Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут» імені Ігоря Сікорського
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота № 6

із дисципліни: «Технології розроблення програмного забезпечення»

на тему: ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Виконав:
студент групи ІА-34
Чунаков А.Р.
Перевірив:
Мягкий Михайло Юрійович

Київ-2025

Мета: Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

Короткі теоретичні відомості:

SOLID — це правило, що об’єднує п’ять фундаментальних принципів об’єктно-орієнтованого програмування та проєктування (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion), дотримання яких дозволяє створювати гнучкі, масштабовані та легкі для підтримки програмні системи. Суть цих принципів полягає у зменшенні залежностей між компонентами коду, полегшенні його рефакторингу та забезпеченні можливості розширення функціоналу без необхідності модифікації вже існуючого робочого коду, що є критично важливим для довгострокового успіху великих проєктів.

Шаблон «Абстрактна фабрика» — це патерн, який надає інтерфейс для створення сімейств взаємопов’язаних або залежних об’єктів, не специфікуючи їхніх конкретних класів. Він дозволяє клієнтському коду працювати з фабриками та продуктами через абстрактні інтерфейси, що забезпечує взаємозамінність цілих сімейств продуктів (наприклад, зміна стилю інтерфейсу з Windows на macOS) та гарантує, що всі створені фабрикою об'єкти будуть сумісні між собою.

Шаблон «Фабричний метод» — це патерн, який визначає загальний інтерфейс для створення об’єктів у суперкласі, але дозволяє підкласам змінювати тип створюваних об’єктів. Його головна ідея полягає у делегуванні логіки інстанціювання (створення екземплярів) спадкоємцям, що дозволяє системі залишатися незалежною від процесу створення конкретних продуктів і легко додавати нові типи об’єктів без зміни основного коду програми.

Шаблон «Знімок» — це патерн, який дозволяє зберігати та відновлювати минулий стан об’єкта, не розкриваючи подробиць його внутрішньої реалізації та не порушуючи інкапсуляцію. Він зазвичай використовується для реалізації функції «Скасувати» (Undo), де об’єкт (Originator) створює знімок свого стану, який зберігається в спеціальному об’єкті-зберігачі (Caretaker), і може бути використаний для повернення системи до попереднього стану в разі помилки або дії користувача.

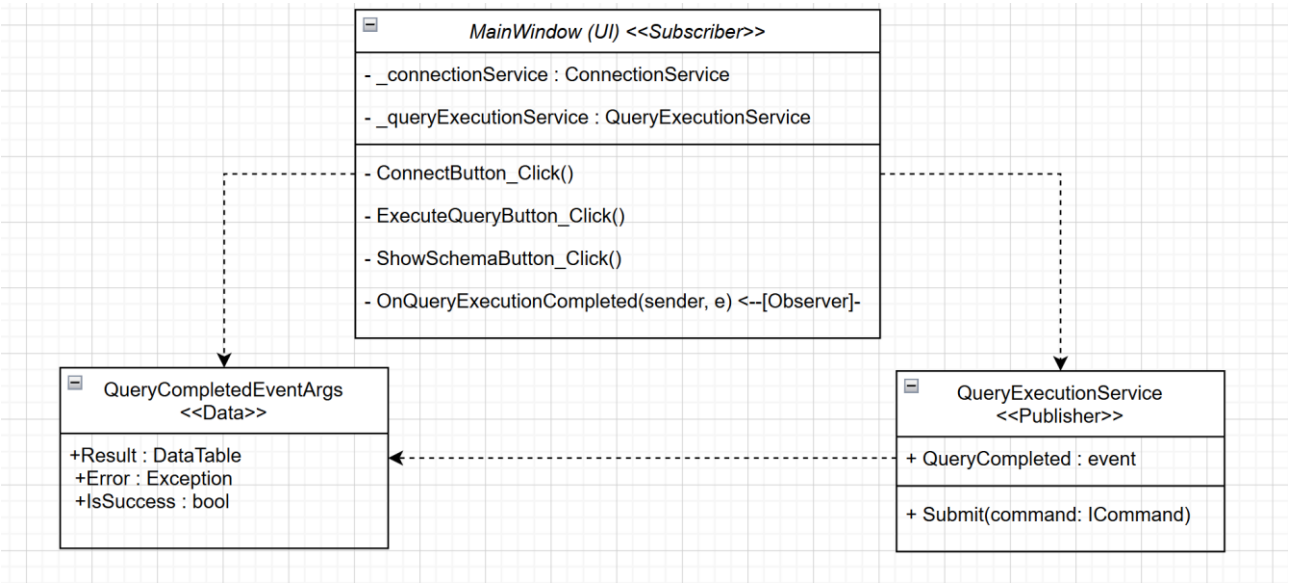
Шаблон «Спостерігач» — це патерн, який визначає механізм підписки «один до багатьох» для сповіщення кількох об’єктів про будь-які події, що стаються з об’єктом, за яким вони спостерігають. Цей шаблон дозволяє створити слабко зв’язану архітектуру, де суб’єкт (видавець) не знає конкретних класів своїх підписників, а лише сповіщає їх через спільний інтерфейс, що є ідеальним для реалізації систем обробки подій у користувацьких інтерфейсах.

