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 № 5

із дисципліни: «Технології розроблення програмного забезпечення»

на тему: «ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»»

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

Київ-2025

Мета: Вивчити структуру шаблонів «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE» та навчитися застосовувати їх в реалізації програмної системи.

Короткі теоретичні відомості:

«Анти-шаблони» проєктування «Анти-шаблони» — це типові, часто інтуїтивні рішення поширених проблем, які на перший погляд здаються ефективними, але в довгостроковій перспективі призводять до серйозних проблем, таких як надмірна складність коду, важкість підтримки та непередбачувана поведінка системи. Вони є своєрідними "шкідливими звичками" в програмуванні, вивчення яких так само важливе, як і вивчення правильних патернів, щоб вміти вчасно їх розпізнавати та виправляти (рефакторити) для забезпечення якості програмного продукту.

Шаблон «Adapter» (Адаптер) «Адаптер» — це структурний патерн проєктування, який дозволяє об'єктам з несумісними інтерфейсами працювати разом, виступаючи як проміжний шар, що трансформує виклики одного об'єкта у формат, зрозумілий іншому. Він часто використовується для інтеграції нових класів зі старим кодом або для використання сторонніх бібліотек, інтерфейс яких не відповідає вимогам вашої системи, без необхідності змінювати код цих класів.

Шаблон «Builder» (Будівельник) «Будівельник» — це породжувальний патерн, призначений для покрокового створення складних об'єктів, відділяючи процес їх конструювання від їхнього представлення, що дозволяє використовувати один і той самий процес будівництва для отримання різних варіацій об'єкта. Цей патерн особливо корисний, коли об'єкт має велику кількість можливих конфігурацій або параметрів конструктора, дозволяючи зробити код створення об'єкта більш читабельним і гнучким.

Шаблон «Command» (Команда) «Команда» — це поведінковий патерн, який інкапсулює запит на виконання дії у вигляді окремого об'єкта, що містить всю інформацію про операцію, її параметри та об'єкт-отримувач. Це дозволяє параметризувати клієнтів різними запитами, ставити операції в чергу, логувати їх, а також підтримувати операції скасування (undo), оскільки сама дія та її параметри зберігаються окремо від ініціатора запиту.

Шаблон «Chain of Responsibility» (Ланцюжок обов'язків) «Ланцюжок обов'язків» — це поведінковий патерн, який дозволяє передавати запит послідовно через ланцюжок потенційних обробників, поки один з них не візьме на себе відповідальність за його обробку. Це дозволяє уникнути жорсткої прив'язки відправника запиту до його отримувача і дає можливість динамічно змінювати ланцюжок або додавати нові обробники без зміни коду клієнта.

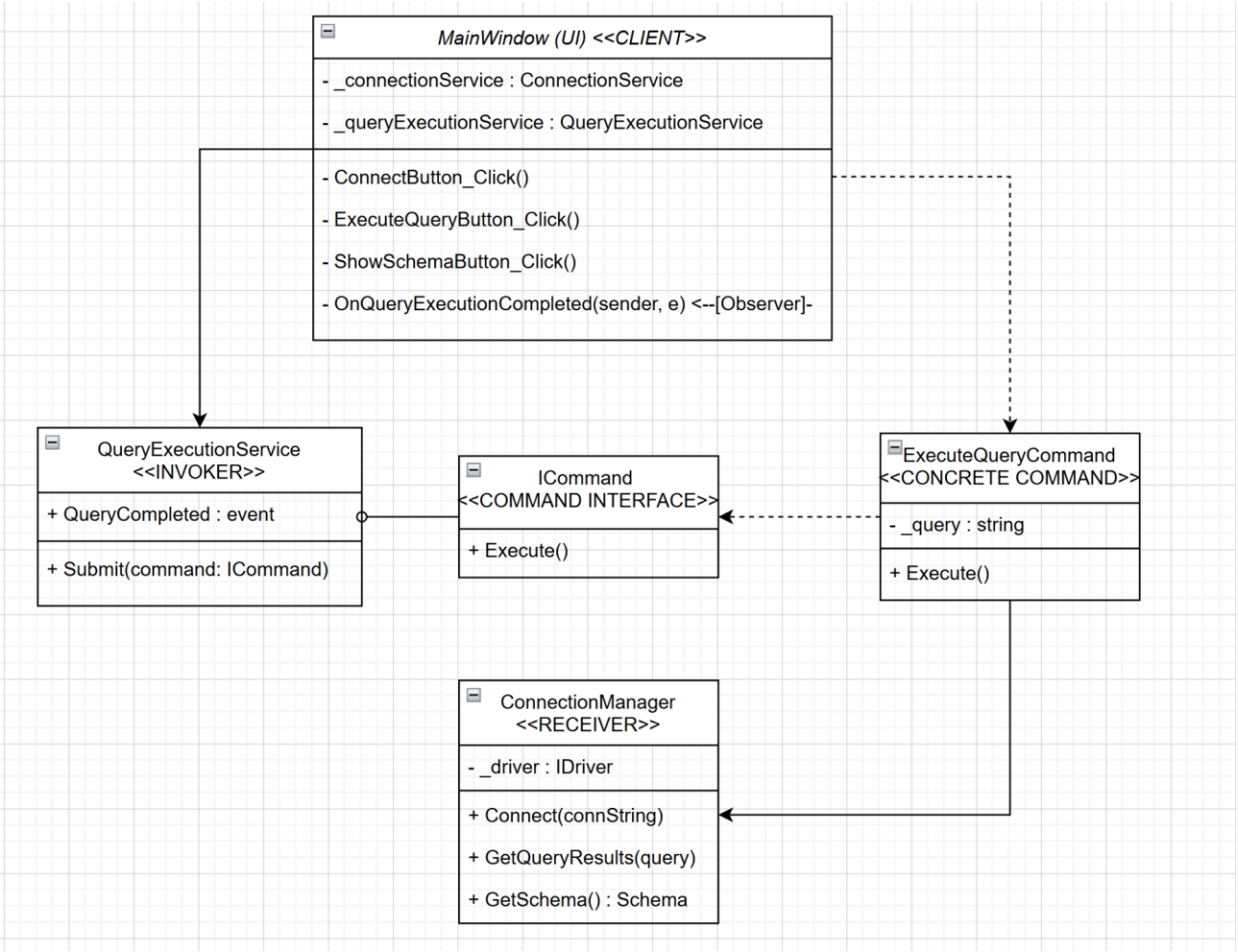
Шаблон «Prototype» (Прототип) «Прототип» — це породжувальний патерн, який дозволяє копіювати (клонувати) об'єкти, не вдаючись у подробиці їхньої

