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 №6

ШАБЛОНИ «Abstract Factory»,«Factory Method»,
«Memento», «Observer», «Decorator»

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

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

Завдання

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їх взаємодій 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

1. Abstract Factory (Абстрактна фабрика)

Abstract Factory — це шаблон проектування, що дозволяє створювати сімейства пов'язаних або залежних об'єктів без вказівки їх конкретних класів. Це корисно, коли система повинна бути незалежною від того, як створюються, компонуються та представлені її об'єкти. Шаблон дозволяє в будь-який момент змінювати створювані об'єкти, зберігаючи інтерфейс.

Основні компоненти:

Абстрактна фабрика (AbstractFactory): надає методи для створення абстрактних продуктів.

Конкретна фабрика (ConcreteFactory): реалізує методи для створення конкретних продуктів.

Абстрактний продукт (AbstractProduct): визначає інтерфейс для створюваних продуктів.

Конкретний продукт (ConcreteProduct): реалізує інтерфейс абстрактного продукту.

2. Factory Method (Фабричний метод)

Factory Method — це шаблон проектування, який визначає інтерфейс для створення об'єктів, але дозволяє підкласам змінювати тип створюваного об'єкта. Це дає можливість інкапсулювати процес створення об'єктів в одному методі, що спрощує зміну створюваних класів без змін у коді, що використовує ці об'єкти.

Основні компоненти:

Продукт (Product): об'єкт, який створюється фабрикою.

Creator: абстрактний клас або інтерфейс, що визначає метод для створення продукту.

ConcreteCreator: клас, що реалізує метод створення конкретного продукту.

3. Memento (Мементо)

Memento — це шаблон проектування, який дозволяє зберігати та відновлювати попередній стан об'єкта без порушення інкапсуляції. Він надає можливість зберегти стан об'єкта у момент часу, щоб пізніше відновити його, не змінюючи внутрішні дані об'єкта.

Основні компоненти:

Originator: об'єкт, чий стан зберігається. Він створює мементо та відновлює його.

Memento: об'єкт, що містить відображення стану Originator.

Caretaker: об'єкт, який зберігає мементо та може запитати його для відновлення стану, але не змінює сам стан.

4. Observer (Спостерігач)

Observer — це шаблон проектування, який визначає залежність типу "один до багатьох" між об'єктами таким чином, що коли один об'єкт змінює свій стан, всі його залежні об'єкти автоматично оновлюються. Цей шаблон використовується для реалізації механізму повідомлення, наприклад, в системах подій.

Основні компоненти:

Subject (Тема): об'єкт, який містить стан і змінюється. Після зміни стану, він повідомляє всіх спостерігачів.

Observer (Спостерігач): об'єкт, який підписаний на зміни в об'єкті Subject.

ConcreteSubject: реалізує методи для підписки та відписки спостерігачів, а також для інформування спостерігачів про зміни.

5. Decorator (Декоратор)

Decorator — це шаблон проектування, який дозволяє динамічно додавати нову поведінку об'єктам, обгортаючи їх в об'єкти декоратора. Це альтернатива підкласуванню, дозволяючи гнучко розширювати функціональність об'єкта, не змінюючи його код.

Основні компоненти:

Component: інтерфейс або абстрактний клас, який визначає основну поведінку об'єкта.

ConcreteComponent: реалізація основної поведінки об'єкта.

Decorator: абстрактний клас або інтерфейс, що розширює Component і містить обгорнутий об'єкт.

ConcreteDecorator: конкретні класи, які додають нову поведінку до обгорнутого об'єкта.

Хід роботи

Реалізація шаблону **Observer**

Шаблон Observer (Спостерігач) використовується для організації механізму, де один об'єкт (суб'єкт) сповіщає інші об'єкти (спостерігачів) про зміни в його стані. Це дозволяє зберігати слабку зв'язність між компонентами, тобто спостерігачі можуть реагувати на зміни суб'єкта без необхідності знати конкретну реалізацію цього суб'єкта. Це особливо корисно в системах, де кілька компонентів повинні реагувати на зміни в стані іншого об'єкта.

У вашому прикладі використано шаблон Observer для відстеження прогресу завантаження файлів за допомогою DownloadManager та спостерігачів, таких як ProgressBarObserver і LabelObserver.

```
6 references
public interface IObserver
{
    3 references
    void Update(DownloadState state);
}
```

Рис. 1 - Інтерфейс IObserver

Це інтерфейс для спостерігачів. Він визначає метод Update(DownloadState state), який буде викликатися при зміні стану суб'єкта. У цьому випадку метод отримує новий стан завантаження (DownloadState), який містить такі параметри як прогрес, повідомлення про статус, чи завершено завантаження або скасовано.

```
8 references
public class DownloadState
{
    4 references
    public double Progress { get; set; }
    5 references
    public string StatusMessage { get; set; }
    2 references
    public bool IsCompleted { get; set; }
    2 references
    public bool IsCanceled { get; set; }
}
```

Рис. 2 - Клас DownloadState

Це клас, що зберігає стан завантаження. Він містить такі властивості як Progress (прогрес завантаження в відсотках), StatusMessage (повідомлення про статус) та інші (завершено чи скасовано).

