



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 №7
З дисципліни «Технології розроблення програмного забезпечення»
Тема: «**ШАБЛОНИ «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE
METHOD»»**
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 «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

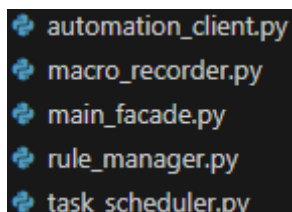
Мета:

Ознайомитися з основними шаблонами проектування, такими як «Mediator», «Facade», «Bridge», «Template Method», дослідити їхні принципи роботи та навчитися використовувати їх для створення гнучкого та масштабованого програмного забезпечення.

Завдання:

Інструмент автоматизації (стратегія, прототип, абстрактна фабрика, міст, композит, SOA)

Десктопний додаток для автоматизації повсякденних завдань із можливістю створення правил (аналогічно сервісу IFTTT), запису макросів (натискання клавіш, дії миші) та використання планувальника завдань. Додаток забезпечує функції, як-от автоматичне завантаження нових серій серіалів, книг чи інших файлів у визначений час, зміну статусів у месенджерах (наприклад, встановлення статусу "відсутній" у Skype при довгій неактивності), а також виконання завдань за розкладом (наприклад, запуск роздачі торрентів о 5 ранку).

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

```
automation_client.py
macro_recorder.py
main_facade.py
rule_manager.py
task_scheduler.py
```

Рис. 1 — Структура проекту

У ході роботи було розроблено наступні класи:

1. *MainFacade* (Фасад автоматизації)

Цей клас виконує роль центрального елементу автоматизаційної системи. Він об'єднує функціональність різних компонентів: менеджера правил (RuleManager), записувача макросів (MacroRecorder) та планувальника завдань (TaskScheduler). Клас дозволяє користувачеві через єдиний інтерфейс управляти всіма аспектами автоматизації, забезпечуючи простоту використання.

Основні методи:

- `add_rule(trigger, action)`: Додає нове правило через RuleManager.
- `record_macro(macro_name)`: Запускає запис макросу через MacroRecorder.

- `schedule_task(task_name, time)`: Планує завдання через TaskScheduler.
- `execute_all()`: Виконує всі активні правила, макроси та завдання.

2. *RuleManager (Менеджер правил)*

Клас відповідає за створення, зберігання і виконання правил автоматизації. Він реалізує принципи "Якщо-то" (trigger-action). Наприклад, завантаження серіалів у певний день тижня або зміна статусу в месенджері при неактивності користувача.

Основні методи:

- `create_rule(trigger, action)`: Дозволяє створити нове правило, вказуючи тригер та дію.
- `delete_rule(rule_id)`: Видаляє існуюче правило за його ідентифікатором.
- `get_rules()`: Повертає список усіх правил, які налаштовані в системі.
- `execute_rules()`: Перевіряє всі правила та виконує дії, якщо тригери спрацювали.

3. *MacroRecorder (Записувач макросів)*

Цей клас відповідає за запис і відтворення дій користувача, таких як натискання клавіш або рухи миші. Записані макроси можуть бути використані для автоматизації повторюваних задач.

Основні методи:

- `start_recording()`: Починає запис макросу.
- `stop_recording()`: Завершує запис і зберігає його у файлі.
- `play_macro(macro_name)`: Виконує записаний макрос за назвою.
- `get_macros()`: Повертає список доступних макросів.

4. *TaskScheduler (Планувальник завдань)*

Цей клас відповідає за виконання завдань у визначений час або за певним розкладом. Він дозволяє користувачам створювати повторювані та одноразові завдання.

Основні методи:

- `schedule_task(task_name, time)`: Планує завдання на певний час.
- `cancel_task(task_id)`: Видаляє завдання за його ідентифікатором.
- `execute_scheduled_tasks()`: Виконує всі завдання, час яких настав.
- `get_scheduled_tasks()`: Повертає список усіх запланованих завдань.

