



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:

Візуальний додаток для складання "карт пам'яті" з можливістю роботи з декількома картами (у вкладках), автоматичного промальовування ліній, додавання вкладених файлів, картинок, відеофайлів (попередній перегляд); можливість додавання значків категорій / терміновості, обведення областей карти (поділ пунктирною лінією).

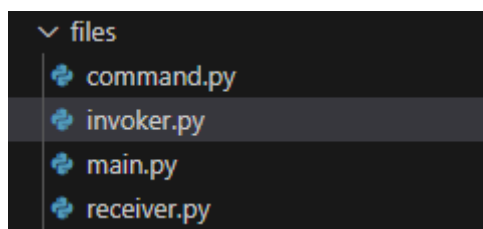
Хід роботи**1. Реалізувати не менше 3-х класів відповідно до обраної теми**

Рис. 1 — Структура проекту

Command

- **Опис:** Це абстрактний базовий клас для всіх команд. Він визначає загальний інтерфейс для конкретних команд. Усі команди повинні реалізувати метод `execute()`, який виконує дію відповідно до командного запиту.
- **Методи:**
 - `execute()` — абстрактний метод, який мають реалізувати всі конкретні команди для виконання відповідної дії.

2. UpdateStatusCommand

- **Опис:** Конкретна команда для оновлення статусу користувача в системі (наприклад, у месенджері чи комунікаторі). Ця команда отримує об'єкт `StatusUpdater` і статус, який потрібно встановити.
- **Методи:**
 - `execute()` — викликає метод `updateStatus()` об'єкта `StatusUpdater`, щоб оновити статус користувача.

3. DownloadFileCommand

- **Опис:** Конкретна команда для завантаження файлів з інтернету за вказаним URL. Вона приймає об'єкт FileDownloader і URL, з якого має бути завантажений файл.
- **Методи:**
 - execute() — викликає метод downloadFile() об'єкта FileDownloader, щоб завантажити файл з зазначеного URL.

4. ExecuteScriptCommand

- **Опис:** Конкретна команда для виконання скрипта. Вона отримує об'єкт ScriptExecutor і сам скрипт, який має бути виконаний.
- **Методи:**
 - execute() — викликає метод executeScript() об'єкта ScriptExecutor, щоб виконати переданий скрипт.

5. Invoker

- **Опис:** Клас, який керує командами та їх виконанням. Він зберігає список команд і відповідає за їх поетапне виконання. Invoker викликає команди, зберігаючи абстракцію від конкретних реалізацій.
- **Методи:**
 - add_command(command: Command) — додає команду до списку команд для подальшого виконання.
 - execute_commands() — виконує всі додані команди в порядку їх додавання.

6. FileDownloader

- **Опис:** Клас, який реалізує логіку завантаження файлів. Відповідає за фактичне завантаження файлів з інтернету через вказаний URL.
- **Методи:**
 - downloadFile(url: str) — завантажує файл за переданим URL.

7. StatusUpdater

- **Опис:** Клас, що відповідає за оновлення статусу користувача. Може бути застосований для оновлення статусів у різних комунікаторах чи месенджерах.
- **Методи:**
 - updateStatus(status: str) — оновлює статус користувача на нове значення.

8. ScriptExecutor

- **Опис:** Клас, який виконує скрипти або інші командні файли. Він інтерпретує або виконує передані скрипти.
- **Методи:**
 - `executeScript(script: str)` — виконує переданий скрипт.

