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

Лабораторна робота №5
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»»**
Download manager

Виконала:
Студентка групи ІА-22
Степанюк-Боримська А. І.

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:
ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Мета:
Ознайомитися з основними шаблонами проєктування, такими як «Adapter», «Builder», «Command», «Chain of Responsibility», «Prototype», вивчити їхні принципи роботи та навчитись застосовувати для створення гнучкого та масштабованого програмного забезпечення в загальній розробці програмних систем.

Хід роботи

1. Реалізувати не менше 3-х класів відповідно до обраної теми

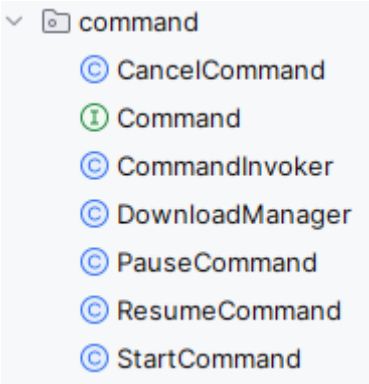

Рис. 1 — Структура проекту

Під час виконання лабораторної роботи було розроблено сім класів, які реалізують функціонал менеджера завантажень (рис. 1). Нижче наведено детальний опис кожного з цих класів:

1. **Command**
   Опис:
   Інтерфейс, що визначає метод execute(), який має бути реалізований всіма конкретними командами. Це забезпечує абстракцію, яка дозволяє визначати загальну структуру для всіх команд незалежно від їх конкретної реалізації.
   Призначення:
   Усі конкретні команди (наприклад, почати, призупинити, відновити завантаження) повинні реалізувати цей інтерфейс для виконання відповідних операцій.
2. **StartDownloadCommand**

Опис:

Конкретна команда для початку завантаження. Цей клас реалізує метод execute(), викликаючи метод startDownload() в об'єкті DownloadManager.

Призначення:

Запуск поточного завантаження, змінюючи статус завантаження на "в процесі". Команда інтерфейсу Command забезпечує абстракцію для цього процесу.

3. **PauseDownloadCommand**

Опис:

Конкретна команда для призупинення завантаження. Цей клас реалізує метод execute(), викликаючи метод pauseDownload() в об'єкті DownloadManager.

Призначення:

Призупиняє поточне завантаження, змінюючи його статус знову на "очікується". Команда гарантує, що операція призупинення буде коректно виконана.

4. **ResumeDownloadCommand**

Опис:

Конкретна команда для відновлення завантаження. Цей клас реалізує метод execute(), викликаючи метод resumeDownload() в об'єкті DownloadManager.

Призначення:

Відновлює завантаження, змінюючи його статус на "в процесі", якщо воно було призупинене. Команда відповідає за логіку відновлення.

5. **CancelDownloadCommand**

Опис:

Конкретна команда для скасування завантаження. Цей клас реалізує метод execute(), викликаючи метод cancelDownload() в об'єкті DownloadManager.

Призначення:

Скасовує поточне завантаження, змінюючи його статус на "скасовано". Команда гарантовано відміняє завантаження і коректно оновлює його статус.

6. **DownloadManager**

Опис:

Клас, який безпосередньо керує процесами завантаження. Він містить методи для початку, призупинення, відновлення і скасування завантажень. Реалізує бізнес-логіку для кожної з операцій.

Призначення:

Виконує дії над об'єктами завантажень, змінюючи їхні стани. Він є рецівером у патерні команд, оскільки виконання всіх операцій контролюється ним.

7. **CommandInvoker**

Опис:

Клас, який виконує команди за допомогою інтерфейсу Command. Він зберігає історію виконаних команд і може запускати їх за потреби.

Призначення:

Інвокер зберігає і виконує команди. Він відповідає за збереження історії команд, щоб мати змогу їх повторно виконати (якщо це необхідно). Інвокер також є посередником, який викликає методи виконання команд на основі вхідних запитів.

