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 № 7

із дисципліни: «Технології розроблення програмного забезпечення»

на тему: ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

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

Київ-2025

Мета: Вивчити структуру шаблонів «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD» та навчитися застосовувати їх в реалізації програмної системи.

Короткі теоретичні відомості:

Принципи розробки:

Принцип DRY (Don’t Repeat Yourself) Принцип **DRY** («Не повторюй себе») стверджує, що кожна частина знання або логіки повинна мати єдине, однозначне представлення в системі. Його мета — уникнути дублювання коду, оскільки копіювання логіки в різні місця призводить до того, що при необхідності змін доводиться редагувати багато файлів, що підвищує ризик виникнення помилок і розсинхронізації поведінки програми.

Принцип KISS (Keep It Simple, Stupid) Принцип **KISS** (Залиш це простим) пропагує ідею, що системи працюють найкраще, якщо вони залишаються простими, а не ускладненими. У розробці це означає уникнення надмірної інженерії (over-engineering), використання зрозумілих рішень замість хитромудрих алгоритмів без нагальної потреби, оскільки простий код легше читати, тестувати та підтримувати.

Принцип You only load it once! (YOLO) вказує на необхідність підвантаження ініціалізаційних і конфігураційних змінних один раз при запуску програми, щоб уникнути проблем зі швидкістю (повторні зчитування даних з вінчестера).

Шаблони:

Шаблон «Mediator» (Посередник) — це патерн, який зменшує хаотичні зв'язки між об'єктами, змушуючи їх спілкуватися не напряму один з одним, а через спеціальний об'єкт-посередник. Це дозволяє усунути жорстку зв'язність (coupling) між компонентами системи, оскільки вони більше не залежать від конкретних класів своїх "сусідів", а вся логіка взаємодії централізується в одному місці, що спрощує її зміну та налагодження.

Шаблон «Facade» (Фасад) — це патерн, який надає простий, уніфікований інтерфейс до складної підсистеми, що містить безліч класів. Він приховує складність внутрішньої реалізації системи (наприклад, ініціалізацію бібліотек, налаштування залежностей, порядок викликів) за "фасадом", дозволяючи клієнтському коду взаємодіяти з системою через один зрозумілий об'єкт, не заглиблюючись у деталі.

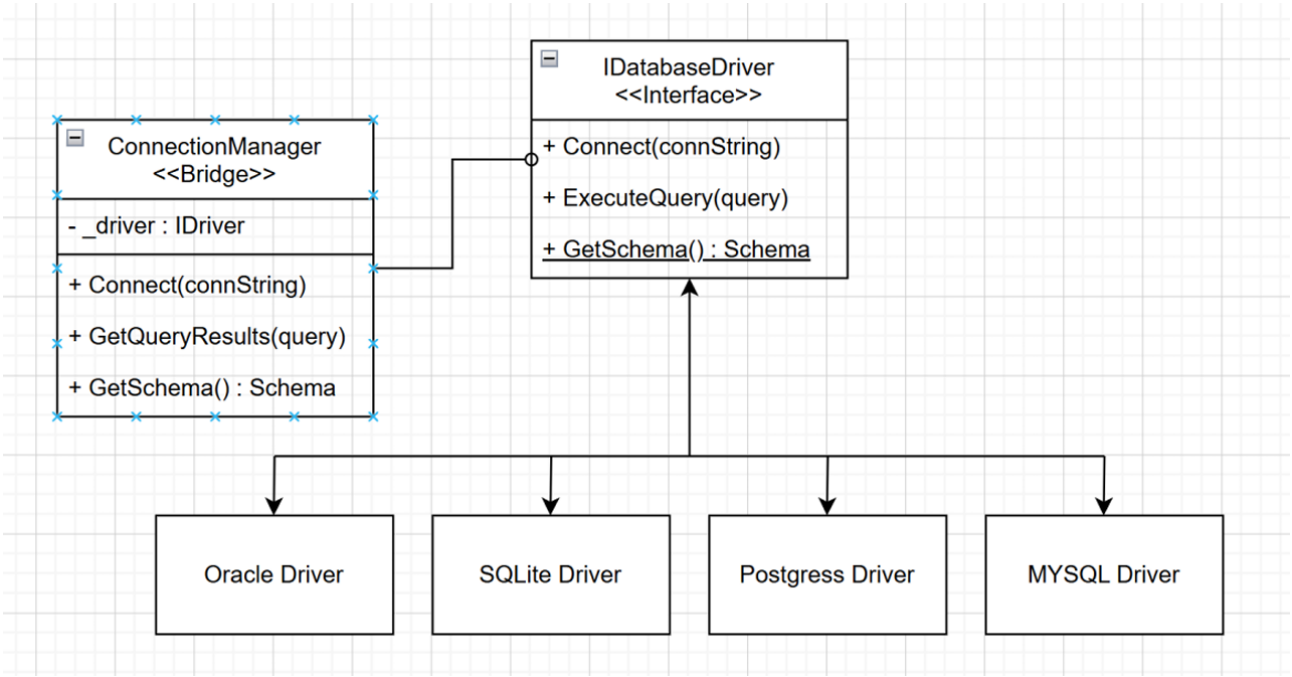
Шаблон «Bridge» (міст) використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами.

