



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»»**
Flexible Automatical Tool

Виконав:
Студент групи ІА-22
Сидорін Д.О.

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:.....	3
Мета:	3
Завдання:.....	3
Хід роботи	3
1. Реалізувати не менше 3-х класів відповідно до обраної теми	3
2. Реалізувати один з розглянутих шаблонів за обраною темою	5
Перевірка патерну	7
Висновки:	8

Тема:

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

Мета:

Ознайомитися з основними шаблонами проектування, такими як «Adapter», «Builder», «Command», «Chain of Responsibility», «Prototype», дослідити їхні принципи роботи та навчитися використовувати їх для створення гнучкого та масштабованого програмного забезпечення.

Завдання:

Реалізувати частину функціоналу робочої програми автоматизації у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

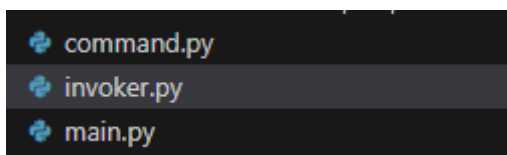
Хід роботи**1. Реалізувати не менше 3-х класів відповідно до обраної теми**

Рис. 1 — Структура проекту

1. Клас: Command (Інтерфейс)

Цей клас є абстрактним і визначає метод, який повинні реалізувати всі конкретні команди. Кожна команда має виконувати певну дію при виклику методу `execute`.

Методи:

- **`execute(self)` -> `None`:** Абстрактний метод, який повинні реалізувати всі конкретні команди. Він відповідає за виконання дії, пов'язаної з командою.

2. Клас: DownloadContentCommand (Конкретна команда для завантаження контенту)

Цей клас реалізує команду для завантаження контенту (фільмів, книг, файлів тощо). Він містить URL для завантаження та тип контенту.

Атрибути:

- **`_content_type`:** Тип контенту, який потрібно завантажити (наприклад, "фільм").

- **_content_url**: URL-адреса для завантаження контенту.

Методи:

- **__init__(self, content_type: str, content_url: str) -> None**: Конструктор класу, що ініціалізує атрибути контенту та URL.
- **execute(self) -> None**: Реалізація методу execute, що виводить повідомлення про завантаження контенту з зазначеного URL.

3. Клас: SetStatusCommand (Конкретна команда для зміни статусу в месенджерах)

Цей клас реалізує команду для зміни статусу в месенджерах (наприклад, Skype, Telegram). Він містить месенджер та новий статус.

Атрибути:

- **_messenger**: Назва месенджера, в якому змінюється статус (наприклад, "Skype").
- **_status**: Новий статус, який буде встановлено (наприклад, "Away").

Методи:

- **__init__(self, messenger: str, status: str) -> None**: Конструктор класу, що ініціалізує месенджер та статус.
- **execute(self) -> None**: Реалізація методу execute, що виводить повідомлення про встановлення нового статусу в зазначеному месенджері.

4. Клас: Invoker (Клас для управління командами)

Цей клас є "виконавцем" команд. Він відповідає за додавання команд до списку та їх виконання. Клас використовує команду для виконання певних завдань.

Атрибути:

- **_commands**: Список команд, які потрібно виконати.

Методи:

- **__init__(self)**: Конструктор класу, що ініціалізує список команд.
- **add_command(self, command: Command) -> None**: Метод для додавання команди до списку.
- **execute_commands(self) -> None**: Метод для виконання всіх команд у списку. Він викликає метод execute кожної команди.

Опис взаємодії класів:

1. Клас **Invoker** керує виконанням команд, додаючи їх до свого списку і викликаючи метод `execute` кожної команди.
2. Клас **DownloadContentCommand** виконує дію завантаження контенту, а клас **SetStatusCommand** змінює статус в месенджері.
3. Клас **Command** є базовим інтерфейсом, який гарантує, що всі конкретні команди реалізують метод `execute`.

Загальний потік роботи:

1. Створюється екземпляр класу **Invoker**.
2. До нього додаються конкретні команди, такі як **DownloadContentCommand** та **SetStatusCommand**.
3. Викликається метод `execute_commands`, який запускає метод `execute` для кожної з доданих команд.

