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 №4
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»»**
Flexible Automatical Tool

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

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:
ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»
Мета:
Ознайомитися з основними шаблонами проєктування, такими як «Singleton», «Iterator», «Proxy», «State» та «Strategy», дослідити їхні принципи роботи та навчитися використовувати 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

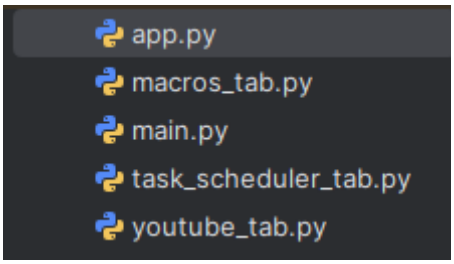

Рис. 1 — Структура проекту

Опис файлів і класів:

1. **main.py**
   Головний файл програми, в якому запускається додаток. Він ініціалізує основний графічний інтерфейс і забезпечує зв'язок між усіма модулями програми.
2. **app.py**
   У цьому файлі створюється додаток із трьома вкладками. Він відповідає за управління загальним інтерфейсом програми та організацію взаємодії між вкладками.
3. **macros_tab.py**
   Файл, який реалізує одну з вкладок програми.
   - **MacrosTab** — клас, що відповідає за запис макросів. Дозволяє користувачам записувати дії (натискання клавіш, рухи миші) та зберігати їх для подальшого використання. Ця функціональність реалізована як одна зі стратегій автоматизації завдань.
4. **task_scheduler.py**
   Файл, у якому реалізовано функціонал для додавання завдань на виконання за розкладом.

- **TaskScheduler** — клас, який дозволяє створювати, редагувати та запускати завдання у визначений користувачем час. Наприклад, це може бути роздача торрент-файлів або запуск інших автоматизованих дій.

5. **youtube_tab.py**
   Файл, що відповідає за функціонал пошуку та завантаження відео з YouTube.
   - **YouTubeTab** — клас, який дозволяє здійснювати пошук відео за URL каналу, отримувати доступ до нових відео та завантажувати їх. Реалізує функціонал автоматизації взаємодії з YouTube як окрему стратегію.

