



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 №6
З дисципліни «Технології розроблення програмного забезпечення»
Тема: «**ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento»,
«Observer», «Decorator»»**
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 «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Мета:

Ознайомитися з основними шаблонами проєктування, такими як «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator», дослідити їхні принципи роботи та навчитися використовувати їх для створення гнучкого та масштабованого програмного забезпечення.

Завдання:

Розробка інструмента для автоматизації простих дій користувача в вигляді візуального додатку, який дозволяє організовувати та автоматизувати рутинні процеси. Додаток включає можливість створення "карт пам'яті" з функціоналом для роботи з кількома картами в різних вкладках. Передбачено автоматичне з'єднання елементів карт за допомогою ліній, а також можливість додавання різноманітних медіафайлів (вкладених файлів, зображень, відео з попереднім переглядом). Користувач може також додавати значки для категорій та терміновості завдань, а також окреслювати важливі області карти за допомогою пунктирних ліній.

Автоматизація процесів дозволяє інтегрувати правила для регулярних дій (наприклад, автоматичне оновлення карт чи додавання нових елементів за певним розкладом) через планувальник завдань або за допомогою макросів, що забезпечують зручність і ефективність роботи з додатком.

Хід роботи

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



Рис. 1 — Структура проєкту

У ході роботи було розроблено та модифіковано наступні класи:

1. AutomationFactory (Інтерфейс)

- Визначає абстрактні методи для створення компонентів автоматизації (завантаження файлів, встановлення статусу, планування завдань). Це основа для реалізації конкретних фабрик.

2. FileAutomationFactory (Конкретна фабрика 1)

- Реалізує AutomationFactory для створення компонентів автоматизації, орієнтованих на завантаження файлів (наприклад, фільмів чи книг).

3. **StatusAutomationFactory (Конкретна фабрика 2)**

- Реалізує AutomationFactory для створення компонентів автоматизації, орієнтованих на встановлення статусів в комунікаторах (наприклад, Skype).

4. **FileAutomation (Інтерфейс)**

- Абстрактне представлення дії завантаження файлів. Містить методи для визначення типу контенту, його завантаження та обробки.

5. **DownloadMovieAction (Конкретний клас 1)**

- Реалізує FileAutomation для завантаження нових серій фільмів або телевізійних шоу. Підтримує налаштування для автоматичного завантаження контенту за розкладом.

6. **StatusAutomation (Інтерфейс)**

- Абстрактне представлення дії для встановлення статусу в комунікаторах. Містить методи для налаштування і зміни статусу на основі активності.

7. **SetSkypeStatusAction (Конкретний клас 2)**

- Реалізує StatusAutomation для встановлення статусу в Skype або інших месенджерах залежно від активності користувача (наприклад, змінює статус на "Відсутній" після тривалого бездіяльності).

8. **TaskScheduler (Інтерфейс)**

- Абстрактне представлення планувальника завдань. Містить методи для створення, редагування і виконання запланованих завдань.

9. **TorrentSchedulerAction (Конкретний клас 3)**

- Реалізує TaskScheduler для автоматичного запуску та роздачі торрент-файлів у визначений час. Підтримує налаштування для запуску певних завдань о 5 ранку або в інший час за графіком.

