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

Технології розроблення програмного забезпечення

Шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Варіант №5

Виконав
студент групи ІА-13:
Засінець А. Є.

Київ 2023

Тема: Шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Хід роботи:

Тема проєкту:

..5 Аудіо редактор (singleton, adapter, observer, mediator, composite, client-server)

Аудіо редактор повинен володіти наступним функціоналом: представлення аудіо даних будь-якого формату в WAVE-формі, вибір і подальші операції копіювання / вставки / вирізання / деформації по сегменту аудіозапису, можливість роботи з декількома звуковими доріжками, кодування в найбільш поширених форматах (ogg, flac, mp3).

Завдання:

1. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

Для початку визначимо та опишемо кожен із шаблонів, що розглядаються та застосовуються під час виконання даної лабораторної роботи.

Abstract Factory

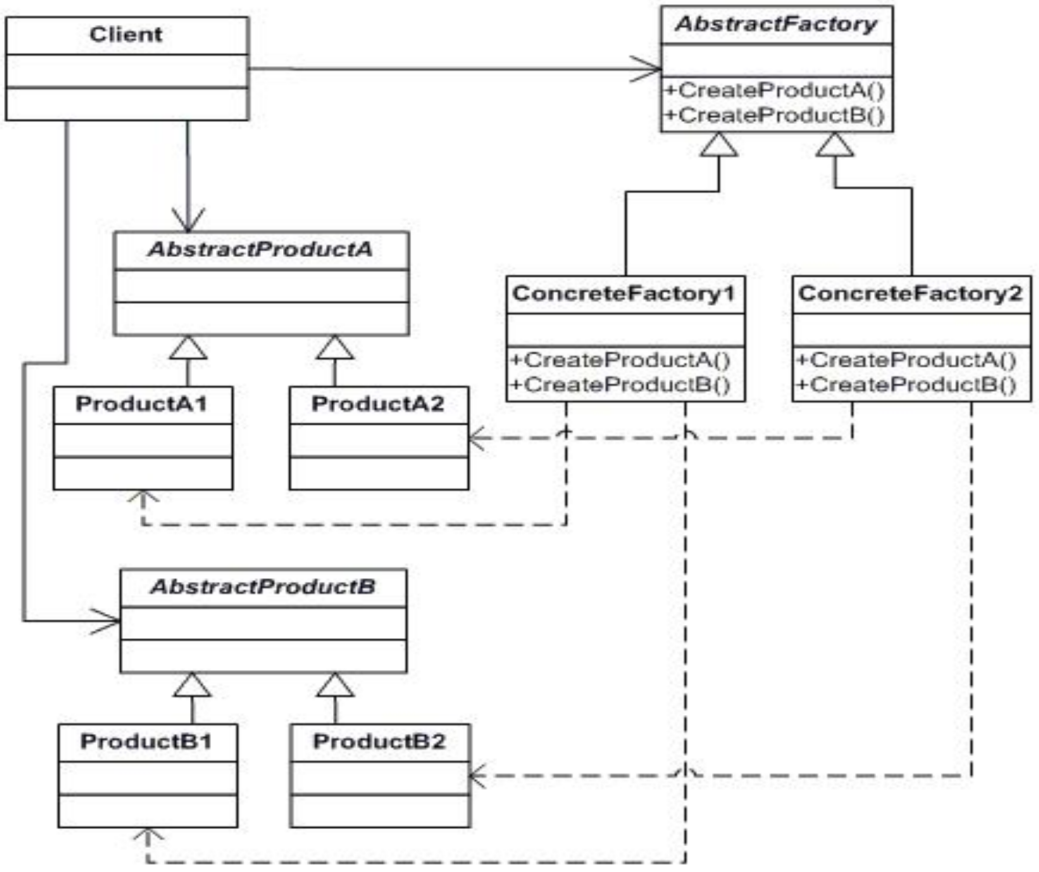

Рис. 1 - Структура Abstract Factory

Даний паттерн використовується для створення сімейств об'єктів без вказівки їх конкретних класів. Для цього виноситься загальний інтерфейс фабрики (AbstractFactory) і створюються його реалізації для різних сімейств продуктів.