2. Реалізувати один з розглянутих шаблонів за обраною темою

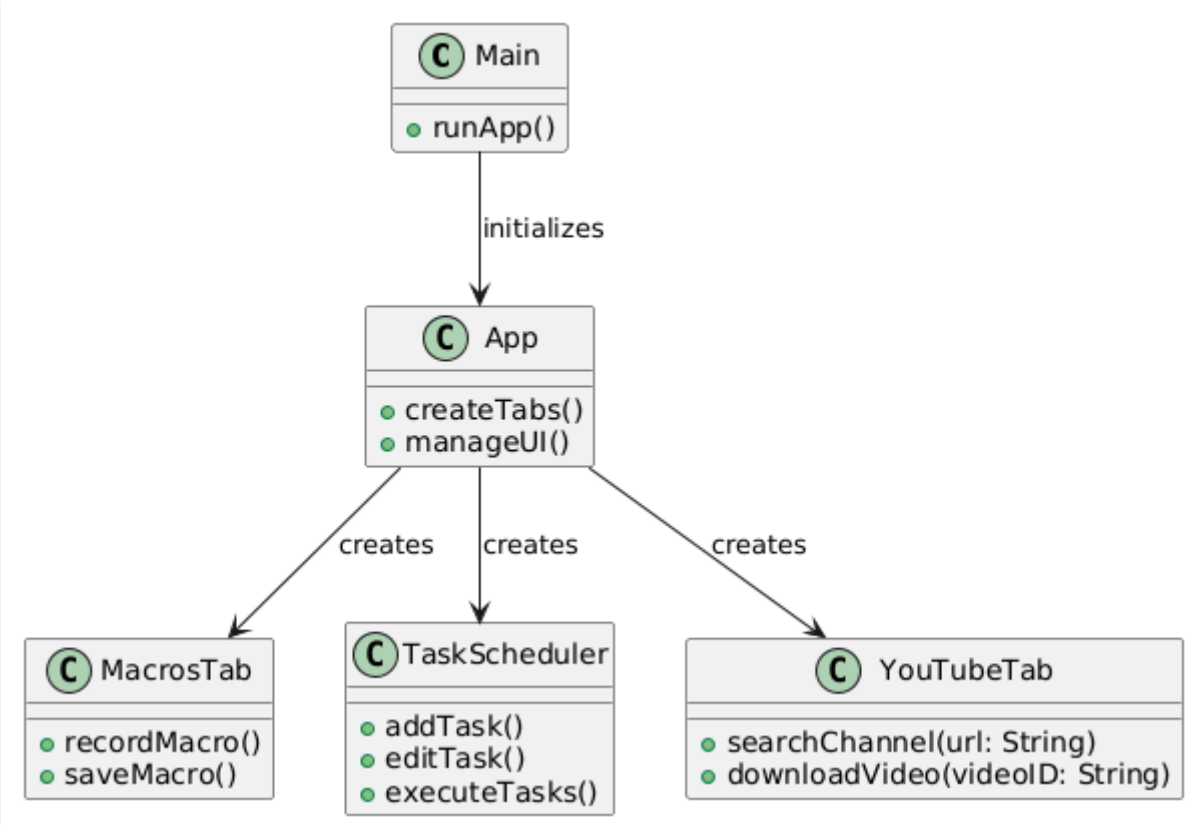

Рис. 2 — Діаграма класів

У нашому додатку для автоматизації задач ми реалізували патерн Strategy для динамічного вибору та зміни способу виконання автоматичних дій. Стратегія виконання завдань є важливим компонентом роботи додатку, що дозволяє налаштовувати різні сценарії, такі як моніторинг YouTube-каналів, запис і виконання макросів, або запуск запланованих дій. Це підвищує зручність і гнучкість роботи користувача з програмою.

Як це працює

Патерн Strategy дозволяє абстрагувати логіку виконання різних автоматичних дій у нашому додатку. Ми створили інтерфейс TaskExecutionStrategy, який визначає метод executeTask. Кожна конкретна стратегія (наприклад, моніторинг YouTube, запис макросів або планування задач) реалізує цей інтерфейс, надаючи власну реалізацію виконання задачі.

Клас AutomationTool виступає контекстом, у якому залежно від потреби встановлюється потрібна стратегія через метод setTaskExecutionStrategy. Коли необхідно виконати дію, AutomationTool викликає метод стратегії executeTask, не потребуючи знань про конкретну реалізацію стратегії.

Проблеми, які вирішує патерн Strategy

1. Гнучкість у виборі способу виконання:

Додаток підтримує кілька способів виконання автоматичних дій. Це дозволяє користувачеві обирати потрібну функцію, наприклад, налаштувати моніторинг YouTube-каналів чи автоматизувати повторення дій за допомогою макросів.

2. Зменшення залежностей:

Контекст AutomationTool не містить жодної конкретної логіки виконання, що робить його менш залежним від змін у реалізації стратегій.

3. Масштабованість:

Легко додавати нові способи автоматизації (нові стратегії), не змінюючи існуючий код контексту чи інших класів.

4. Розширення функціональності:

Користувачі отримують можливість інтерактивно обирати або змінювати спосіб автоматизації без необхідності змін у базовому функціоналі програми.

Переваги використання патерну

1. Інкапсуляція змін:

Логіка кожного способу автоматизації ізольована в окремих класах, що полегшує їх розуміння, тестування та підтримку.

2. Зменшення дублювання коду:

Кожна стратегія реалізує лише свій спосіб виконання, не дублюючи загальні частини.

3. Проста інтеграція нових стратегій:

Для додавання нового способу автоматизації потрібно лише створити нову реалізацію інтерфейсу TaskExecutionStrategy, не змінюючи інші частини програми.

4. Динамічний вибір поведінки:

Додаток дозволяє користувачеві або системі динамічно змінювати спосіб виконання залежно від контексту використання.

Завдяки реалізації патерну Strategy, наш додаток отримав гнучкість, що робить його більш зручним для користувачів і готовим до масштабування.

Перевірка патерну

main.py

Plugins supporting *.py files found.

```
from strategies import YouTubeMonitorStrategy, MacrosStrategy, TaskSchedulerStrategy
from automation_tool import AutomationTool

if __name__ == "__main__":

    automation_tool = AutomationTool()


    print("=== Використання стратегії моніторингу YouTube ===")
    automation_tool.set_task_execution_strategy(YouTubeMonitorStrategy())
    automation_tool.execute_task()


    print("\n=== Використання стратегії запису макросів ===")
    automation_tool.set_task_execution_strategy(MacrosStrategy())
    automation_tool.execute_task()


    print("\n=== Використання стратегії планувальника задач ===")
    automation_tool.set_task_execution_strategy(TaskSchedulerStrategy())
    automation_tool.execute_task()
```

Рис. 3 — Перевірка роботи

Опис

1. **Код створює об'єкт AutomationTool**, який виступає контекстом для виконання завдань.
2. **По черзі встановлюються різні стратегії**:
    - YouTubeMonitorStrategy для моніторингу YouTube-каналів.
    - MacrosStrategy для запису та виконання макросів.
    - TaskSchedulerStrategy для планування задач.
3. Для кожної стратегії викликається метод execute_task(), який виконує завдання відповідно до вибраної стратегії.

Вивід програми

Очікуваний результат у терміналі:

Виконується моніторинг YouTube-каналів...
Запис та відтворення макросів...
Запуск задачі за розкладом...

Код: https://github.com/Lepseich/trpz/tree/main/lab4