2. Реалізувати один з розглянутих шаблонів за обраною темою

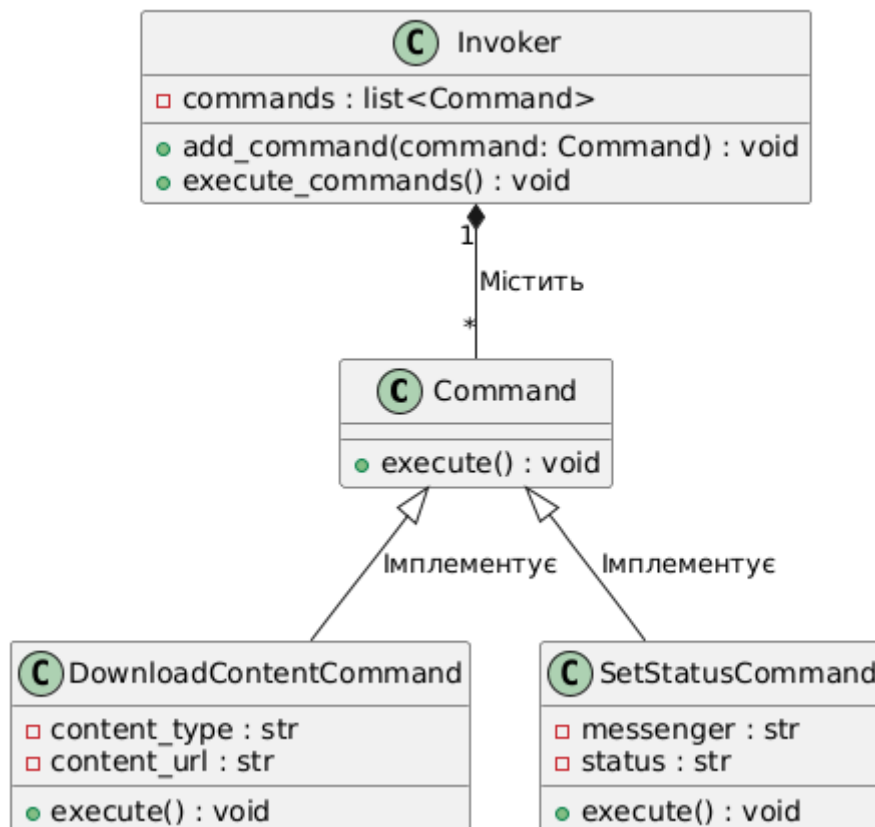


Рис. 2 — Діаграма класів

Ця діаграма класів описує структуру застосування патерну **Command** для автоматизації різних завдань у вашому додатку.

- **Command** — абстрактний клас, що визначає загальний інтерфейс для всіх конкретних команд. Кожна команда повинна реалізувати метод `execute()`, який виконує відповідну дію. Це дозволяє зберігати різні команди в однаковому форматі.
- **DownloadContentCommand** і **SetStatusCommand** — конкретні реалізації команд, які визначають, що саме буде виконано під час виклику методу `execute()`. Наприклад, в класі `DownloadContentCommand` реалізовано завантаження для завантаження контенту з певного URL, а в класі `SetStatusCommand` — для зміни статусу в певному месенджері.
- **Invoker** — клас, який утримує список команд і керує їх виконанням. Він надає методи для додавання команд (`add_command()`) і виконання всіх доданих команд (`execute_commands()`).

Зв'язки:

- Клас **Command** є базовим для двох конкретних команд: **DownloadContentCommand** та **SetStatusCommand**. Ці класи реалізують абстрактний метод `execute()` і виконують відповідні дії.
- **Invoker** містить список команд (асоціація типу "містить" з кількістю "*"), що дає змогу динамічно додавати та виконувати різні команди.

Проблеми, які вирішує патерн **Command**:

1. **Запис і виконання операцій з великою кількістю кроків:** Патерн **Command** дозволяє перетворювати складні операції (наприклад, завантаження контенту або зміну статусів) у команди, які можуть бути збережені, виконані або навіть відкладені на майбутнє.
2. **Інкапсуляція запитів на виконання дії:** Кожен конкретний клас команди інкапсулює одну операцію. Це дозволяє клієнтському коду працювати з абстракцією, замість того, щоб знати конкретні деталі виконання кожної операції.
3. **Підтримка незмінності команд:** Оскільки команди інкапсулюють дію, це дозволяє зробити систему більш гнучкою. Наприклад, можна створювати нові команди без змін в основному коді програми.
4. **Реалізація функціоналу скасування або повтору:** Команди можуть бути збережені у списку, і їх виконання можна відслідковувати, що дозволяє створювати функціонал скасування або повтору (наприклад, повернення до попереднього статусу).

Переваги використання патерну **Command**:

1. **Гнучкість і розширюваність:** Команди можна додавати або змінювати без необхідності змінювати код клієнтів, які взаємодіють з ними. Це дозволяє легше змінювати або розширювати функціональність програми, додаючи нові команди або змінюючи старі.