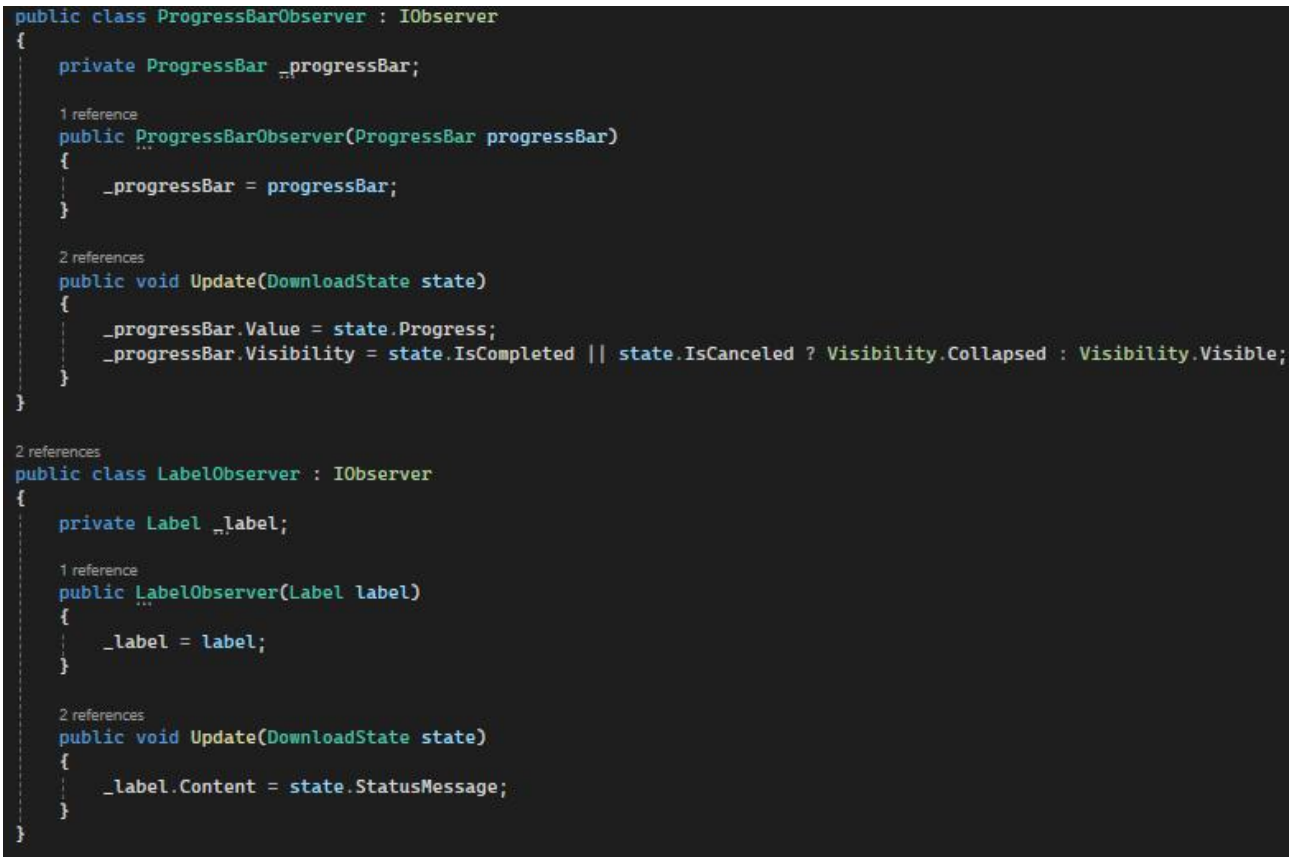
```
4 references
public class DownloadManager
{
    private readonly List<IObserver> _observers = new List<IObserver>();

    2 references
    public void Attach(IObserver observer) => _observers.Add(observer);
    0 references
    public void Detach(IObserver observer) => _observers.Remove(observer);

    4 references
    public void Notify(DownloadState state)
    {
        foreach (var observer in _observers)
        {
            observer.Update(state);
        }
    }
}
```

Рис. 3 - Клас DownloadManager

Цей клас діє як суб'єкт, що управляє спостерігачами. Він дозволяє додавати та видаляти спостерігачів через методи Attach та Detach. Коли стан змінюється (наприклад, під час завантаження файлу), метод Notify сповіщає всіх підписаних спостерігачів про зміни.



```
public class ProgressBarObserver : IObserver
{
    private ProgressBar _progressBar;

    1 reference
    public ProgressBarObserver(ProgressBar progressBar)
    {
        _progressBar = progressBar;
    }

    2 references
    public void Update(DownloadState state)
    {
        _progressBar.Value = state.Progress;
        _progressBar.Visibility = state.IsCompleted || state.IsCanceled ? Visibility.Collapsed : Visibility.Visible;
    }
}

2 references
public class LabelObserver : IObserver
{
    private Label _label;

    1 reference
    public LabelObserver(Label label)
    {
        _label = label;
    }

    2 references
    public void Update(DownloadState state)
    {
        _label.Content = state.StatusMessage;
    }
}
```

Рис. 4 - Спостерігачі (ProgressBarObserver, LabelObserver)

ProgressBarObserver: Оновлює значення прогрес-бару на основі прогресу завантаження. Він отримує оновлений стан і змінює значення властивості Value прогрес-бару.

LabelObserver: Оновлює текст на мітці для показу повідомлення про статус завантаження.

Таким чином, патерн Observer дозволяє розділити логіку оновлення інтерфейсу користувача від основної логіки завантаження, що робить код більш гнучким і підтримуваним.

Висновок

Шаблон Observer забезпечує ефективну організацію механізму спостереження, коли один об'єкт повідомляє інші про зміни в своєму стані. У нашому випадку він дозволяє оновлювати інтерфейс завантаження файлів, зберігаючи слабку зв'язність між компонентами. Це підвищує гнучкість, підтримуваність і дозволяє легко додавати нові функціональності без змін у основному коді