2. Реалізувати один з розглянутих шаблонів за обраною темою

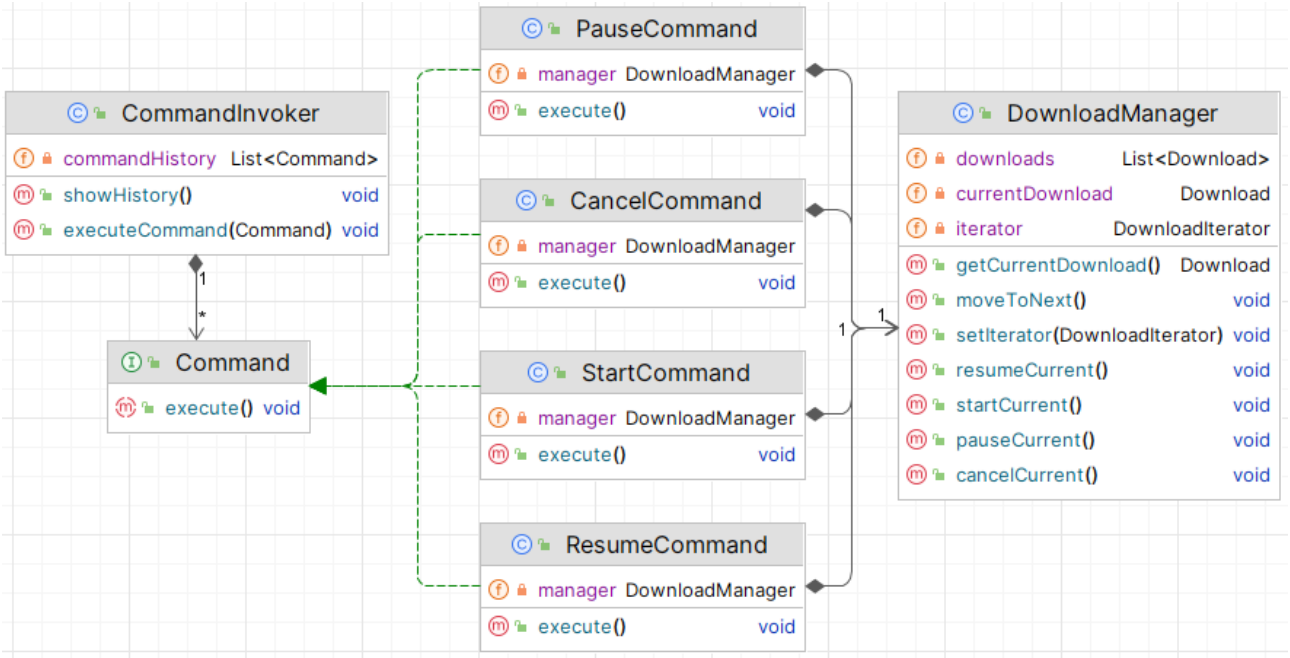

Рис. 2 — Діаграма класів

У контексті проекту менеджера завантажень, патерн Command реалізовано для спрощення управління операціями над завантаженнями та забезпечення гнучкості в їх виконанні. Патерн дозволяє розділити логіку виконання команд (наприклад, запуску, паузи, відновлення, скасування завантажень) і їх виклику в окремі компоненти, що дає низку переваг для розробки та підтримки програми.

Як реалізований патерн Command

1. Інтерфейс Command визначає загальну структуру для всіх операцій, що можуть бути виконані над завантаженнями (початок, пауза, відновлення, скасування). Кожна команда реалізує цей інтерфейс і забезпечує метод для виконання конкретної дії.
2. Команди: Кожна конкретна команда (StartDownloadCommand, PauseDownloadCommand, ResumeDownloadCommand, CancelDownloadCommand) відповідає за конкретну операцію над поточним завантаженням. Це дозволяє централізувати логіку операцій у відповідних класах, що полегшує їх тестування та змінення.
3. Рецівер (Receiver): Клас DownloadManager виступає як рецівер, який фактично виконуватиме дії над завантаженнями. Цей клас містить бізнес-логіку, необхідну для кожної операції: старт, пауза, відновлення та скасування завантаження.
4. Інвокер (Invoker): Клас CommandInvoker викликає відповідні команди, зберігає їх виконання в історії та дозволяє здійснювати маніпуляції з виконаними командами. Інвокер не піклується про те, як саме виконуються команди; він лише ініціює їх виконання.