2. Реалізувати один з розглянутих шаблонів за обраною темою

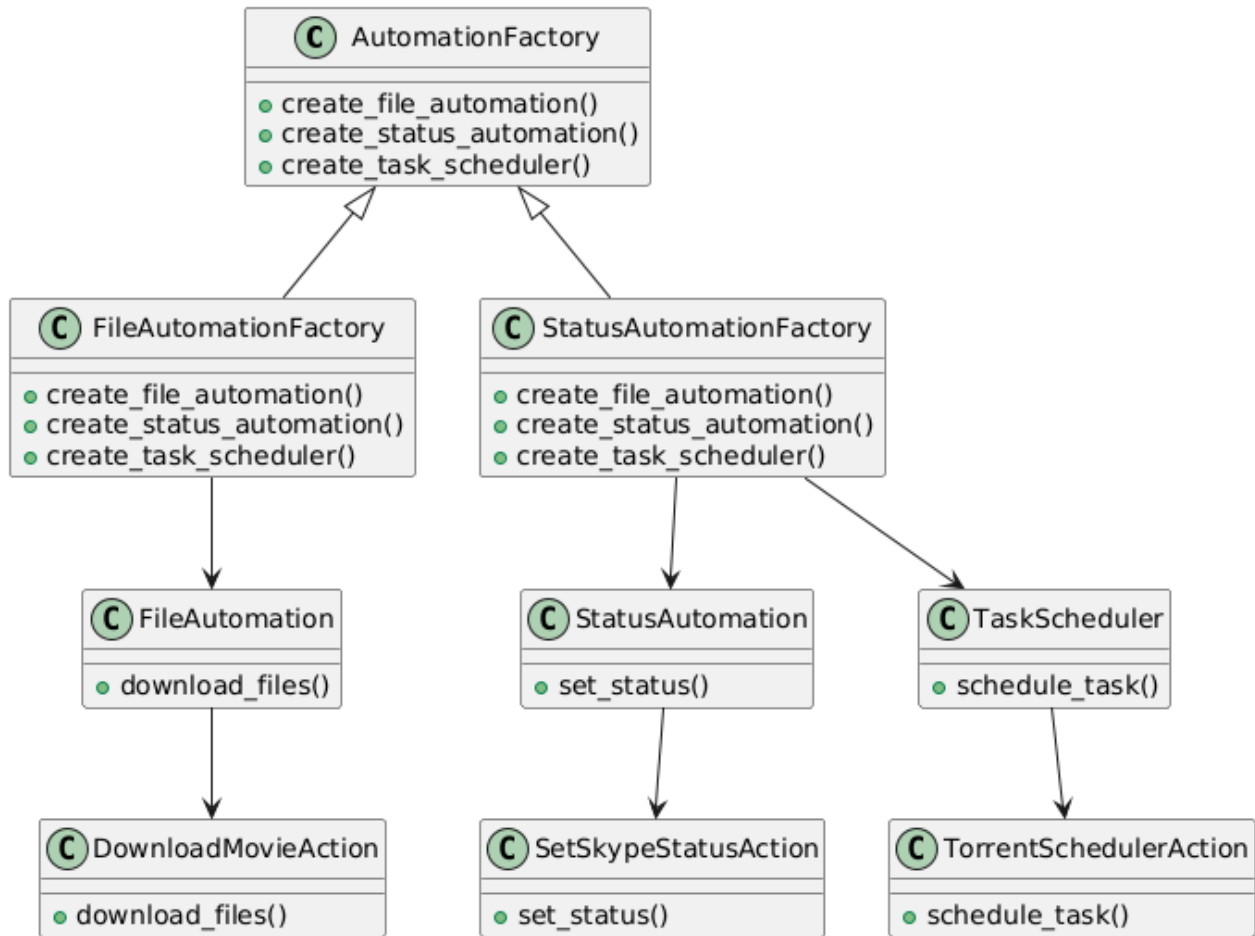


Рис. 2 — Діаграма класів

У проєкті інструмента автоматизації для виконання простих завдань (завантаження файлів, встановлення статусів, планування завдань) реалізація патерну Abstract Factory забезпечує модульність і гнучкість у створенні компонентів для різних типів автоматизації, таких як завантаження файлів, управління статусами і планування завдань.

Проблеми, які вирішує патерн Abstract Factory:

1. Залежність від конкретних реалізацій

Без використання Abstract Factory клієнтський код (додаток) був би жорстко прив'язаний до конкретних класів для кожного виду автоматизації (наприклад, `DownloadMovieAction` або `SetSkypeStatusAction`). Це ускладнює підтримку та розширення програми.

2. Дублювання коду

Якщо кожен компонент автоматизації створювати вручну, це призведе до повторюваного коду для визначення, які саме класи потрібно створити для кожного завдання.

3. Складність модифікації

Якщо потрібно додати нові типи автоматизації (наприклад, нові дії для

управління іншими програмами або сервісами), без Abstract Factory довелося б переписувати значну частину клієнтського коду.

Переваги використання патерну Abstract Factory:

1. Гнучкість

Легко додавати нові типи автоматизацій (наприклад, автоматизацію для нових месенджерів або нових типів завдань). Для цього потрібно лише створити нову фабрику та відповідні реалізації продуктів, не змінюючи існуючий код.

2. Інкапсуляція створення об'єктів

Вся логіка створення компонентів автоматизації зібрана в конкретних фабриках. Клієнтський код не потребує знання, які саме класи використовуються для завантаження файлів, встановлення статусу чи планування завдань.

