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 № 4

із дисципліни: «Технології розроблення програмного забезпечення»

на тему: «Вступ до паттернів проектування»

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

Київ-2025

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

Короткі теоретичні відомості:

1. Що таке шаблони проєктування і навіщо вони потрібні?

Шаблони проєктування (Design Patterns) — це типові, перевірені часом архітектурні рішення поширених проблем, що виникають під час розробки програмного забезпечення. Це концептуальні схеми (описи взаємодії об'єктів та класів), які розробник адаптує під конкретну задачу.

Навіщо вони потрібні:

1. Уніфікація термінології: Шаблони створюють спільну мову для розробників. Сказавши «ми використаємо тут Singleton», програміст миттєво передає колезі складну концепцію без необхідності пояснювати деталі реалізації.
2. Використання кращих практик: Шаблони є квінтесенцією досвіду тисяч розробників. Використовуючи їх, ви уникаєте «винахідництва велосипеда» та типових архітектурних помилок.
3. Гнучкість та розширюваність: Правильно застосований шаблон робить систему більш стійкою до змін. Код стає легше підтримувати, оскільки логіка розділена на незалежні модулі.

2. Сутність шаблонів

Singleton (Одинак)

Гарантує, що клас має лише один екземпляр, і надає глобальну точку доступу до нього. Клас сам контролює створення свого єдиного об'єкта. Конструктор робиться приватним, щоб заборонити створення екземплярів ззовні, а доступ до об'єкта здійснюється через статичний метод або властивість. Застосування: Логери, драйвери баз даних, кеші, менеджери налаштувань.

Iterator (Ітератор)

Дає можливість послідовно обходити елементи складових об'єктів (колекцій), не розкриваючи їхнього внутрішнього представлення. Основна ідея полягає у винесенні логіки обходу колекції в окремий об'єкт — ітератор. Клієнтський код використовує інтерфейс ітератора (методи на кшталт Next, HasNext) і не залежить від того, як дані зберігаються (масив, список, дерево). Застосування: Обхід списків, дерев, результатів SQL-запитів.

Proxy (Замісник)

Надає об'єкт-замінник, який керує доступом до іншого об'єкта. Замісник має той самий інтерфейс, що й реальний об'єкт. Клієнт працює із замісником так само як

з оригіналом. Замісник перехоплює виклики клієнта, виконує якусь додаткову роботу (перевірка прав доступу, лінива ініціалізація, логування) і потім передає виклик реальному об'єкту. Застосування: Ліниве завантаження важких об'єктів, захист доступу, логування запитів.

State (Стан)

Дозволяє об'єкту змінювати свою поведінку залежно від свого внутрішнього стану. Ззовні створюється враження, що змінився клас об'єкта. Логіка, що залежить від стану, виноситься в окремі класи. Початковий об'єкт (Context) зберігає посилання на один з об'єктів-станів і делегує йому виконання роботи. При зміні стану контекст просто змінює посилання на інший об'єкт-стан. Застосування: TCP-з'єднання (встановлено, слухає, закрито), статус замовлення, поведінка персонажа в грі.

Strategy (Стратегія)