Цей шаблон передусім структурує знання про схожі об'єкти (що називаються сімействами, як класи для доступу до БД) і створює можливість взаємозаміни різних сімейств (робота з Oracle ведеться також, як і робота з SQL Server). Проте, при використанні такої схеми украй незручно розширювати фабрику - для додавання нового методу у фабрику необхідно додати його в усіх фабриках і створити відповідні класи, що створюються цим методом.

Патерн Абстрактна фабрика пропонує виділити загальні інтерфейси для окремих продуктів, що складають одне сімейство, і описати в них спільну для цих продуктів поведінку

Всі варіації одного й того самого об’єкта мають жити в одній ієрархії класів.

Переваги та недоліки:

+ Гарантує поєднання створюваних продуктів

+ Звільняє клієнтський код від прив’язки до конкретних класів продукту.

+ Реалізує принцип відкритості/закритості.

- Ускладнює код програми внаслідок введення додаткових класів.

Factory Method

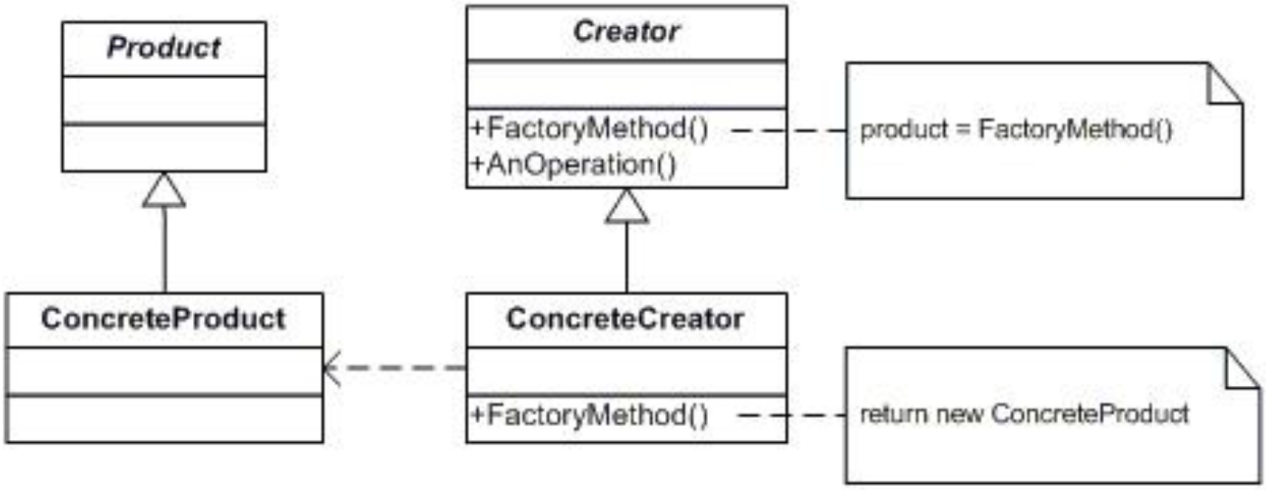

Рис. 2 - Структура паттерна Factory Method

Цей паттерн визначає інтерфейс для створення об'єктів певного базового типу. Це зручно, коли хочеться додати можливість створення об'єктів не базового типу, а деякого дочірнього. Фабричний метод у такому разі є зачіпкою для впровадження власного конструктора об'єктів. Основна ідея полягає саме в заміні об'єктів їх підтипами, що при цьому зберігає ту ж функціональність; інша частина поведінки об'єктів не є інтерфейсною (AnOperation) і дозволяє взаємодіяти із створеними об'єктами як з об'єктами базового типу. Тому шаблон "фабричний метод" носить ще назву "Віртуальний конструктор".

Factory Method пропонує відмовитись від безпосереднього створення об’єктів за допомогою оператора new, замінивши його викликом особливого фабричного методу.

Переваги та недоліки:

+ Позбавляє клас від прив’язки до конкретних класів продуктів.