реалізації та не залежачи від їхніх класів. Замість створення нового екземпляра через конструктор і ручного налаштування всіх полів, клієнт звертається до існуючого об'єкта-прототипу з проханням створити свою копію, що особливо ефективно, коли створення об'єкта "з нуля" є ресурсомісткою операцією.

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 реалізації шаблону **Command**

Хід роботи:

Призначення шаблону в проєкті: Шаблон **Command** було використано для інкапсуляції запиту на виконання SQL-команди у вигляді окремого об'єкта. Це дозволило відділити ініціатора запиту (користувацький інтерфейс, кнопка "Виконати") від об'єкта, який безпосередньо виконує дію (драйвер бази даних). Такий підхід забезпечує слабку зв'язність системи та дозволяє централізовано керувати виконанням запитів.

Переваги використання у цьому проєкті:

- **Розділення відповідальності:** Клас вікна (MainWindow) не містить логіки роботи з базою даних, а лише створює об'єкти-команди.
- **Розширюваність:** У майбутньому легко додати нові типи команд (наприклад, ExportDataCommand або BackupDatabaseCommand), просто реалізувавши інтерфейс ICommand, не змінюючи код "Виконавця" (QueryExecutionService).
- **Керування виконанням:** Оскільки всі запити проходять через QueryExecutionService у вигляді об'єктів, ми легко змогли додати логування (історію запитів) та сповіщення про завершення (через подію QueryCompleted).

Класи, що реалізують шаблон «Команда» (Command)

Для реалізації цього шаблону було розроблено наступні три компоненти:

Client (Клієнт) — MainWindow:

- Відповідає за створення конкретної команди.
- У методі ExecuteQueryButton_Click він створює екземпляр ExecuteQueryCommand, передаючи йому необхідні дані (текст запиту) та посилання на отримувача (ConnectionManager).

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

Invoker (Виконавець) — QueryExecutionService:

- Клас, який ініціює виконання команди.
- Він не знає, що саме робить команда і хто її виконує. Він працює лише з інтерфейсом ICommand через метод Submit(), викликаючи метод Execute().

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

Command (Інтерфейс Команди) — ICommand:

- Оголошує спільний інтерфейс для всіх команд (метод Execute). Це дозволяє Invoker-у працювати з будь-якими типами команд однаково.

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

Concrete Command (Конкретна Команда) — ExecuteQueryCommand:

- Реалізує метод Execute().
- Зв'язує дію з конкретним отримувачем (Receiver).
- Зберігає стан запиту (параметр _query) та результат виконання (Result).
- При виклику Execute() делегує роботу методу GetQueryResults() отримувача.

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

Receiver (Отримувач) — ConnectionManager:

- Клас, який містить реальну бізнес-логіку. Він знає, як фізично виконати запит до бази даних. Команда лише передає йому управління.