Проблеми, які вирішує патерн Command

1. Ізоляція логіки: Завдяки використанню команд, бізнес-логіка (DownloadManager) відокремлена від логіки виконання самих операцій (виконання команд). Це дозволяє змінювати або додавати нові операції без необхідності змінювати існуючий код.
2. Гнучкість та розширюваність: Легко додавати нові команди для нових операцій (наприклад, продовження завантаження, зміна швидкості завантаження), не змінюючи основну структуру програми. Додавання нових команд займає мінімум часу і не вимагає переписування інших компонентів.
3. Повторне використання команд: Команди можна зберігати та виконувати повторно через інвокер. Це зручно, якщо потрібно скасувати чи відновити попередньо виконані операції.
4. Зменшення залежностей: Взаємодія між об'єктами зводиться до мінімуму, оскільки інвокер лише викликає виконання команд. Клас DownloadManager не залежить від конкретної команди або її реалізації, що робить систему більш модульною.

5. Легкість тестування та підтримки: Кожну команду можна протестувати окремо, що дозволяє перевірити її поведінку в ізольованих умовах. Якщо потрібно змінити логіку однієї операції, це можна зробити без впливу на інші частини програми.

Переваги використання

1. Розширюваність: Патерн Command дозволяє легко додавати нові операції (команди) для маніпулювання завантаженнями без змін у загальній структурі програми. Це особливо корисно в проектах, де функціонал може змінюватися або розширюватися в майбутньому.
2. Чистота коду: Завдяки розділенню відповідальностей між командами, інвокером і рецівером, код стає більш зрозумілим і підтримуваним. Кожен клас виконує одну конкретну задачу, що спрощує його тестування та модифікацію.
3. Гнучкість у виконанні операцій: Користувач може здійснювати операції над завантаженнями незалежно один від одного, без необхідності знати, як саме вони реалізуються. Команди можуть бути виконані в будь-якому порядку або навіть скасовані, що дає високу ступінь контролю над процесом завантаження.
4. Збереження історії операцій: Використання інвокера дозволяє зберігати історію виконаних команд, що може бути корисно для аналізу дій користувача, відновлення попереднього стану завантажень чи підтримки додаткових функцій, як, наприклад, скасування останніх дій.
5. Зниження складності при керуванні багатьма завантаженнями: Кожна команда працює над поточним завантаженням, що робить керування багатьма завантаженнями простішим та зручнішим. Це дозволяє зменшити загальну складність коду, оскільки взаємодія з поточним завантаженням відбувається через інтерфейс команд.

Перевірка роботи

```
public class DownloadManagerApp {
    public static void main(String[] args) {
        Download file1 = new Download( fileName: "file1.zip",  url: "https://example.com/file1.zip",  fileSize: 100, DownloadStatus.PENDING,  progress: 0);
        Download file2 = new Download( fileName: "file2.zip",  url: "https://example.com/file2.zip",  fileSize: 200, DownloadStatus.PENDING,  progress: 0);

        DownloadManager manager = new DownloadManager(Arrays.asList(file1, file2));
        CommandInvoker invoker = new CommandInvoker();

        Command startCommand = new StartCommand(manager);
        Command pauseCommand = new PauseCommand(manager);
        Command resumeCommand = new ResumeCommand(manager);
        Command cancelCommand = new CancelCommand(manager);

        invoker.executeCommand(startCommand);
        invoker.executeCommand(pauseCommand);
        invoker.executeCommand(resumeCommand);
        invoker.executeCommand(cancelCommand);

        manager.moveToNext();

        invoker.executeCommand(startCommand);
    }
}
```

Рис. 3 — Перевірка роботи

Ми створили клас DownloadManagerApp, в якому ініціалізували два завантаження (file1.zip та file2.zip) з різними параметрами, такими як URL, розмір, статус та прогрес. Далі, ми використовували патерн Команда (Command) для керування цими завантаженнями. Ми створили команди для початку, паузи, відновлення та скасування завантажень, а також використовували об'єкт CommandInvoker для їх виконання.

```
Started downloading: file1.zip
Paused downloading: file1.zip
Resumed downloading: file1.zip
Cancelled downloading: file1.zip
Started downloading: file2.zip
```

Рис. 4 — Результат роботи

Результат демонструє успішну реалізацію патерну Команда. Виконання команд змінює статуси завантажень, що видно при виконанні методів executeCommand через інтерфейс команд. Зміни статусів відображають, що завантаження може бути кероване через окремі команди (початок, пауза, відновлення, скасування), що свідчить про правильну реалізацію цього патерну в коді.

Висновки:

У ході лабораторної роботи було реалізовано патерн Command для менеджера завантажень, що забезпечує гнучке управління операціями, розділяючи логіку команд та їх виконання. Це спрощує розширення функціоналу, знижує складність коду та покращує керування завантаженнями.