Шаблон «Template Method» (Шаблонний метод) — це патерн, який визначає "скелет" алгоритму в суперкласі, але дозволяє підкласам перевизначати певні кроки цього алгоритму без зміни його загальної структури. Це корисно, коли у вас є кілька класів, що виконують схожі дії з незначними відмінностями: спільна логіка залишається в батьківському класі (щоб уникнути дублювання за принципом DRY), а унікальні деталі реалізуються в нащадках.

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 Bridge

Хід роботи:

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

Для досягнення функціональної можливості універсальної роботи з різними системами управління базами даних (СУБД) через єдиний інтерфейс, було розроблено та налаштовано взаємодію наступних класів:

- **ConnectionManager (Абстракція):** Це високорівневий клас, який використовують інші частини програми (сервіси, команди). Він визначає, що саме потрібно зробити (наприклад, «Виконати запит» або «Отримати схему»), але сам не знає специфіки SQL-діалекту конкретної бази. Він містить посилання на інтерфейс драйвера.

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

- **IDatabaseDriver (Інтерфейс реалізації):** Це контракт, який описує низькорівневі операції, необхідні для роботи з будь-якою базою даних: Connect, Disconnect, ExecuteQuery, GetSchema.

  Посилання на код - https://github.com/1chuny/SQL-IDE/blob/main/DataAccess/Drivers/IDatabaseDriver.cs

- **SqliteDriver, PostgreSqlDriver, MySqlDriver... (Конкретні реалізації):** Це класи, які містять специфічний код для кожної СУБД. Вони використовують відповідні сторонні бібліотеки (Microsoft.Data.Sqlite, Npgsql тощо) для фізичного з'єднання з сервером та виконання команд.

  Посилання на код - https://github.com/1chuny/SQL-IDE/tree/main/DataAccess/Drivers

Опис взаємодії:

1. При створенні підключення, ConnectionService створює екземпляр конкретного драйвера (наприклад, PostgreSqlDriver) і передає його в конструктор ConnectionManager.
2. Коли користувач хоче виконати запит, UI викликає метод GetQueryResults у об'єкта ConnectionManager.
3. ConnectionManager не виконує запит самостійно, а делегує цю роботу збереженому драйверу, викликаючи його метод ExecuteQuery.
4. Драйвер виконує роботу і повертає результат, який через ConnectionManager потрапляє назад до програми.

2. Застосування шаблону «Міст» (Bridge) при реалізації програми

У проєкті шаблон **Bridge** було використано для розділення абстракції роботи з базою даних від її технічної реалізації.

Роль Абстракції (Abstraction): Клас ConnectionManager. Він надає клієнтському коду зручні методи для роботи, приховуючи складність. Він володіє посиланням на об'єкт типу IDatabaseDriver (поле _driver) і перенаправляє йому виклики.

Роль Реалізації (Implementor): Інтерфейс IDatabaseDriver. Він задає спільний інтерфейс для всіх можливих варіантів баз даних.

Роль Конкретної Реалізації (Concrete Implementor): Класи SqliteDriver, PostgreSqlDriver та інші. Кожен з них реалізує методи інтерфейсу IDatabaseDriver у свій унікальний спосіб, враховуючи особливості конкретної СУБД (різні рядки підключення, різні системні таблиці для отримання схеми).