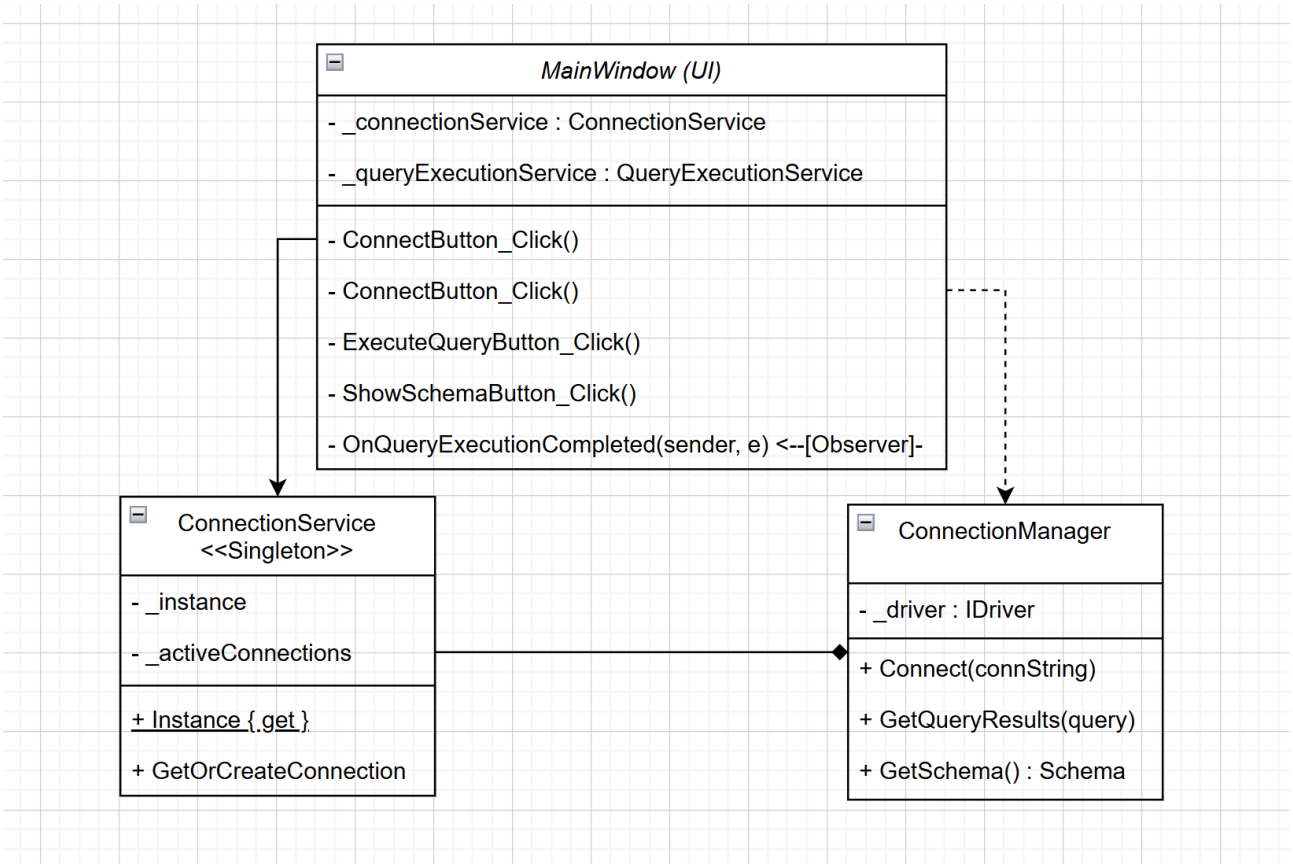
Визначає сімейство алгоритмів, інкапсулює кожен з них і робить їх взаємозамінними. Дозволяє змінювати алгоритми незалежно від клієнтів, які їх використовують. Клас-контекст не реалізує алгоритм самостійно, а зберігає посилання на інтерфейс стратегії. Конкретна реалізація алгоритму обирається під час виконання (наприклад, вибір способу сортування або маршруту на карті). Застосування: Різні способи сортування даних, різні стратегії навігації (пішки, авто, громадський транспорт), різні способи розрахунку податків.

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.

Хід роботи:



Діаграма класів шаблону Singleton

Реалізація функціональних класів системи:

Для досягнення функціональної можливості централізованого керування підключеннями до баз даних було розроблено та реалізовано взаємодію наступних класів:

- **MainWindow (UI):** Відповідає за взаємодію з користувачем. При натисканні кнопки "Підключитись" він звертається до сервісного шару. Посилання на код - https://github.com/1chuny/SQL-IDE/blob/main/MainWindow.xaml.cs
- **ConnectionService (Service):** Центральний клас, який відповідає за створення, зберігання та видачу активних менеджерів підключень. Посилання на код - https://github.com/1chuny/SQL-IDE/blob/main/Services/ConnectionService.cs
- **ConnectionManager (Domain Logic):** Клас, що безпосередньо керує драйвером бази даних. Посилання на код - https://github.com/1chuny/SQL-IDE/blob/main/DataAccess/ConnectionManager.cs

Взаємодія: MainWindow не створює підключення самостійно. Замість цього він звертається до ConnectionService, передаючи параметри (тип БД, рядок підключення). Сервіс перевіряє, чи існує вже таке підключення, і або повертає існуюче, або створює нове, повертаючи об'єкт ConnectionManager назад у UI для подальшої роботи.