Шаблон «Декоратор» — це патерн, який дозволяє динамічно додавати об’єктам нову функціональність, загортаючи їх у корисні «обгортки» (wrappers), що містять цю поведінку. На відміну від спадкування, яке розширює функціонал статично під час компіляції, декоратори надають гнучкішу альтернативу, дозволяючи комбінувати різні поведінки у довільному порядку під час виконання програми, не змінюючи код початкового класу.

Обрана тема лабораторної роботи: **SQL IDE**

9. **Sql IDE** (singleton, command, observer, bridge, visitor, SOA)

Інтегроване середовище розробки скриптів SQL повинна вміти працювати з більшістю сучасних баз даних (Microsoft SQL server, Oracle, IBM DB2, MySQL, PostgreSQL), виділяти синтаксичні одиниці конструкцій, виконувати і показувати результат їх виконання, отримувати схеми даних з джерела (список таблиць, стовпців, ключів і т.д.), порівнювати схеми даних з різних джерел.



Діаграма класів реазілації шаблону Observer

Хід роботи:

1. Реалізація частини функціоналу робочої програми у вигляді класів та їх взаємодій

Для досягнення функціональної можливості асинхронного сповіщення інтерфейсу користувача про завершення виконання SQL-запиту було розроблено та налаштовано взаємодію наступних класів:

- **QueryExecutionService (Publisher / Видавець):** Цей сервіс відповідає за бізнес-логіку виконання команд. Він не знає, хто саме ініціював запит, і не

має прямих посилань на елементи інтерфейсу (кнопки чи таблиці). Його задача — виконати роботу і повідомити про результат.

Посилання на код - https://github.com/1chuny/SQL-IDE/blob/main/Services/QueryExecutionService.cs

- **MainWindow (Subscriber / Підписник):** Це клас головного вікна (UI). Він зацікавлений у тому, щоб дізнатися, коли запит завершиться, щоб відобразити дані у DataGrid або показати помилку.

  Посилання на код - https://github.com/1chuny/SQL-IDE/blob/main/MainWindow.xaml.cs

- **QueryCompletedEventArgs (Data / Дані події):** Спеціальний клас-контейнер, який передається від Видавця до Підписника. Він містить результат операції (таблицю DataTable) або інформацію про помилку (Exception), а також статус успішності (IsSuccess).

  Посилання на код - https://github.com/1chuny/SQL-IDE/blob/main/Services/QueryCompletedEventArgs.cs

Опис взаємодії:

1. При ініціалізації програми (у конструкторі), MainWindow підписує свій метод OnQueryExecutionCompleted на подію сервісу.
2. Коли користувач натискає кнопку "Виконати", команда передається у QueryExecutionService.
3. Після завершення роботи з базою даних, QueryExecutionService генерує подію.
4. Завдяки механізму підписки, MainWindow автоматично отримує сповіщення, розпаковує дані з аргументів події та оновлює графічний інтерфейс.

2. Застосування шаблону Observer

У проєкті шаблон **Observer** було реалізовано за допомогою стандартного механізму подій (Events) платформи .NET.

- **Роль суб'єкта (Subject):** Клас QueryExecutionService. У ньому оголошено публічну подію public event EventHandler<QueryCompletedEventArgs> QueryCompleted. Це точка, до якої можуть підключатися спостерігачі. Коли виконання команди завершується, метод OnQueryCompleted ініціює цю подію, розсилаючи сповіщення всім підписникам.
- **Роль спостерігача (Observer):** Клас MainWindow. Він реалізує логіку реакції на подію. У нашому випадку це метод OnQueryExecutionCompleted,

який приймає результат виконання запиту і прив'язує його до ResultsDataGrid.ItemsSource.