Обґрунтування використання: Використання шаблону Bridge дозволило уникнути жорсткої прив'язки бізнес-логіки програми до конкретної бази даних. Завдяки цьому ми змогли реалізувати підтримку чотирьох різних СУБД, не створюючи заплутану структуру з безліччю умовних операторів (if/else). Додавання нової СУБД (наприклад, MS SQL Server) вимагає лише створення одного нового класу-драйвера, без необхідності змінювати код ConnectionManager або MainWindow. Зміни в API сторонніх бібліотек драйверів локалізовані всередині конкретних класів реалізації і не впливають на решту системи.

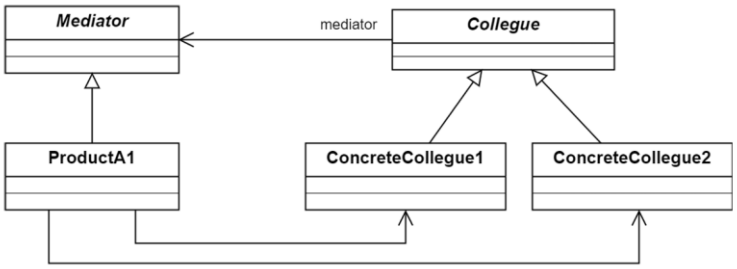
Висновок:

У ході виконання лабораторної роботи було програмно реалізовано структурний шаблон проєктування «Міст» (Bridge), що дозволило розділити високорівневу логіку керування підключеннями (ConnectionManager) від низькорівневої реалізації взаємодії з конкретними базами даних. Завдяки впровадженню інтерфейсу IDatabaseDriver та його конкретних реалізацій для SQLite, PostgreSQL, MySQL та Oracle, система отримала можливість працювати з різними джерелами даних через єдиний уніфікований інтерфейс. Таке архітектурне рішення забезпечило дотримання принципу відкритості/закритості (OCP), дозволивши додавати підтримку нових СУБД без модифікації існуючого коду бізнес-логіки та інтерфейсу користувача, що робить систему гнучкою та легко масштабованою.

Контрольні питання:

1. Яке призначення шаблону «Посередник» (Mediator)? Призначення шаблону «Посередник» полягає в зменшенні зв'язності (coupling) між класами, змушуючи їх взаємодіяти не напряму, а через спеціальний об'єкт-посередник. Це дозволяє централізувати складну логіку комунікації в одному місці, що спрощує модифікацію взаємодій та повторне використання окремих компонентів системи, які стають незалежними один від одного.

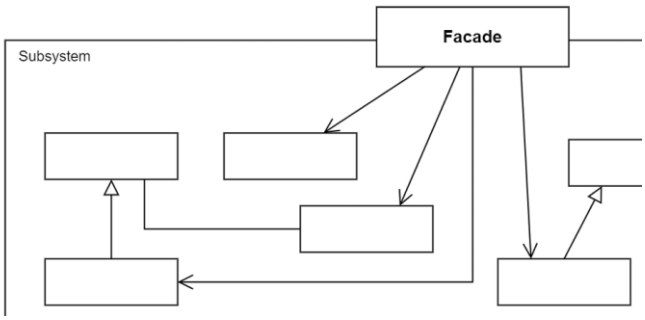
2. Нарисуйте структуру шаблону «Посередник».



3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія? До шаблону входять: Mediator (інтерфейс для взаємодії з колегами), ConcreteMediator (реалізує інтерфейс та координує дії), та Colleague (класи, що взаємодіють). Колеги (Colleagues) не знають один про одного; коли відбувається подія, вони повідомляють про це Посередника, який, у свою чергу, вирішує, якому іншому колезі передати управління або дані, реалізуючи таким чином всю логіку взаємодії.

4. Яке призначення шаблону «Фасад» (Facade)? Шаблон «Фасад» призначений для надання простого, уніфікованого інтерфейсу до складної підсистеми, що складається з багатьох класів. Він приховує складність внутрішньої реалізації, налаштування залежностей та порядок викликів, дозволяючи клієнтам взаємодіяти з системою через "одне вікно" без необхідності розбиратися в деталях її роботи.