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

Висновок

У ході виконання лабораторної роботи було успішно реалізовано поведінковий шаблон проєктування Command (Команда) для інкапсуляції операцій виконання SQL-запитів. Впровадження цього патерну дозволило досягти високого рівня абстракції та слабкої зв'язності між інтерфейсом користувача (ініціатором подій) та бізнес-логікою доступу до даних. Завдяки тому, що кожен запит тепер представлений як окремий об'єкт ExecuteQueryCommand, система отримала гнучку архітектуру, яка легко піддається розширенню новими типами операцій без зміни коду виконавця. Крім того, таке архітектурне рішення значно спростило реалізацію додаткового функціоналу, такого як логування дій користувача та збереження історії виконаних запитів, оскільки вся інформація про дію вже міститься всередині об'єкта команди.

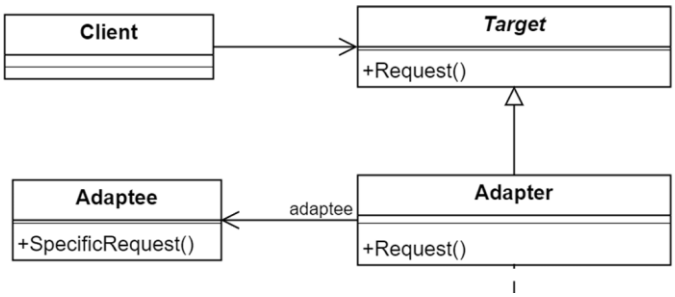
Контрольні питання

1. Яке призначення шаблону «Адаптер»?

Призначення шаблону «Адаптер» (Adapter) полягає у перетворенні інтерфейсу одного класу в інтерфейс, який очікує клієнтський клас, що дозволяє класам, які в іншому випадку не змогли б працювати разом через несумісність їхніх

інтерфейсів, співпрацювати, виступаючи як обгортка, яка приховує деталі перетворення та узгоджує методи існуючого класу (Adaptee) з інтерфейсом, який вимагає клієнт (Target).

2. Нарисуйте структуру шаблону «Адаптер».



3. Які класи входять в шаблон «Адаптер», та яка між ними взаємодія?

У шаблон «Адаптер» входять три основні класи: Target (Ціль) — інтерфейс, який очікує клієнт; Adaptee (Адаптований Клас) — існуючий клас із несумісним інтерфейсом; та Adapter (Адаптер) — клас, який реалізує інтерфейс Target і делегує запити об'єкту Adaptee (або успадковує його функціональність), перетворюючи виклики Target у виклики Adaptee.

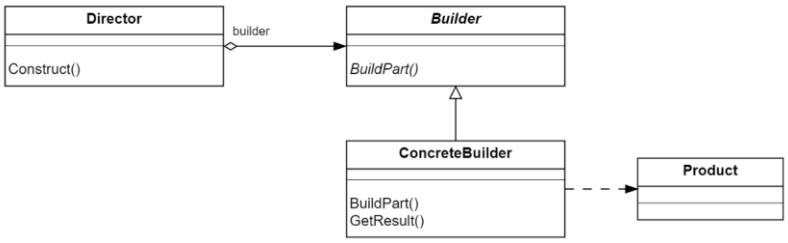
4. Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?

Різниця полягає у механізмі використання Adaptee: Об'єктний Адаптер реалізується через композицію (Adapter містить посилання на об'єкт Adaptee), що є гнучкішим і дозволяє працювати з підкласами Adaptee, тоді як Класовий Адаптер реалізується через множинне спадкування (Adapter успадковує як Target, так і Adaptee, що можливо не у всіх мовах програмування), дозволяючи йому безпосередньо перевизначати методи Adaptee.

5. Яке призначення шаблону «Будівельник»?

Призначення шаблону «Будівельник» (Builder) полягає у відокремленні конструювання складного об'єкта від його представлення, що дозволяє використовувати один і той же процес конструювання (визначений Director) для створення різних представлень (варіантів) об'єкта (Product), що особливо корисно, коли об'єкт має багато параметрів, і його створення є багатоетапним.

6. Нарисуйте структуру шаблону «Будівельник».



7. Які класи входять в шаблон «Будівельник», та яка між ними взаємодія?

У шаблон «Будівельник» входять Product (складний об'єкт), Builder (інтерфейс, що оголошує кроки конструювання), ConcreteBuilder (реалізує кроки та надає результат) і Director (керує процесом, викликаючи послідовно методи ConcreteBuilder); взаємодія полягає в тому, що Director використовує ConcreteBuilder для послідовного складання Product, а клієнт отримує готовий Product від Builder.