2. **Інкапсуляція операцій:** Патерн дозволяє інкапсулювати запити на виконання операцій, таким чином знижуючи залежності між об'єктами. Клієнт не потребує знань про те, як саме виконується операція, він лише має доступ до команди.
3. **Підтримка скасування і повтору дій:** Завдяки тому, що кожна команда є окремим об'єктом, можна створити механізм скасування (undo) або повтору (redo) виконаних операцій, що особливо корисно в застосунках з високим рівнем взаємодії.
4. **Легкість додавання нових функціональностей:** Додавати нові типи команд дуже просто, адже для цього достатньо створити новий клас, що реалізує інтерфейс **Command**, без зміни існуючої логіки додатку.
5. **Чистота коду і зниження складності:** Патерн Command допомагає зробити код більш чистим і розподіленим, що зменшує складність програми, особливо якщо в системі є багато різних типів операцій, які потрібно виконати в різних контекстах.

Ці переваги роблять патерн **Command** дуже корисним для систем автоматизації, де потрібна гнучкість у виконанні завдань та підтримка розширюваності.

Перевірка патерну

```
test_command_pattern.py X
test_command_pattern.py > ...
1  from command import DownloadContentCommand, SetStatusCommand
2  from invoker import Invoker
3
4  def test_command_pattern():
5      # Ініціалізація Invoker
6      invoker = Invoker()
7
8      # Створення команд
9      download_command = DownloadContentCommand("фільм", "https://example.com/movie.mp4")
10     set_status_command = SetStatusCommand("Skype", "Away")
11
12     # Додавання команд до Invoker
13     invoker.add_command(download_command)
14     invoker.add_command(set_status_command)
15
16     # Виконання команд
17     print("Виконання всіх команд:")
18     invoker.execute_commands()
19
20 if __name__ == "__main__":
21     test_command_pattern()
22
```

Рис. 3 — Перевірка роботи

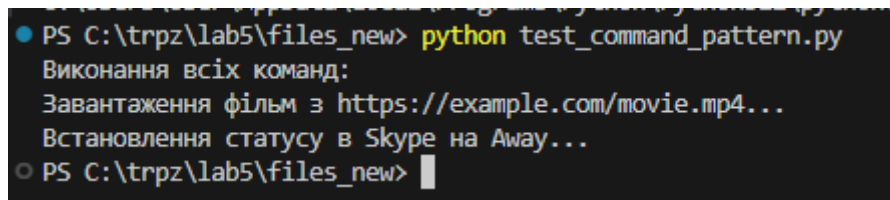
Для перевірки роботи патерну **Command** було створено тестовий сценарій, який реалізує та виконує кілька команд автоматизації. В тесті були використані дві основні команди:

1. **DownloadContentCommand** — команда для завантаження контенту (наприклад, фільмів чи книг) з URL.
2. **SetStatusCommand** — команда для встановлення статусу в месенджері (наприклад, "Skype").

У рамках тесту:

- Створюється об'єкт **Invoker**, який зберігає команди.
- Додаються команди до об'єкта **Invoker** за допомогою методу `add_command()`.
- Виконується виконання всіх доданих команд методом `execute_commands()`.

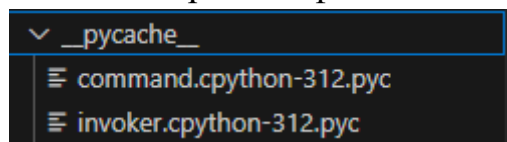
Після виконання команд, кожна команда виводить повідомлення в консоль про свою дію (завантаження контенту або встановлення статусу).



```
PS C:\trpz\lab5\files_new> python test_command_pattern.py
Виконання всіх команд:
Завантаження фільм з https://example.com/movie.mp4...
Встановлення статусу в Skype на Away...
PS C:\trpz\lab5\files_new>
```

Рис. 4 — Результат роботи

Утворились файли:



Результат роботи

Результат підтверджує успішну реалізацію патерну Command, оскільки кожна команда була виконана коректно і послідовно відповідно до її призначення. У `pycache` були створені системою файли які брегігають байт-код, він є більш ефективним для виконання і не потребує повторної компіляції під час кожного запуску програми

Висновки

Реалізован паттерн Command, було розроблено класи під тему, та перевірка, у якій ми побачили що все працює та виконується, також була розроблена діаграма класів шаблону.

Код: <https://github.com/Lepseich/trpz/tree/main/lab5>