Ця структура дозволяє спростити роботу з автоматизацією, об'єднавши різні компоненти через єдиний фасад, що відповідає принципам паттерну **Facade**.

2. Реалізувати один з розглянутих шаблонів за обраною темою

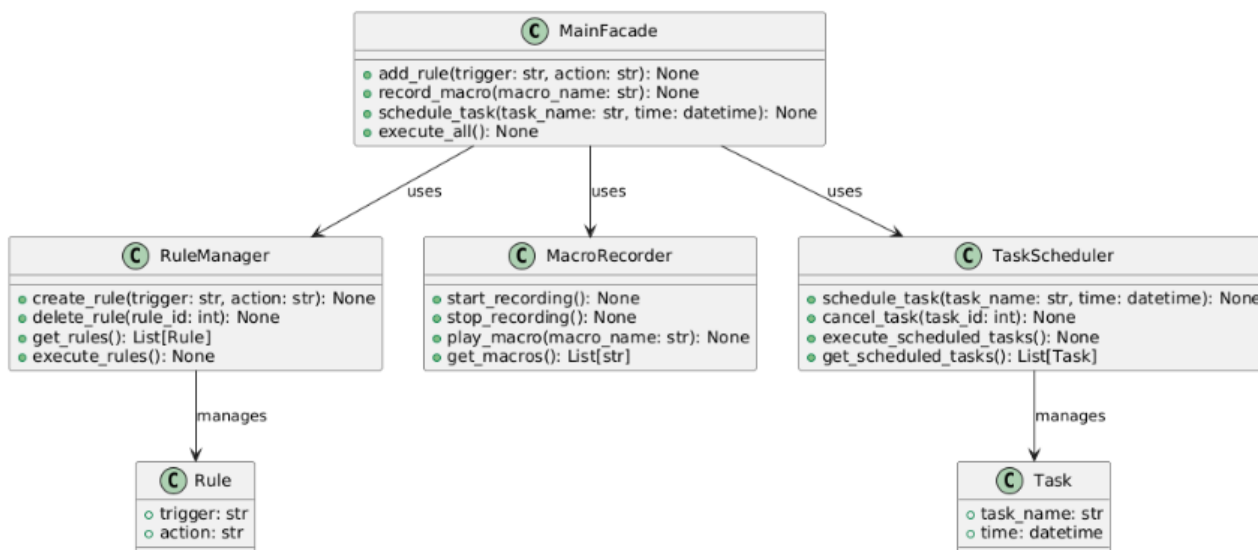


Рис. 2 — Діаграма класів

У проєкті автоматизації для спрощення управління базовими автоматичними діями було реалізовано патерн **Facade**, щоб об'єднати різні компоненти (менеджер правил, записувач макросів і планувальник завдань) в єдиний інтерфейс. Цей підхід дозволяє зменшити складність використання системи та забезпечити централізоване управління автоматизацією.

Проблеми, які вирішує патерн **Facade**:

1. Спрощування складності

Замість прямої взаємодії з кількома різними модулями (**RuleManager**, **MacroRecorder**, **TaskScheduler**), користувач працює з єдиним інтерфейсом **MainFacade**, що значно спрощує використання додатку.

2. Ізоляція компонентів

Фасад забезпечує чітке розділення між клієнтським кодом та внутрішньою реалізацією компонентів автоматизації, дозволяючи вносити зміни в окремі модулі без впливу на інші.

3. Централізоване управління

Фасад дозволяє об'єднати всі аспекти автоматизації, такі як управління правилами, запис і виконання макросів та запуск завдань за розкладом, в одному місці.

Переваги використання патерну **Facade**:

1. Простота використання

Завдяки фасаді, користувачі можуть виконувати всі автоматизаційні дії

через єдиний інтерфейс, що робить систему інтуїтивно зрозумілою навіть для новачків.

2. **Зменшення залежностей**

Основний клієнтський код взаємодіє лише з фасадом, а не з внутрішніми модулями, що знижує залежність між компонентами та покращує підтримку коду.