3. Модульність

Кожен тип автоматизації (завантаження файлів, управління статусами, планування завдань) реалізований незалежно, що робить код більш структурованим і зрозумілим.

4. Легкість у підтримці

Оскільки клієнтський код працює лише з інтерфейсами, зміни в реалізаціях продуктів або додавання нових не впливають на основну логіку програми.

5. Зменшення ризиків помилок

Завдяки використанню фабрик, зменшується кількість випадкових помилок, пов'язаних із невідповідним створенням об'єктів (наприклад, створенням неправильних типів автоматизації для конкретного завдання).

Таким чином, використання патерну Abstract Factory дозволяє забезпечити розширюваність, модульність і підтримуваність проєкту, що є важливим для довгострокового розвитку та підтримки інструмента автоматизації.

Перевірка патерну

```
from abc import ABC, abstractmethod

class AutomationFactory(ABC): 2 usages
    @abstractmethod 1 usage
    def create_automation(self):
        pass

class FileAutomationFactory(AutomationFactory): 1 usage
    def create_automation(self):
        return DownloadMovieAction()

class Automation(ABC): 1 usage
    @abstractmethod 1 usage (1 dynamic)
    def execute(self):
        pass

class DownloadMovieAction(Automation): 1 usage
    def execute(self): 1 usage (1 dynamic)
        return "Завантаження нових серій фільмів..."

def run_automation(factory: AutomationFactory): 1 usage
    automation = factory.create_automation()
    print(automation.execute())

factory = FileAutomationFactory()
run_automation(factory)
```

Рис. 3 — Перевірка роботи

Опис коду:

1. Абстрактний клас **AutomationFactory**:

- Це базовий клас для всіх фабрик автоматизації, який містить один абстрактний метод: `create_automation()`. Цей метод має бути реалізований у конкретних фабриках для створення об'єкта автоматизації.

2. Конкретна фабрика **FileAutomationFactory**:

- Цей клас є конкретною реалізацією фабрики, яка створює об'єкти автоматизації, зокрема для завантаження фільмів. У методі `create_automation()` ця фабрика повертає об'єкт класу `DownloadMovieAction`, який є конкретною реалізацією автоматизації.

3. Абстрактний клас **Automation**:

- Це базовий клас для всіх типів автоматизацій, які можуть виконувати різні дії. Клас містить абстрактний метод `execute()`, який повинен бути реалізований у конкретних класах, що визначають дію автоматизації.

4. Конкретна реалізація автоматизації **DownloadMovieAction**:

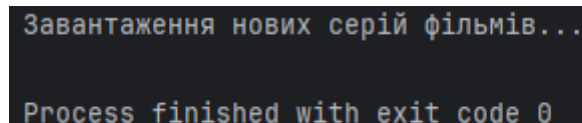
- Це клас, який реалізує конкретну дію автоматизації: завантаження нових серій фільмів. Метод `execute()` повертає рядок, що вказує на виконання цієї дії: "Завантаження нових серій фільмів...".

5. Клієнтський код **run_automation()**:

- У цьому методі ми приймаємо фабрику як параметр і використовуємо її для створення об'єкта автоматизації. Після цього викликається метод `execute()` на створеному об'єкті автоматизації, що призводить до виведення результату.

6. Тестування фабрики:

- У кінці коду створюється екземпляр фабрики `FileAutomationFactory` і передається в метод `run_automation()`, щоб перевірити, що буде виведено після виконання автоматизації.



```
Завантаження нових серій фільмів...  
Process finished with exit code 0
```

Рис. 4 — Результат роботи

Після виконання коду програма створює об'єкт автоматизації через фабрику `FileAutomationFactory`, яка в свою чергу створює об'єкт класу `DownloadMovieAction`. Коли викликається метод `execute()` цього об'єкта, виводиться рядок вище

Це підтверджує, що патерн `Abstract Factory` працює коректно: фабрика створює відповідний об'єкт автоматизації (у цьому випадку для завантаження фільмів), і його методи можуть бути викликані для виконання конкретної дії.

Висновки:

Використання патерна Abstract Factory дозволяє ефективно організувати створення компонентів автоматизації, роблячи додаток гнучким, масштабованим і легким у підтримці. Це особливо важливо для проектів, які підтримують кілька варіантів реалізації автоматизацій (наприклад, завантаження файлів, встановлення статусів, планування завдань) і можуть бути розширені в майбутньому.

Код: <https://github.com/Lepseich/trpz/tree/main/lab6/files>