2. Реалізувати один з розглянутих шаблонів за обраною темою

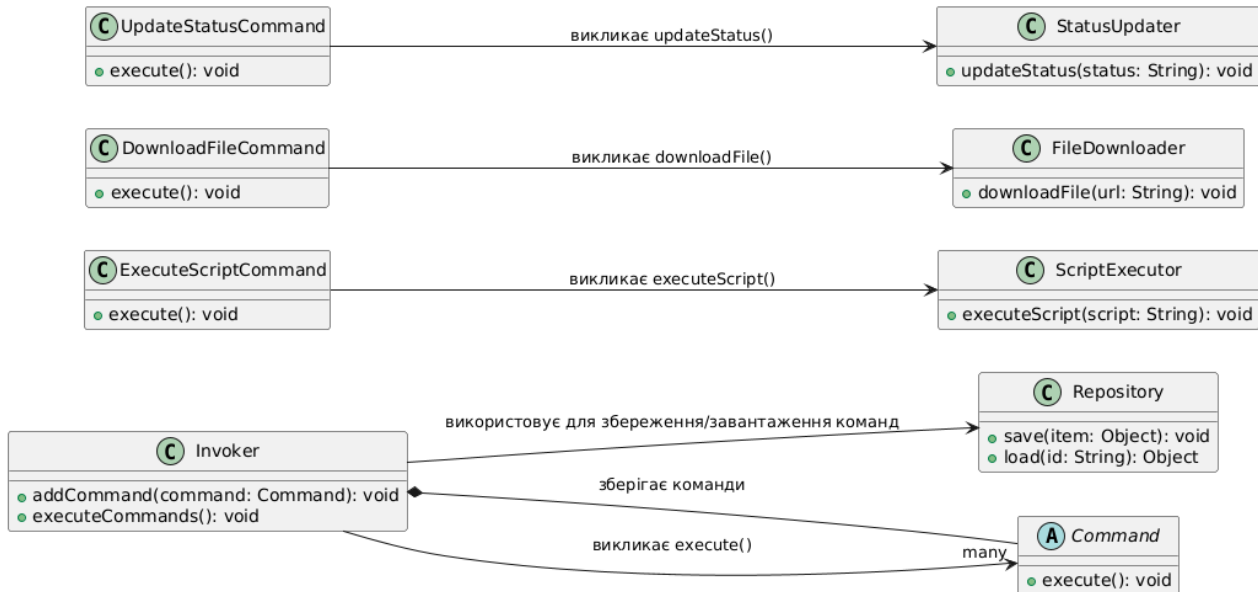


Рис. 2 — Діаграма класів

Патерн **Command** використовується для інкапсуляції запиту (команди) як об'єкта. Це дозволяє параметризувати методи клієнта різними запитами, ставити запити в чергу, реєструвати лог операцій, а також підтримувати відміну операцій (undo). У даному випадку патерн Command дозволяє створювати систему для автоматизації різних дій без необхідності тісно зв'язувати виконання команд з конкретними класами чи логікою. Патерн також забезпечує можливість виконання команд за графіком або умовами, що робить його потужним інструментом у системах автоматизації.

Проблеми, які вирішує патерн Command:

1. **Інкапсуляція команд:** Патерн Command дозволяє представити запити як об'єкти, що спрощує виконання дій, пов'язаних з автоматизацією. Запити, такі як "завантажити файл", "оновити статус" чи "виконати скрипт", можуть бути подані як об'єкти, що полегшує їх обробку, зберігання та чергування. Це значно зменшує залежність між компонентами системи.

2. **Гнучкість у виконанні команд:** Завдяки тому, що запити інкапсульовані у вигляді об'єктів, їх можна легко маніпулювати, змінювати порядок виконання, ставити в чергу або виконувати в різний час. Крім того, можна реалізувати можливість скасування команд (undo) або повторення (redo).
3. **Відокремлення ініціатора та виконавця:** За допомогою патерну Command ініціатор команди (наприклад, користувач, система або інвокер) не має прямої залежності від класу, що виконує команду. Це зменшує складність і зв'язність системи, оскільки ініціатор просто створює об'єкти команд і передає їх інвокеру для виконання.
4. **Можливість керування запитам:** Патерн дає можливість зберігати команди для їх виконання в майбутньому або їх чергування. Це важливо для автоматизаційних інструментів, де необхідно виконувати різноманітні завдання на основі конкретних умов або за розкладом (наприклад, автоматичне завантаження файлів чи оновлення статусу).
5. **Виконання команд за розкладом або умовами:** Можливість створення та виконання команд на основі певних умов або за графіком (наприклад, запуск певної команди о певній годині) додає гнучкості в систему автоматизації.