+ Виділяє код виробництва продуктів в одне місце, спрощуючи підтримку коду.

+ Спрощує додавання нових продуктів до програми.

- Може призвести до створення великих паралельних ієрархій класів.

Memento

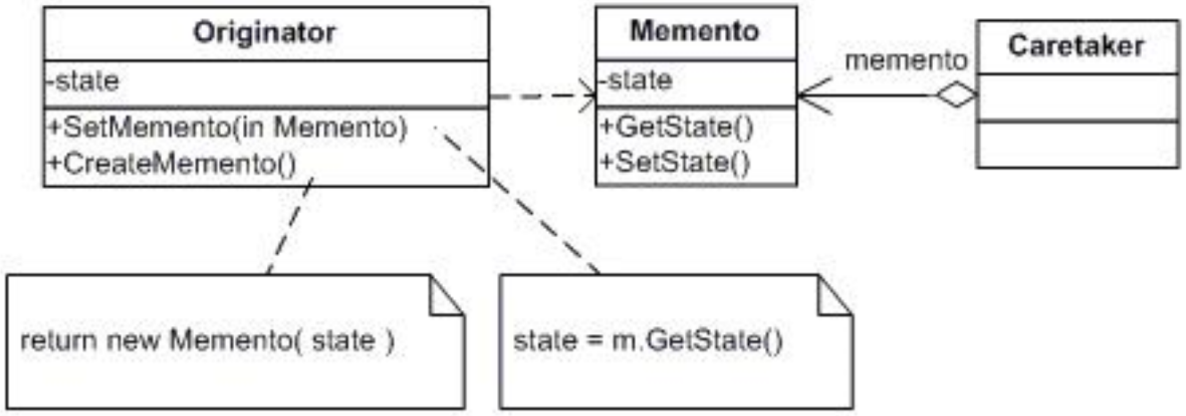

Рис. 3 - Структура паттерна Memento

Цей шаблон використовується для збереження і відновлення стану об'єктів без порушення інкапсуляції. Об'єкт "мементо" служить виключно для збереження змін над початковим об'єктом (Originator). Лише початковий об'єкт має можливість зберігати і отримувати стан об'єкту "мементо" для власних цілей, цей об'єкт є "порожнім" для кого-небудь ще. Об'єкт Caretaker використовується для передачі і зберігання мементо об'єктів в системі.

Шаблон "мементо" дуже зручно використати разом з шаблоном "команда" для реалізації "скасовних" дій - дані про дію зберігаються в мементо, а команда має можливість вважати і відновити початкове положення відповідних об'єктів.

Патерн Знімок доручає створення копії стану об’єкта самому об’єкту, який цим станом во- лодіє. Замість того, щоб робити знімок «ззовні», наш редактор сам зробить копію своїх полів, адже йому доступні всі поля, навіть приватні.

Знімок повністю відкритий для творця, але лише частково відкритий для опікунів.

Така схема дозволяє творцям робити знімки та віддавати їх на зберігання іншим об’єктам, що називаються *опікунами*.

Переваги та недоліки:

+ Не порушує інкапсуляцію вихідного об’єкта.

+ Спрощує структуру вихідного об’єкта. Не потрібно зберігати історію версій свого стану.

- Вимагає багато пам’яті, якщо клієнти дуже часто створюють знімки.

- Може спричинити додаткові витрати пам’яті, якщо об’єкти, що зберігають історію, не звільняють ресурси, зайняті застарілими знімками.