Реалізація шаблону Singleton:

При реалізації класу **ConnectionService** було застосовано патерн **Singleton**.

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

У програмі необхідно уникнути ситуації, коли різні частини інтерфейсу (наприклад, головне вікно та вікно історії) створюють власні, незалежні екземпляри сервісу підключень. Це призвело б до втрати контексту, дублювання з'єднань та неможливості керувати станом програми централізовано.

1. Конструктор класу ConnectionService оголошено як private, що забороняє створення екземплярів через оператор new ззовні класу.
2. Створено статичне приватне поле _instance для зберігання єдиного екземпляра.
3. Реалізовано публічну статичну властивість Instance. Вона містить логіку перевірки: якщо об'єкт ще не створено — він ініціалізується; якщо вже існує — повертається посилання на нього.

Це гарантує, що всі компоненти системи (UI, команди, інші сервіси) завжди працюють з **одним і тим самим** об'єктом ConnectionService, маючи спільний доступ до списку активних підключень.

ВИСНОВОК

У ході виконання даної лабораторної роботи було реалізовано породжувальний шаблон проєктування **Singleton (Одинак)**. Цей патерн було застосовано до класу ConnectionService, який відповідає за керування підключеннями до баз даних.

Вибір саме цього шаблону зумовлений необхідністю мати **єдину точку керування станом** програми. У нашому SQL IDE користувач встановлює з'єднання з базою даних один раз, і це активне з'єднання має бути доступним для всіх частин програми (головного вікна, вікна історії, редактора запитів тощо).

Без використання Singleton нам довелося б створювати нові екземпляри сервісу в кожному вікні або передавати посилання на об'єкт через параметри, що призвело б до надмірної зв'язності коду та ризику розсинхронізації (коли одна частина програми "думає", що ми підключені, а інша — ні).

Реалізація ConnectionService через Singleton (з приватним конструктором та статичною властивістю Instance) гарантує, що в системі завжди існує лише один менеджер підключень. Це забезпечує цілісність даних, економію системних ресурсів та надає зручний глобальний доступ до активного з'єднання з будь-якої точки коду.

Контрольні питання:

1. Що таке шаблон проєктування?

Шаблон проєктування (Design Pattern) — це формалізований опис найкращого, перевіреного рішення типової, часто повторюваної проблеми, що виникає під час проєктування архітектури програмного забезпечення. Шаблони не є готовим кодом чи бібліотеками; це скоріше креслення або рецепти, які описують взаємодію класів і об'єктів для розв'язання конкретної проблеми проєктування в певному контексті, дозволяючи розробникам створювати гнучкі, багаторазово використовувані та зрозумілі системи.

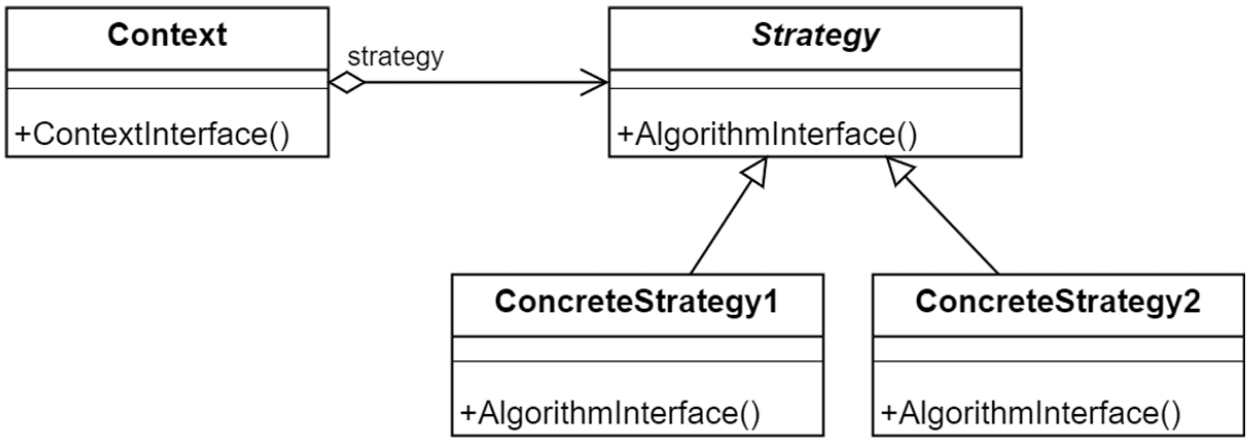
2. Навіщо використовувати шаблони проєктування?

