



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	4
Перевірка патерну	6
Висновки:	7

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).

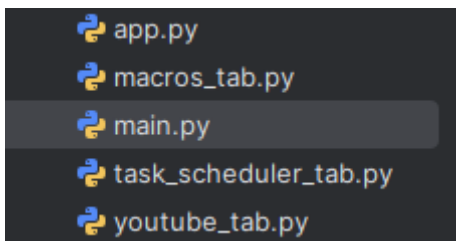
Хід роботи**1. Реалізувати не менше 3-х класів відповідно до обраної теми**

Рис. 1 — Структура проекту

У ході роботи було розроблено та модифіковано наступні класи:

1. **App** Клас **App** є основним класом програми, який ініціалізує графічний інтерфейс користувача та запускає додаток. Він відповідає за створення вікна додатка з вкладками для різних функцій (моніторинг YouTube, макроси, планувальник задач) та забезпечує їхнє відображення. Цей клас визначає вікно програми та керує розміщенням вкладок, що відповідають за окремі функціональності. Він має метод **create_tabs**, який ініціалізує вкладки для кожного з модулів. Клас **App** відповідає за загальне керування інтерфейсом і запуск програми.
2. **MacroTab** Клас **MacroTab** відповідає за роботу з макросами: запис, збереження та відтворення дій користувача. В цьому класі реалізовано запис дій користувача, таких як натискання клавіш та кліки миші, та їхнє збереження у файл. Патерн **Command** використовується для інкапсуляції кожної дії, наприклад, натискання клавіші або кліку миші. Це дозволяє

відтворити ці дії в будь-який час. Клас містить методи для запису макросів **record_macro**, зупинки запису **stop_recording** та відтворення макросу **play_macro**.

3. **YouTubeMonitorTab** Клас **YouTubeMonitorTab** є однією з вкладок додатка і реалізує функціонал моніторингу нових відео на YouTube. Він відповідає за підключення до API YouTube, перевірку наявності нових відео та їх автоматичне завантаження або інформування користувача. Цей клас дозволяє автоматизувати процес завантаження нових серій відео чи фільмів по мірі їхнього випуску, що є частиною загальної мети автоматизації задач користувача.

Ці класи сприяють розвитку додатка, який може автоматизувати різні завдання, включаючи моніторинг YouTube, запис макросів для автоматизації дій користувача та керування задачами через графічний інтерфейс.

2. Реалізувати один з розглянутих шаблонів за обраною темою

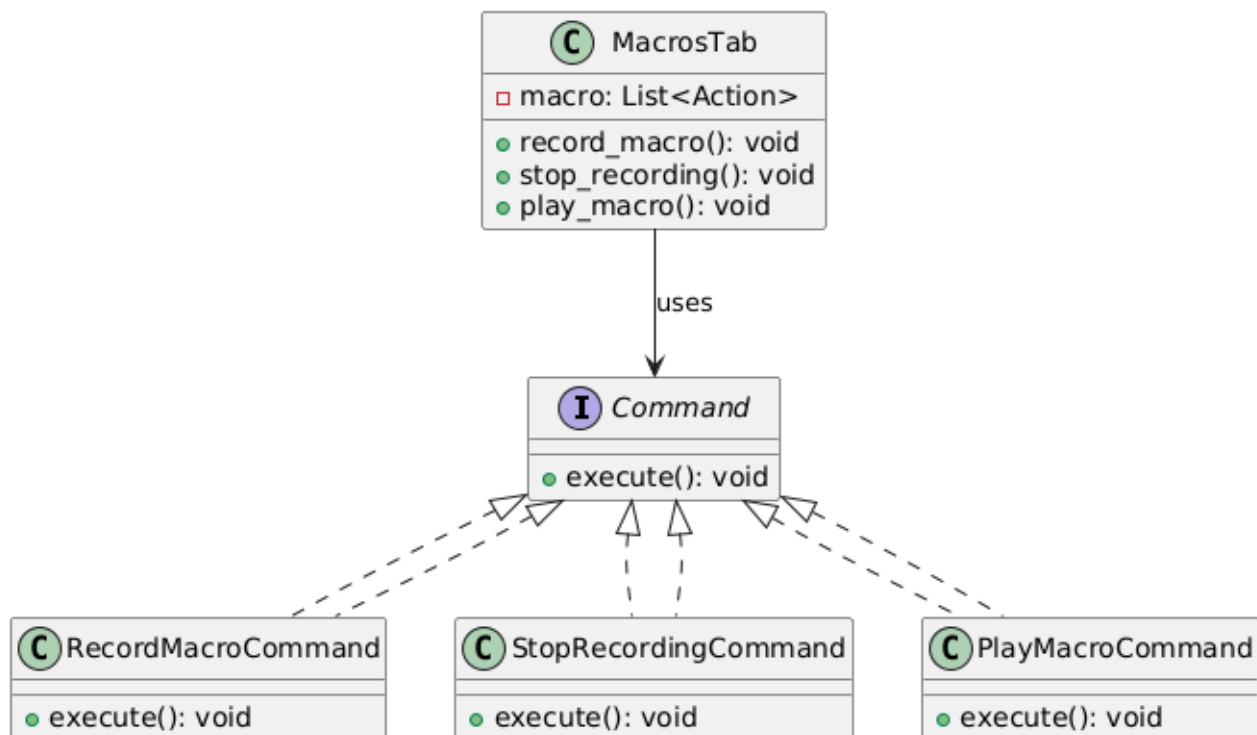


Рис. 2 — Діаграма класів

У контексті розробки інструменту автоматизації, патерн **Command** забезпечує чітке розмежування між виконавцями дій та ініціаторами цих дій. Це дозволяє легко додавати нові функціональні можливості, такі як запис, зупинка та відтворення макросів, без необхідності модифікації основного коду.