Observer

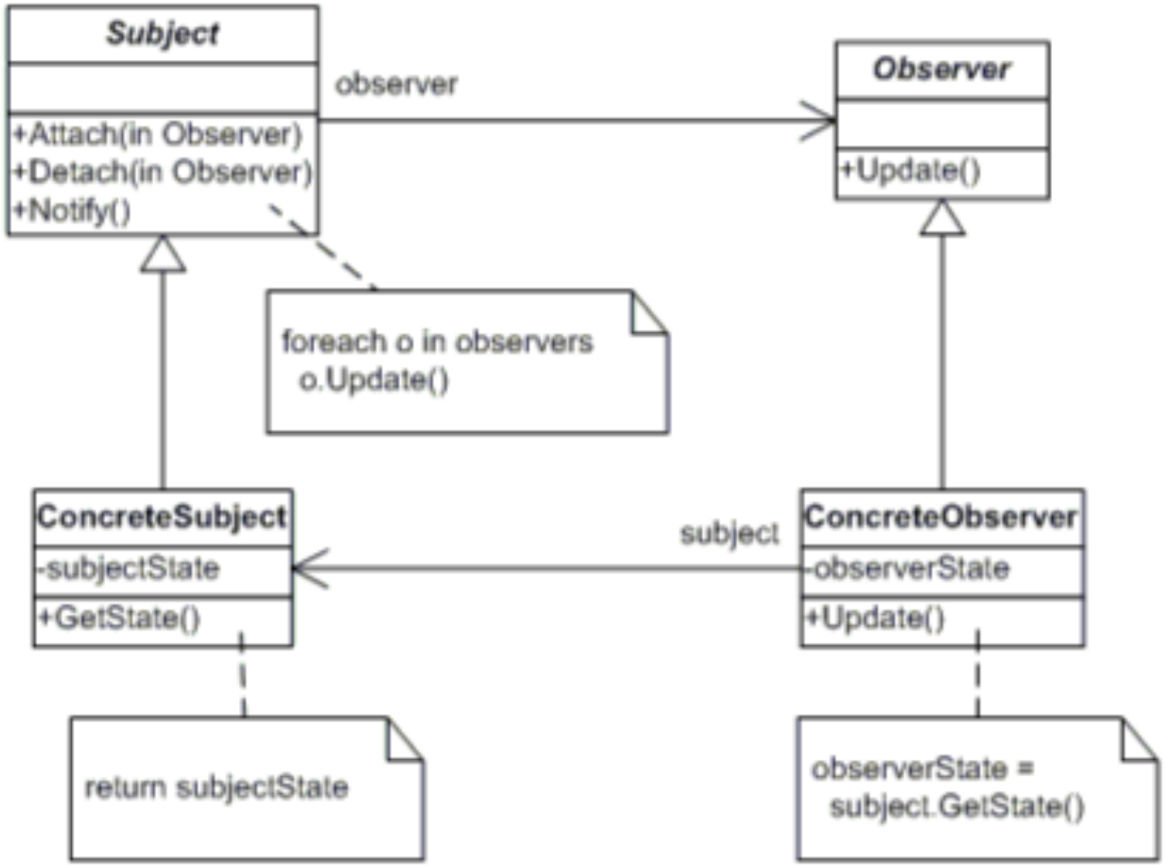

Рис. 4 - Структура паттерна Observer

Цей шаблон визначає залежність "один-до-багатьох" таким чином, що коли один об'єкт змінює власний стан, усі інші об'єкти отримують про це сповіщення і мають можливість змінити власний стан також.

Цей шаблон дуже широко поширений в шаблоні MVVM і механізмі "прив'язок" (bindings) в WPF і частково в WinForms. Інша назва шаблону - підписка/розсилка. Кожен з оглядачів власноручно підписується на зміни конкретного об'єкту, а об'єкти зобов'язані сповіщати своїх передплатників про усі свої зміни.

Патерн Спостерігач пропонує зберігати всередині об’єкта видавця список посилань на об’єкти підписників. Причому видавець не повинен вести список підписки самостійно. Він повинен надати методи, за допомогою яких підписники могли б додавати або прибирати себе зі списку.

Переваги та недоліки:

+ Ви можете підписувати і відписувати одержувачів «на льоту».

+ Видавці не залежать від конкретних класів підписників і навпаки.

+ Реалізує *принцип відкритості/закритості*.

- Підписники сповіщуються у випадковій послідовності.