Обґрунтування використання: Використання шаблону Observer дозволило досягти **слабкої зв'язності (loose coupling)** між логікою програми та інтерфейсом. Сервіс виконання запитів може працювати незалежно від того, існує головне вікно чи ні, і йому не потрібно знати про специфічні елементи управління (DataGrid, TextBox). Це значно спрощує підтримку коду та дозволяє легко додавати нових спостерігачів (наприклад, модуль логування), не змінюючи код самого сервісу.

Висновок

У ході виконання лабораторної роботи було програмно реалізовано поведінковий шаблон проєктування «Спостерігач» (Observer) з використанням механізму подій платформи .NET. Це дозволило налагодити асинхронну взаємодію між шаром бізнес-логіки (QueryExecutionService) та інтерфейсом користувача (MainWindow), забезпечивши автоматичне оновлення таблиці результатів та статус-бару після завершення виконання SQL-запитів. Впровадження цього патерну дозволило досягти слабкої зв'язності компонентів системи, оскільки сервіс виконання запитів тепер функціонує незалежно від наявності чи стану графічного інтерфейсу, що значно підвищує гнучкість архітектури та спрощує подальшу підтримку програмного продукту.

Контрольні питання

1. Яке призначення шаблону «Абстрактна фабрика»?

Призначення шаблону «Абстрактна фабрика» (Abstract Factory) полягає у наданні інтерфейсу для створення сімейств взаємопов'язаних або взаємозалежних об'єктів без зазначення їхніх конкретних класів. Він ізолює клієнтський код від конкретних класів продуктів, дозволяючи системі бути конфігурованою для роботи з різними сімействами продуктів (наприклад, створення елементів інтерфейсу для Windows або macOS), забезпечуючи при цьому сумісність створених об'єктів.

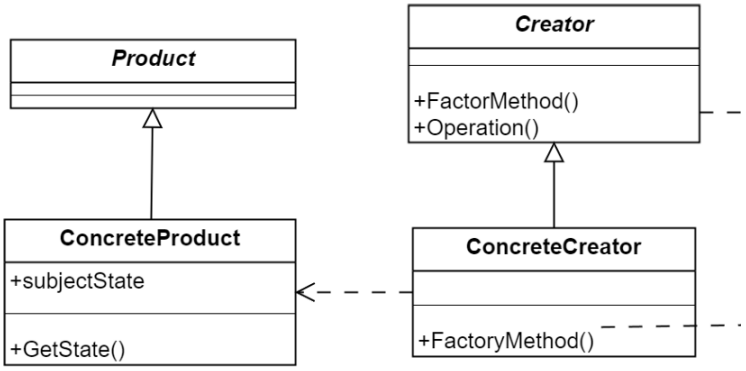
2. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія?

У шаблон «Абстрактна фабрика» входять: AbstractFactory (інтерфейс для створення продуктів), ConcreteFactory (реалізує інтерфейс для створення конкретного сімейства продуктів), AbstractProduct (інтерфейси для типів продуктів), ConcreteProduct (конкретні реалізації продуктів) і Client (використовує AbstractFactory та AbstractProduct для створення об'єктів). Взаємодія полягає в тому, що Client отримує ConcreteFactory, яка створює та повертає об'єкти ConcreteProduct, які відповідають AbstractProduct.

3. Яке призначення шаблону «Фабричний метод»?

Призначення шаблону «Фабричний метод» (Factory Method) полягає у визначенні інтерфейсу для створення об'єкта, але дозволі підкласам змінити або вирішити, який клас створювати. Він делегує відповідальність за інстанціювання (створення) об'єктів підкласам, тим самим усуваючи необхідність прив'язки клієнтського коду до конкретного класу продукту та забезпечуючи більшу гнучкість у розширенні системи новими типами продуктів.

4. Нарисуйте структуру шаблону «Фабричний метод».



5. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія?

У шаблон «Фабричний метод» входять: Creator (абстрактний клас або інтерфейс, який оголошує фабричний метод), ConcreteCreator (підкласи, які реалізують фабричний метод і повертають екземпляр конкретного продукту), Product (інтерфейс або абстрактний клас для об'єктів, які створюються) і ConcreteProduct (конкретна реалізація продукту). Взаємодія полягає в тому, що Client використовує метод Creator для отримання Product, а фактичний об'єкт ConcreteProduct створюється в підкласі ConcreteCreator.