Переваги використання патерну Command:

1. **Гнучкість і розширюваність:** Патерн Command дозволяє зручніше додавати нові команди в систему, оскільки кожна нова команда просто реалізує інтерфейс `execute()`. Це значно спрощує розширення системи без потреби змінювати вже існуючі компоненти.
2. **Зниження залежностей між компонентами:** Всі компоненти, які виконують дії (як, наприклад, **StatusUpdater**, **FileDownloader**, **ScriptExecutor**), мають чітке розмежування через команди. Це знижує жорстку зв'язність між ініціатором і виконавцем команд, що дозволяє змінювати поведінку без змін основної логіки програми.
3. **Легкість у реалізації скасування операцій (Undo/Redo):** Якщо система потребує можливості скасування операцій, патерн Command забезпечує це простим шляхом: зберігаючи стани виконаних команд, можна реалізувати механізм скасування або відновлення команд.
4. **Зручне чергування та виконання за умовами:** Оскільки всі команди інкапсульовані як об'єкти, їх можна зберігати в черзі, виконувати за розкладом або при виконанні певних умов. Це зручно для автоматизації задач, таких як планування завдань, регулярне оновлення статусу або завантаження файлів.
5. **Реалізація складних сценаріїв через композицію команд:** Патерн Command дозволяє комбінувати прості команди для створення складніших сценаріїв. Це дозволяє створювати ефективнішу й складнішу бізнес-логіку, не збільшуючи складність кодової бази.

6. **Підтримка журналювання та аудит:** Команди можуть бути записані в журнал виконання, що дозволяє зберігати історію дій і мати змогу аналізувати та відновлювати їх у разі необхідності.
7. **Підвищена тестованість:** Кожну команду можна тестувати окремо, не турбуючись про зовнішні залежності, оскільки команда є незалежним об'єктом, що виконує конкретну дію.

Перевірка патерну

```
# Команди
class Command:
    def execute(self):
        pass

class UpdateStatusCommand(Command):
    def execute(self):
        print("Оновлення статусу: Відсутній")

class DownloadFileCommand(Command):
    def execute(self):
        print("Завантаження файлу з URL: https://example.com/file1")

class ExecuteScriptCommand(Command):
    def execute(self):
        print("Виконання скрипту: run_update.sh")

# Інвокер
class Invoker:
    def __init__(self):
        self.commands = []

    def add_command(self, command):
        self.commands.append(command)

    def execute_commands(self):
        for command in self.commands:
            command.execute()

# Тестування патерну Command
if __name__ == "__main__":
    invoker = Invoker()

    # Додавання команд до інвокера
    invoker.add_command(UpdateStatusCommand())
    invoker.add_command(DownloadFileCommand())
    invoker.add_command(ExecuteScriptCommand())

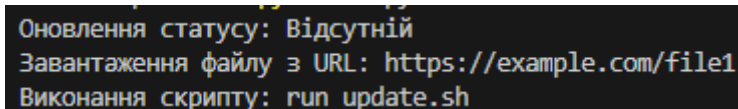
    # Виконання всіх команд
    invoker.execute_commands()
```

Рис. 3 — Перевірка роботи

У цьому коді реалізовано патерн **Command**, який дозволяє інкапсулювати дії у вигляді окремих об'єктів команд.

1. **Command** — базовий клас для команд з методом `execute()`, який буде реалізований у конкретних командах.
2. **Конкретні команди** (`UpdateStatusCommand`, `DownloadFileCommand`, `ExecuteScriptCommand`) реалізують метод `execute()` і виконують відповідні дії (оновлення статусу, завантаження файлу, виконання скрипту).
3. **Invoker** зберігає і виконує команди. Метод `add_command()` додає команди, а `execute_commands()` викликає їх виконання.

Результат: інвокер послідовно виконує всі додані команди, кожна з яких виконує конкретну дію. Це дозволяє легко додавати нові команди без змін у логіці інвокера.



```
Оновлення статусу: Відсутній  
Завантаження файлу з URL: https://example.com/file1  
Виконання скрипту: run_update.sh
```

Рис. 4 — Результат роботи

Результат роботи

Результат підтверджує успішну реалізацію патерну **Command**, оскільки кожна команда була виконана коректно і послідовно відповідно до її призначення. Це підтверджується виведеними повідомленнями для кожної команди: запис макросу, його зупинка та відтворення. Усі команди успішно виконали свої дії, що свідчить про правильну інтеграцію патерну **Command** у систему.

Висновки

Був реалізований патерн **Command**, який дозволяє інкапсулювати дії в об'єкти команд і виконувати їх через інвокера. Кожна команда реалізує метод `execute()`, який виконує конкретну операцію, наприклад, оновлення статусу, завантаження файлів або виконання скриптів. Інвокер зберігає команди і послідовно викликає їх виконання, що забезпечує гнучкість і можливість додавати нові команди без змін у основній логіці. Це підвищує масштабованість і знижує зв'язність компонентів системи.

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