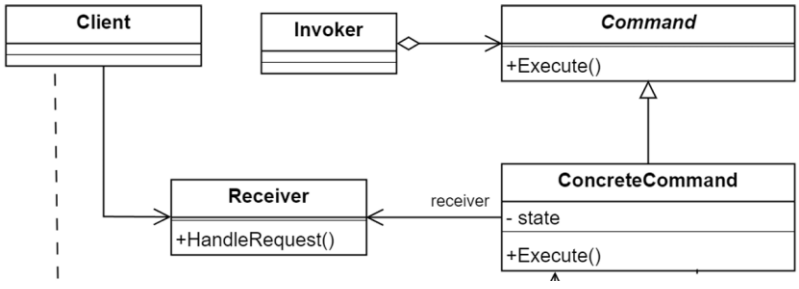
8. У яких випадках варто застосовувати шаблон «Будівельник»?

Шаблон «Будівельник» варто застосовувати, коли процес конструювання об'єкта є складним, багатоетапним і вимагає чіткого порядку, коли потрібно створювати різні варіанти одного і того ж складного об'єкта, використовуючи той самий код конструювання, і коли конструктор об'єкта має велику кількість необов'язкових параметрів, що робить конструктор непрактичним.

9. Яке призначення шаблону «Команда»?

Призначення шаблону «Команда» (Command) полягає у ін капсуляції запиту як об'єкта, що дозволяє параметризувати клієнтів різними запитами, ставити запити в чергу, логувати їх, а також підтримувати скасування (Undo) операцій, оскільки він відокремлює об'єкт, який викликає операцію (Invoker), від об'єкта, який її виконує (Receiver), за допомогою об'єкта-команди.

10. Нарисуйте структуру шаблону «Команда».



11. Які класи входять в шаблон «Команда», та яка між ними взаємодія?

У шаблон «Команда» входять Command (інтерфейс з методом execute()), ConcreteCommand (реалізує Command, зберігає посилання на Receiver та параметри), Invoker (викликає execute() на об'єкті Command) і Receiver (виконує фактичну роботу); взаємодія полягає в тому, що Invoker викликає команду, а команда, своєю чергою, делегує роботу Receiver, інкапсулюючи весь запит.

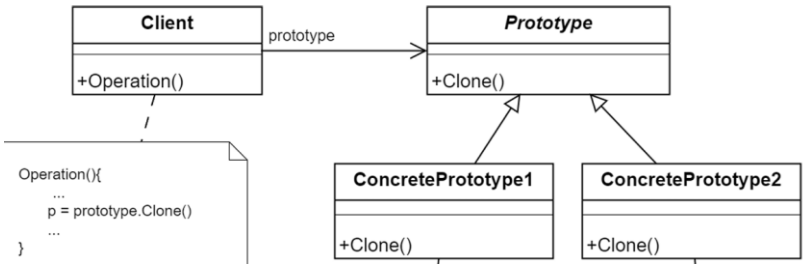
12. Розкажіть як працює шаблон «Команда».

Шаблон «Команда» працює так: клієнт створює об'єкт ConcreteCommand, передаючи йому об'єкт Receiver (який фактично виконує роботу); цей об'єкт команди передається Invoker (наприклад, елементу інтерфейсу); коли потрібно виконати дію, Invoker просто викликає уніфікований метод execute() на об'єкті команди, який, своєю чергою, викликає відповідний метод на об'єкті Receiver, не знаючи про конкретні дії Receiver.

13. Яке призначення шаблону «Прототип»?

Призначення шаблону «Прототип» (Prototype) полягає у створенні нових об'єктів шляхом копіювання (клонування) існуючого об'єкта (прототипу), замість використання традиційного конструктора, що є ефективним, коли створення об'єкта є дорогим або складним, або коли вам потрібно створювати об'єкти динамічно, приховуючи від клієнта деталі їхньої конкретної реалізації.

14. Нарисуйте структуру шаблону «Прототип».



15. Які класи входять в шаблон «Прототип», та яка між ними взаємодія?

У шаблон «Прототип» входять Prototype (інтерфейс або абстрактний клас, який оголошує метод clone()) та ConcretePrototype (класи, які реалізують операцію клонування, створюючи копію самого себе); взаємодія полягає в тому, що клієнт отримує новий екземпляр об'єкта, викликаючи його метод clone(), тим самим створюючи об'єкти без прив'язки до їхнього конкретного класу.

16. Які можна привести приклади використання шаблону «Ланцюжок відповідальності»?

Шаблон «Ланцюжок відповідальності» (Chain of Responsibility) часто використовується в таких сценаріях, як системи обробки подій у графічних інтерфейсах (де подія передається вгору по ієрархії компонентів), механізми валідації (де дані послідовно перевіряються різними валідаторами), системи автентифікації/авторизації (де запит на доступ проходить через кілька перевірок) або конвеєри обробки фільтрів (Middleware) у веб-фреймворках, де кожен елемент ланцюжка має шанс обробити або перенаправити запит.