



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

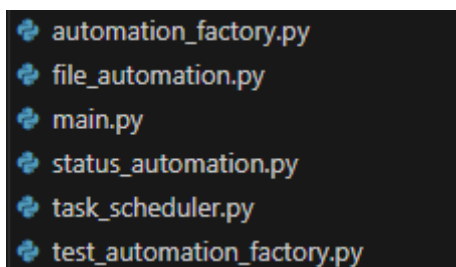


Рис. 1 — Структура проєкту

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

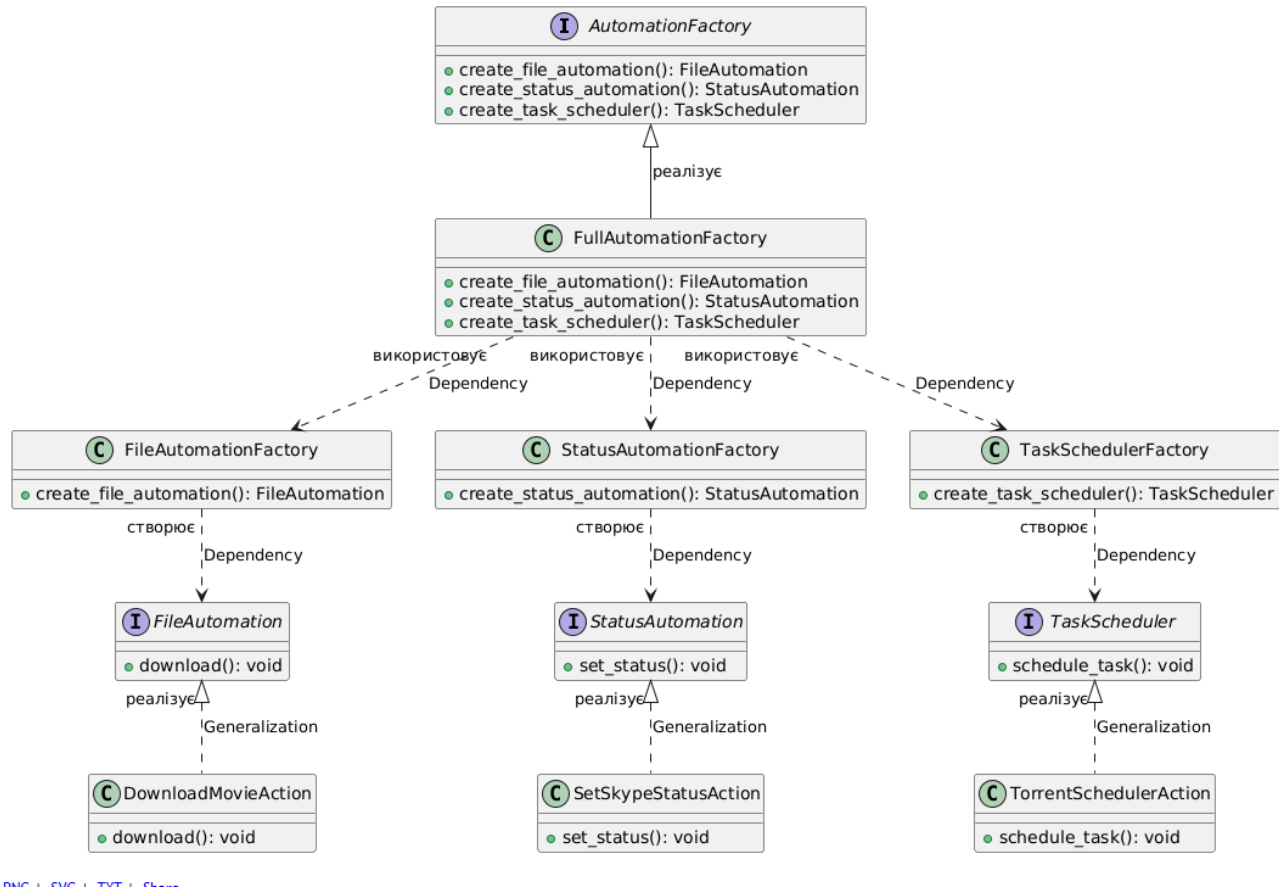


Рис. 2 — Діаграма класів

Діаграма відображає реалізацію патерну "Abstract Factory", який використовується для створення пов'язаних або залежних об'єктів без вказівки їх конкретних класів.