3. **Гнучкість розширення**

Нові функції автоматизації можна додавати, змінюючи або розширюючи модулі (RuleManager, MacroRecorder, TaskScheduler), без необхідності змінювати клієнтський код.

Перевірка патерну

```
class RuleManager:
    def create_rule(self, trigger, action):
        print(f"Rule created: WHEN '{trigger}' THEN '{action}'")

    def execute_rules(self):
        print("Executing all rules...")

class MacroRecorder:
    def record_macro(self, name, actions):
        print(f"Macro '{name}' recorded with actions: {actions}")

    def play_macro(self, name):
        print(f"Playing macro '{name}'")

class TaskScheduler:
    def schedule_task(self, task_name, time):
        print(f"Task '{task_name}' scheduled for {time}")

    def execute_tasks(self):
        print("Executing all scheduled tasks...")

class MainFacade:
    def __init__(self):
        self.rule_manager = RuleManager()
        self.macro_recorder = MacroRecorder()
        self.task_scheduler = TaskScheduler()

    def add_rule(self, trigger, action):
        self.rule_manager.create_rule(trigger, action)

    def record_macro(self, name, actions):
        self.macro_recorder.record_macro(name, actions)

    def schedule_task(self, task_name, time):
        self.task_scheduler.schedule_task(task_name, time)

    def execute_all(self):
        print("\n--- Executing All Automated Actions ---")
        self.rule_manager.execute_rules()
        self.task_scheduler.execute_tasks()

if __name__ == "__main__":
    app = MainFacade()

    app.add_rule("New episode released", "Download the episode")

    app.record_macro("LoginWorkflow", ["Open browser", "Navigate to login page", "Enter credentials"])

    app.schedule_task("Share torrents", "05:00 AM")

    app.execute_all()
```

Рис. 3 — Перевірка роботи

У методі `main` створюється об'єкт типу **MainFacade**, який об'єднує три різні компоненти автоматизації: **RuleManager**, **MacroRecorder** та **TaskScheduler**. Це демонструє реалізацію патерну **Facade**, де складна система управління автоматизацією спрощується через єдиний інтерфейс. За допомогою цього об'єкта виконуються дії: додавання правила (`add_rule`), запис макросу (`record_macro`) та планування завдання (`schedule_task`). Потім викликається метод **execute_all**, який ілюструє, як через фасад можна централізовано виконати всі автоматизовані дії, залишаючи внутрішню логіку компонентів прихованою.

```
Rule created: WHEN 'New episode released' THEN 'Download the episode'
Macro 'LoginWorkflow' recorded with actions: ['Open browser', 'Navigate to login page', 'Enter credentials']
Task 'Share torrents' scheduled for 05:00 AM

--- Executing All Automated Actions ---
Executing all rules...
Executing all scheduled tasks...
```

Рис. 4 — Результат роботи

Результати виконання програми демонструють, як різні компоненти автоматизації виконують свої завдання. Наприклад, у випадку управління правилами, додано правило "New episode released", яке автоматично ініціює завантаження нового епізоду. Для макросів, записано макрос "LoginWorkflow", який включає такі дії, як відкриття браузера, перехід на сторінку входу та введення облікових даних. У той же час, завдання "Share torrents" було успішно заплановано для виконання о 05:00 AM. Це підтверджує ефективність реалізації патерну **Facade**, оскільки кожен компонент виконує свою роль, не впливаючи на інші частини системи.

Висновки:

У цій лабораторній роботі ми реалізували патерн **Facade**, щоб спростити взаємодію з різними компонентами автоматизації через єдиний інтерфейс. Це забезпечило зручність у роботі з різними типами автоматичних дій (управління правилами, макросами, завданнями), дозволяючи зберігати загальну логіку програми. Такий підхід робить додаток гнучким і масштабованим, що дозволяє легко додавати нові функції або змінювати існуючі в майбутньому.

Код: <https://github.com/Lepseich/trpz/tree/main/lab7/files>