а також заводу для створення стратегій в залежності від пріоритету.