Decorator

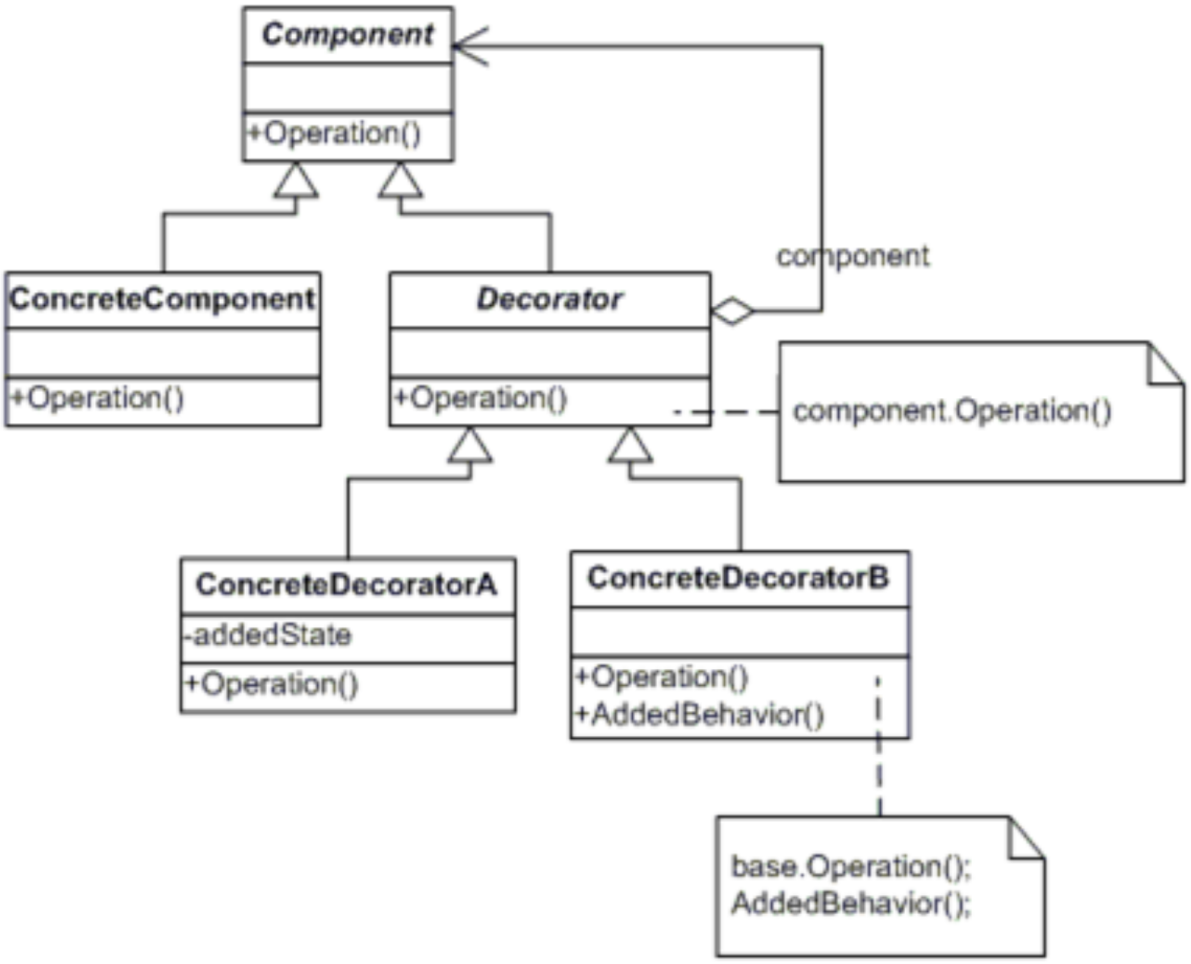

Рис. 5 - Структура паттерна Prototype

Шаблон призначений для динамічного додавання функціональних можливостей об'єкту під час роботи програми. Декоратор деяким чином "обертає" (агрегація) початковий об'єкт зі збереженням його функцій, проте дозволяє додати додаткові дії. Такий шаблон надає гнучкіший спосіб зміни поведінки об'єкту чим просте спадкоємство, оскільки початкова функціональність зберігається в повному об'ємі. Більше того, таку поведінку можна застосовувати до окремих об'єктів, а не до усієї системи в цілому.