### Опис

Діаграма представляє реалізацію патерну **Abstract Factory**, який дозволяє створювати об'єкти без жорсткої прив'язки до їх конкретних класів.

- **AutomationFactory** — це абстрактна фабрика, яка визначає три основні методи:
  - `create_file_automation()` — створення об'єкта для роботи з файлами.
  - `create_status_automation()` — створення об'єкта для роботи зі статусами.
  - `create_task_scheduler()` — створення об'єкта для планування завдань.
- **FullAutomationFactory** — конкретна фабрика, що реалізує **AutomationFactory**. Вона делегує створення продуктів відповідним конкретним фабрикам:
  - **FileAutomationFactory** — створює екземпляри для автоматизації роботи з файлами (наприклад, **DownloadMovieAction**).
  - **StatusAutomationFactory** — створює об'єкти для роботи зі статусами (наприклад, **SetSkypeStatusAction**).

- **TaskSchedulerFactory** — створює планувальники завдань (наприклад, `TorrentSchedulerAction`).

Кожен продукт представлений абстрактним інтерфейсом (`FileAutomation`, `StatusAutomation`, `TaskScheduler`) та його конкретною реалізацією (`DownloadMovieAction`, `SetSkypeStatusAction`, `TorrentSchedulerAction`).

---

## Проблеми, які вирішує патерн Abstract Factory

### 1. Залежність від конкретних класів

Патерн дозволяє клієнтському коду (наприклад, `main.py`) працювати виключно через абстракції, уникаючи залежності від конкретних реалізацій класів. Це полегшує підтримку та розширення системи.

### 2. Сумісність об'єктів

`Abstract Factory` гарантує, що створювані об'єкти є частиною однієї родини (наприклад, всі компоненти автоматизації мають бути сумісними між собою).

### 3. Масштабованість

Додавання нових продуктів або фабрик не потребує змін у клієнтському коді. Наприклад, можна легко додати нову фабрику для іншого типу автоматизації.

### 4. Контрольоване створення об'єктів

Патерн забезпечує централізоване створення об'єктів через фабрики, що спрощує підтримку та тестування.

---

## Переваги використання Abstract Factory

### 1. Гнучкість

- Клієнтський код працює лише через інтерфейси, тому легко змінювати або розширювати систему.
- Можна підмінити одну фабрику іншою без змін у бізнес-логіці.

### 2. Інкапсуляція логіки створення

Логіка створення об'єктів зосереджена у фабриках, що робить код клієнта простішим і зрозумілішим.

### 3. Сумісність між продуктами

`Abstract Factory` гарантує, що створювані продукти сумісні між собою. Наприклад, якщо використовується `FileAutomationFactory`, створені об'єкти `DownloadMovieAction` будуть відповідати очікуванням системи.

### 4. Легке тестування та підтримка

Оскільки клієнт працює через інтерфейси, тестування стає простішим. Можна замінити реальні фабрики тестовими для перевірки системи.

### 5. Масштабованість

Додавання нових типів фабрик або продуктів не впливає на існуючий код клієнта, що робить систему відкритою для розширення.

## 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**:

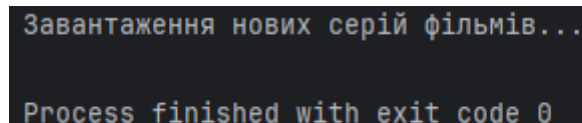
- Це клас, який реалізує конкретну дію автоматизації: завантаження нових серій фільмів. Метод `execute()` повертає рядок, що вказує на виконання цієї дії: "Завантаження нових серій фільмів...".

### 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:**

В цій лабораторній роботі був розроблений шаблон паттерну Abstract Factory, можна побачити діаграму класів, характеристики паттерну, та можна зробити висновок що використання паттерну Abstract Factory дозволяє забезпечити розширюваність, модульність і підтримуваність проєкту.

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