



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 №5

ШАБЛони «ADAPTER», «BUILDER», «COMMAND»,
«CHAIN OF RESPONSIBILITY», «PROTOTYPE»

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 Command	4
Висновок	7

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 браузері (firefox, opera, internet explorer, chrome) (https://github.com/Karmazinanastya/TRPZ_Karmazina_labs/tree/main/FileDownloader)

Теоретичні відомості

Adapter забезпечує взаємодію між класами з несумісними інтерфейсами, перетворюючи інтерфейс одного класу в інтерфейс, зрозумілий іншому.

Builder розділяє процес створення складного об'єкта на окремі кроки, дозволяючи поступово збирати об'єкт, варіюючи його конфігурацію без зміни коду.

Command інкапсулює запити або операції в окремі об'єкти, дозволяючи параметризувати клієнтські класи і створювати черги операцій або скасовувати виконані дії.

Chain of Responsibility організовує послідовність обробників, через які проходить запит, поки його не буде оброблено відповідним обробником, що сприяє зменшенню жорсткої прив'язки між відправником і отримувачем.

Prototype дозволяє створювати нові об'єкти шляхом клонування існуючих, що

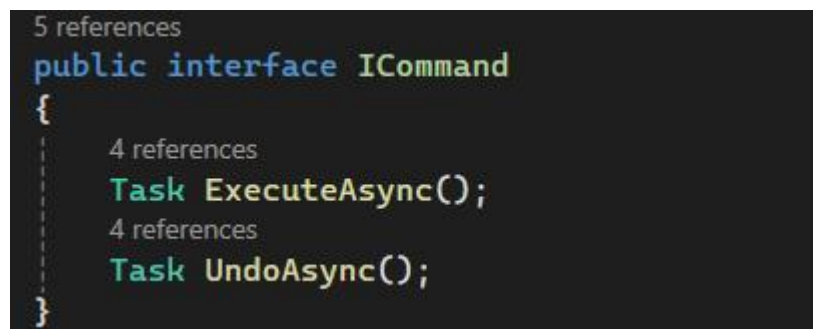
забезпечує ефективність створення складних об'єктів, мінімізуючи витрати на їх ініціалізацію.

Хід роботи

Реалізація шаблону **Command**

Шаблон Command використовується для інкапсуляції запитів або дій, які можуть бути виконані, та дозволяє їх відмінити (undo). Це дозволяє розділити обробку команд від коду, що виконує ці команди, і забезпечити гнучкість у виконанні та скасуванні операцій.

Шаблон Command реалізується через інтерфейс ICommand, а конкретні команди, такі як DownloadCommand, RemoveCommand і ViewHistoryCommand, реалізують цей інтерфейс, що дозволяє зберігати їх у стеку та виконувати або скасовувати за потреби.



```
5 references
public interface ICommand
{
    4 references
    Task ExecuteAsync();
    4 references
    Task UndoAsync();
}
```

Рис. 1 - Інтерфейс ICommand

Він містить два методи: ExecuteAsync() (для виконання команди) та UndoAsync() (для скасування команди). Це дозволяє абстрагувати реальне виконання конкретної дії від її виклику.

Кожна команда, наприклад, DownloadCommand або RemoveCommand, містить логіку для виконання конкретної дії, не залучаючи інтерфейс користувача або інші елементи системи. Це дозволяє зберігати код у чистому і підтримуваному вигляді.

```

public class DownloadCommand : ICommand
{
    private readonly DownloadListBoxItem _item;

    2 references
    public DownloadCommand(DownloadListBoxItem item)
    {
        _item = item;
    }

    2 references
    public async Task ExecuteAsync()
    {
        await _item.DownloadAsync();
    }

    2 references
    public Task UndoAsync()
    {
        _item.TokenSource?.Cancel();
        return Task.CompletedTask;
    }
}

```

Рис. 2 - Клас DownloadCommand

У цьому прикладі команда DownloadCommand інкапсулює всю логіку для ініціювання та скасування завантаження.

Кожна команда підтримує метод UndoAsync, що дозволяє скасувати виконану дію, зокрема скасувати завантаження або відновити елемент у списку після видалення.

Наприклад, після видалення елемента з колекції, його можна повернути за допомогою методу UndoAsync.

```

2 references
public class RemoveCommand : ICommand
{
    private readonly DownloadItemsCollection _itemsCollection;
    private readonly DownloadListBoxItem _item;

    1 reference
    public RemoveCommand(DownloadItemsCollection itemsCollection, DownloadListBoxItem item)
    {
        _itemsCollection = itemsCollection;
        _item = item;
    }

    2 references
    public Task ExecuteAsync()
    {
        _itemsCollection.Remove(_item.RemoveButton);
        return Task.CompletedTask;
    }

    2 references
    public Task UndoAsync()
    {
        _itemsCollection.Add(_item);
        return Task.CompletedTask;
    }
}

```

Рис. 3 - Клас RemoveCommand

Команда ViewHistoryCommand дозволяє відобразити історію виконаних операцій в інтерфейсі користувача, інкапсулюючи всі дії, пов'язані з цією операцією. Вона використовує шаблон Command, що дає змогу забезпечити чітке розділення логіки перегляду історії від інших частин програми, забезпечити гнучкість і підтримку цього функціоналу в майбутньому.

```
2 references
public class ViewHistoryCommand : ICommand
{
    private readonly List<History> _historyList;

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

    2 references
    public Task ExecuteAsync()
    {
        var historyWindow = new HistoryWindow(_historyList);
        historyWindow.Show();
        return Task.CompletedTask;
    }

    2 references
    public Task UndoAsync()
    {
        // Немає необхідності в Undo
        return Task.CompletedTask;
    }
}
```

Рис. 4 - Клас ViewHistoryCommand

Шаблон Command дозволяє створювати історію виконаних команд. Це реалізовано в класі CommandInvoker, який зберігає виконані команди в стосі і дозволяє скасувати останню виконану команду.

```
public class CommandInvoker
{
    private readonly Stack<ICommand> _commandHistory = new();

    4 references
    public async Task ExecuteCommandAsync(ICommand command)
    {
        await command.ExecuteAsync();
        _commandHistory.Push(command);
    }

    1 reference
    public async Task UndoLastCommandAsync()
    {
        if (_commandHistory.Count > 0)
        {
            var command = _commandHistory.Pop();
            await command.UndoAsync();
        }
    }
}
```

Рис. 5 - Клас CommandInvoker

Висновок

Шаблон Command був вибраний для цього проекту, оскільки він дозволяє зручно і модульно обробляти різні дії користувача, підтримувати скасування операцій, зберігати історію виконаних дій і спрощувати взаємодію з інтерфейсом користувача. Це допомагає покращити підтримуваність і масштабованість проекту.