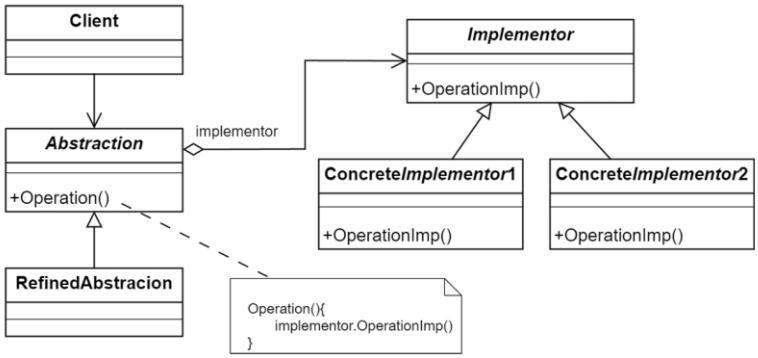
5. Нарисуйте структуру шаблону «Фасад».



6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія? До шаблону входять: Facade (надає спрощений інтерфейс клієнту) та Subsystem Classes (класи складної підсистеми). Клієнт звертається до методів Фасаду, який перенаправляє ці виклики відповідним об'єктам підсистеми в потрібному порядку. Класи підсистеми виконують фактичну роботу, але нічого не знають про існування Фасаду.

7. Яке призначення шаблону «Міст» (Bridge)? Призначення шаблону «Міст» — розділити абстракцію та її реалізацію на дві окремі ієрархії класів, щоб вони могли змінюватися незалежно одна від одної. Це дозволяє уникнути комбінаторного вибуху кількості класів при розширенні системи (наприклад, підтримка нових платформ або баз даних) і дає можливість динамічно змінювати реалізацію під час виконання програми.

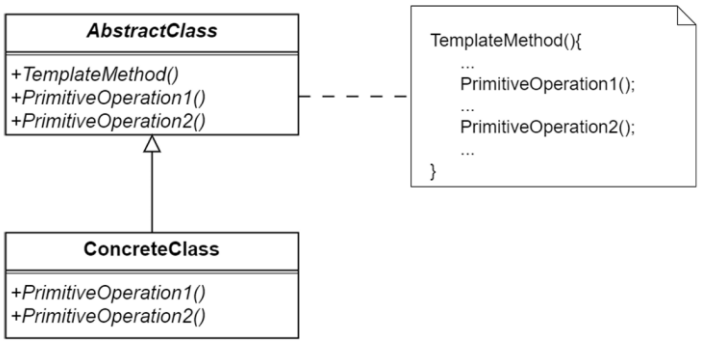
8. Нарисуйте структуру шаблону «Міст».



9. Які класи входять в шаблон «Міст», та яка між ними взаємодія? До шаблону входять: Abstraction (визначає інтерфейс високого рівня), RefinedAbstraction (розширює інтерфейс), Implementor (інтерфейс реалізації) та ConcreteImplementor (конкретна реалізація). Абстракція містить посилання на об'єкт Implementor і делегує йому виконання низькорівневих операцій, виступаючи як "міст" між клієнтським кодом і конкретною платформою чи технологією.

10. Яке призначення шаблону «Шаблонний метод» (Template Method)? Призначення цього шаблону — визначити скелет алгоритму в суперкласі, залишивши реалізацію окремих кроків підкласам. Це дозволяє перевизначати певні частини алгоритму без зміни його загальної структури, що сприяє усуненню дублювання коду в схожих класах.

11. Нарисуйте структуру шаблону «Шаблонний метод».



12. Які класи входять в шаблон «Шаблонний метод», та яка між ними взаємодія? До шаблону входять: AbstractClass (визначає шаблонний метод, що містить виклики кроків алгоритму) та ConcreteClass (реалізує абстрактні методи кроків). Абстрактний клас керує порядком виконання операцій, викликаючи методи, які реалізовані (або перевизначені) у конкретних класах-спадкоємцях.

13. Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»? «Шаблонний метод» — це *поведінковий* патерн, який керує алгоритмом виконання дії, дозволяючи змінювати його кроки через спадкування. «Фабричний метод» — це *породжувальний* патерн, який спеціалізується виключно на створенні об'єктів, делегуючи процес інстанціювання підкласам; по

суті, Фабричний метод часто є спеціалізацією Шаблонного методу, де "кроком алгоритму" є створення об'єкта.

14. Яку функціональність додає шаблон «Міст»? Шаблон «Міст» додає функціональність декулінгу (розв'язування) абстракції від реалізації, що дозволяє розробляти ці дві частини системи незалежно. Це забезпечує гнучкість у виборі реалізації (наприклад, драйвера бази даних) під час виконання програми та спрощує розширення системи новими типами абстракцій або реалізацій без необхідності переписувати існуючий код.