Проблеми, які вирішує патерн Command:

1. Інкапсуляція дій

Кожна дія, наприклад, запис макросу, зупинка запису або відтворення, представлена як окремий об'єкт-команда. Це дозволяє уникнути складності, пов'язаної з жорстко закодованою логікою в класах, спрощуючи управління функціоналом.

2. Розширюваність функцій

За допомогою патерну **Command** легко додавати нові команди, які інтегруються з існуючими механізмами, без необхідності змінювати поточну архітектуру програми. Наприклад, у майбутньому можна додати команди для додаткових дій або сценаріїв автоматизації.

3. Управління складними діями

Завдяки інкапсуляції дій у вигляді команд забезпечується підтримка багатопотокового виконання, наприклад, запис макросів через одночасний контроль за діями миші та клавіатури.

Переваги використання патерну Command:

1. Простота модифікації та повторного використання

Кожна команда є автономним модулем, що дозволяє її легко змінювати чи використовувати в інших частинах програми. Наприклад, команда для запису макросів може бути адаптована для запису подій у різних сценаріях.

2. Зменшення зв'язності компонентів

Клас `MacroTab` не залежить від деталей реалізації команд. Він викликає команди через інтерфейс **Command**, що сприяє більш гнучкій архітектурі програми.

3. Підтримка Undo/Redo (у перспективі)

Використання патерну **Command** відкриває можливість реалізації механізму скасування чи повторення дій, наприклад, скасування запису макросу або повторного виконання певної дії.

Реалізація в інструменті автоматизації:

1. Запис макросу (**RecordMacroCommand**)

Команда, яка ініціює захоплення подій клавіатури та миші, додає їх до масиву дій і дозволяє вести паралельний запис через багатопоточність.

2. Зупинка запису (**StopRecordingCommand**)

Команда, що завершує процес запису, зберігає зібрані події у файл і припиняє потоки, які відповідають за обробку подій.

3. Відтворення макросу (**PlayMacroCommand**)

Команда, яка завантажує записаний макрос з файлу та виконує дії у потрібній послідовності з врахуванням часу подій.

Таким чином, патерн **Command** в інструменті автоматизації дозволяє ефективно організувати архітектуру для управління діями, забезпечуючи розширюваність, простоту додавання нових функцій та гнучкість роботи з макросами.

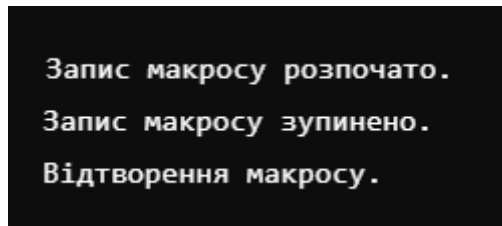
Перевірка патерну

```
1 class Command:
2     """Інтерфейс команди."""
3     def execute(self):
4         pass
5
6 class RecordMacroCommand(Command):
7     """Команда для запису макросу."""
8     def execute(self):
9         print("Запис макросу розпочато.")
10
11 class StopRecordingCommand(Command):
12     """Команда для зупинки запису макросу."""
13     def execute(self):
14         print("Запис макросу зупинено.")
15
16 class PlayMacroCommand(Command):
17     """Команда для відтворення макросу."""
18     def execute(self):
19         print("Відтворення макросу.")
20
21 class MacroController:
22     """Клас, що управляє командами."""
23     def __init__(self):
24         self.commands = []
25
26     def add_command(self, command: Command):
27         self.commands.append(command)
28
29     def execute_commands(self):
30         for command in self.commands:
31             command.execute()
32
33
34 if __name__ == "__main__":
35     controller = MacroController()
36
37     controller.add_command(RecordMacroCommand())
38     controller.add_command(StopRecordingCommand())
39     controller.add_command(PlayMacroCommand())
40
41
42
43 controller.execute_commands()
```

Рис. 3 — Перевірка роботи

- **Створено базовий інтерфейс Command, який визначає метод execute().** Усі команди реалізують цей метод, що дозволяє уніфіковано виконувати різні дії.
- **Реалізовано три конкретні команди:**
 - RecordMacroCommand: починає запис макросу.
 - StopRecordingCommand: зупиняє запис макросу.
 - PlayMacroCommand: відтворює записаний макрос.

□ **Клас MacroController забезпечує управління командами.**
Він збирає команди у список і дозволяє виконати їх у послідовності.



```
Запис макросу розпочато.  
Запис макросу зупинено.  
Відтворення макросу.
```

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

У ході цієї роботи ми реалізували патерн Command для автоматизації дій у додатку. Патерн дозволяє ефективно інкапсулювати логіку кожної окремої дії в окремий клас, забезпечуючи просте управління командами через їх об'єднання в один контролер. Це допомогло підвищити масштабованість та гнучкість системи, зокрема завдяки можливості додавання нових команд без змін існуючого коду.

Використання патерну Command значно спрощує розширення функціональності додатку для автоматизації дій, таких як запис, зупинка та виконання макросів, і гарантує структуровану обробку запитів.

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