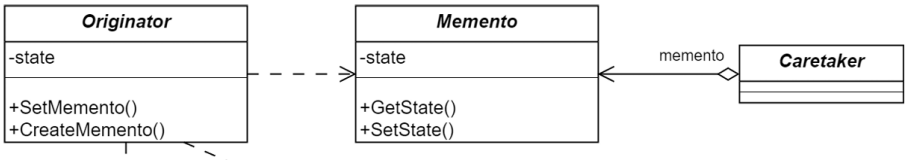
6. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

Головна відмінність полягає в масштабі та призначенні: «Фабричний метод» визначає один метод для створення одного типу об'єкта (продукту) і використовує спадкування (підкласи) для визначення того, який саме продукт створювати; тоді як «Абстрактна фабрика» визначає цілий інтерфейс (набір методів) для створення сімейства взаємопов'язаних продуктів, використовуючи композицію.

7. Яке призначення шаблону «Знімок»?

Призначення шаблону «Знімок» (Memento) полягає у захопленні та збереженні внутрішнього стану об'єкта (Originator) без порушення інкапсуляції, щоб об'єкт можна було пізніше відновити до цього стану. Він використовується для реалізації функціональності скасування (Undo) або для збереження стану об'єктів для подальшого відновлення.

8. Нарисуйте структуру шаблону «Знімок».



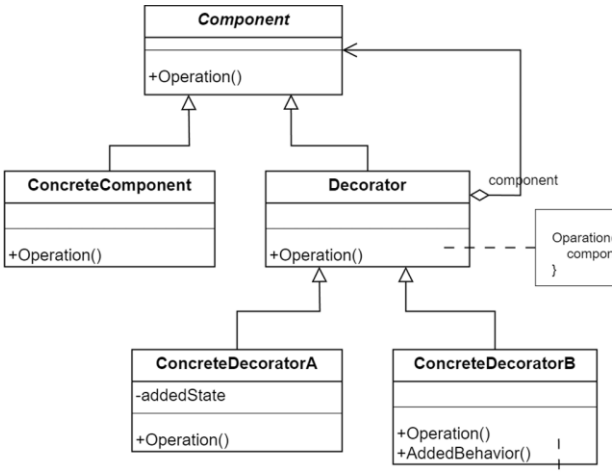
9. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

У шаблон «Знімок» входять: Originator (Творець) — об'єкт, чий стан потрібно зберігати, він створює та відновлює Memento; Memento (Знімок) — об'єкт, який зберігає стан Originator; і Caretaker (Опікун) — об'єкт, який зберігає Memento і ніколи не взаємодіє з його вмістом. Взаємодія полягає в тому, що Caretaker просить Originator створити Memento для збереження, а пізніше Caretaker передає Memento назад Originator для відновлення стану.

10. Яке призначення шаблону «Декоратор»?

Призначення шаблону «Декоратор» (Decorator) полягає у динамічному додаванні нової поведінки та обов'язків до об'єкта (Component) без зміни його класу. Він надає гнучку альтернативу спадкуванню для розширення функціональності, оскільки дозволяє «обернути» об'єкт декоратором, який додає нову функціональність, зберігаючи при цьому той самий інтерфейс.

11. Нарисуйте структуру шаблону «Декоратор».



12. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?

У шаблон «Декоратор» входять: Component (інтерфейс для об'єктів), ConcreteComponent (об'єкт, що декорується), Decorator (абстрактний клас, що реалізує Component і містить посилання на інший об'єкт Component) та ConcreteDecorator (конкретні декоратори, які додають нову поведінку перед або після делегування виклику обгорнутому об'єкту Component). Взаємодія полягає в тому, що декоратори обертають об'єкт, і кожен декоратор додає частину нової функціональності.

13. Які є обмеження використання шаблону «декоратор»?

Обмеження використання шаблону «Декоратор» полягає в тому, що він може ускладнити код, якщо його застосовувати надмірно, оскільки для додавання

простої функціональності потрібна велика кількість невеликих класів; також він може ускладнити ідентифікацію об'єктів, якщо клієнтський код покладається на перевірку типу, оскільки об'єкт і його декоратори мають спільний тип Component, але є різними об'єктами.