Декоратор має альтернативну назву — обгортка. Вона більш вдало описує суть патерна: ви розміщуєте цільовий об’єкт у іншому об’єкті-обгортці, який запускає базову поведінку об’єкта, а потім додає до результату щось своє.

Переваги та недоліки:

+ Дозволяє мати кілька дрібних об’єктів, замість одного об’єкта «на всі випадки життя».

+ Дозволяє додавати обов’язки «на льоту».

+ Більша гнучкість, ніж у спадкування.

- Велика кількість крихітних класів.

- Важко конфігурувати об’єкти, які загорнуто в декілька обгорток одночасно

2. Застосування одного з розглянутих шаблонів при реалізації програми.

Як зазначено в завданні, необхідно реалізувати шаблон Observer.

Це такий паттерн, завдяки якому інші об’єкти сповіщаються про зміни станів у об’єкта.

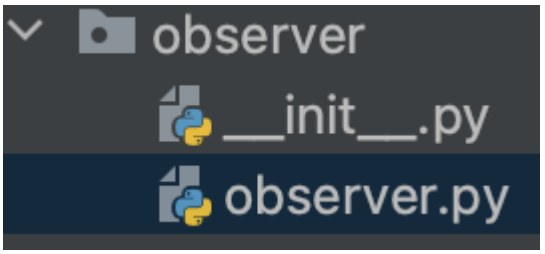

Рис. 6 - Структура програми

У файлі observer.py реалізовані методи підписки/відписки на деякі івенти та саме сповіщення про зміну станів.

```python
class AudioEditorObserver:
    def __init__(self):
        self.subscribers = []

    def subscribe(self, subscriber):
        self.subscribers.append(subscriber)

    def unsubscribe(self, subscriber):
        self.subscribers.remove(subscriber)

    def notify(self, event, *args):
        for subscriber in self.subscribers:
            self.update(event, subscriber, *args)

    def update(self, event, subscriber, *args):
        if event == "AudioOpened": message = "Audio data successfully loaded"
        elif event == "AudioPasted": message = "Audio pasted from clipboard"
        elif event == "TracksMixed":
            if args: message = f"Tracks successfully mixed and added as a new track: {args[0]}"
            else: message = "Tracks not mixed"
        elif event == "split_audio_track": message = "Track successfully split into two"
        elif event == "move_audio_track": message = "Track successfully moved"
        elif event == "SegmentCopied": message = "Audio segment copied to clipboard"
        elif event == "SegmentPasted": message = "Audio segment successfully pasted"
        elif event == "SegmentCut": message = "Audio segment successfully cut"
        elif event == "SegmentSelected": message = "Audio segment successfully selected"
        elif event == "AudioTrackAdded": message = "Audio track successfully added"
        elif event == "AudioDeformed": message = "Audio successfully deformed"
        elif event == "PitchShifted": message = "Pitch shift effect applied successfully"
        elif event == "AudioEncoded": message = "Audio successfully encoded"
        elif event == "AudioTrackSelected": message = "Audio track selected"
        elif event == "AudioTrackRemoved": message = "Audio track removed"
        elif event == "AudioCopied": message = "Audio copied to clipboard"
        elif event == "AudioCut": message = "Audio cut successfully"
        elif event == "list_audio_tracks":
            if args: message = f"List of audio tracks: {args[0]}"
            else: message = "List of audio tracks"
        elif event == "AudioSaved": message = 'Audio track saved'
        elif event == "DurationObtained":
            if args: message = f"Duration of audio track: {args[0]}"
            else: message = "Duration of audio track"
        else: return ''
        try:
            subscriber.sendall(message.encode())
        except Exception as e:
            print(f"Error sending message to client: {str(e)}")
```

Рис. 7 - Вміст файлу observer.py

Як можна побачити, у методі update() відбувається перевірка станів, які будуть реалізовані у наступних лабораторних роботах.

Висновки: Під час виконання даної лабораторної роботи ми вивчили інформацію про шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та реалізували шаблон Adapter у нашому застосунку.