Використання шаблонів проєктування дозволяє розробникам створювати системи, які є більш гнучкими, підтримуваними та розширюваними. Вони забезпечують спільний словник (термінологію) для опису архітектурних рішень, що полегшує спілкування між членами команди. Крім того, шаблони допомагають уникнути поширених проблем проєктування та повторно використовувати досвід, накопичений іншими розробниками, тим самим підвищуючи якість та надійність коду.

3. Яке призначення шаблону «Стратегія»?

Призначення шаблону «Стратегія» (Strategy) полягає у визначенні сімейства алгоритмів, інкапсуляції кожного з них і забезпеченні їхньої взаємозамінності. Цей шаблон дозволяє клієнтському об'єкту (Context) вибирати необхідний алгоритм (стратегію) під час виконання (runtime), а не фіксувати його жорстко, дозволяючи алгоритму змінюватись незалежно від клієнтів, які його використовують.

4. Нарисуйте структуру шаблону «Стратегія».



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

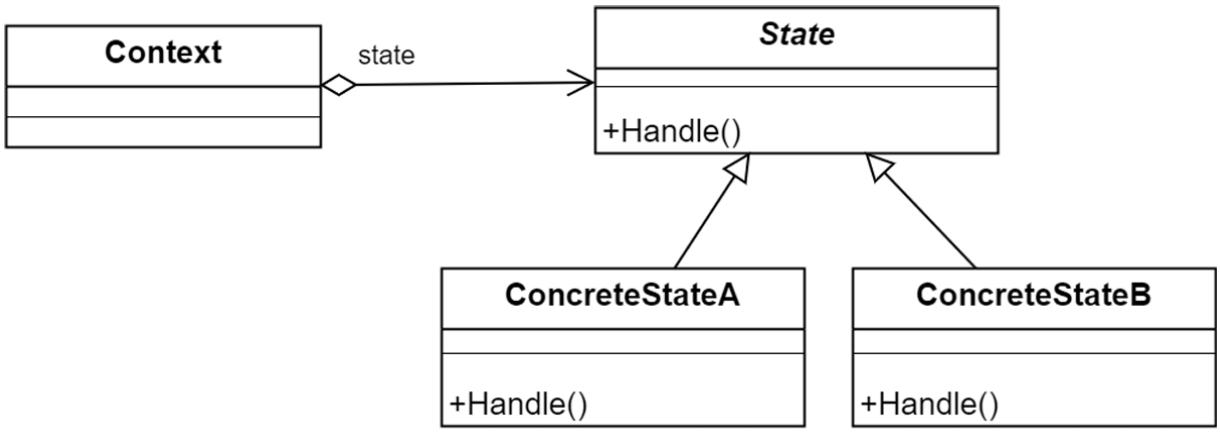
У шаблон «Стратегія» входять три основні класи:

1. Context (Контекст): Клас, який містить посилання на об'єкт Strategy. Він взаємодіє зі стратегією через її спільний інтерфейс.
2. Strategy (Інтерфейс Стратегії): Інтерфейс або абстрактний клас, який оголошує спільний метод для всіх підтримуваних алгоритмів.
3. ConcreteStrategy (Конкретна Стратегія): Класи, які реалізують інтерфейс Strategy, надаючи конкретну реалізацію алгоритму. Взаємодія полягає в тому, що Context делегує виконання певних операцій об'єкту ConcreteStrategy, на який він посилається.

6. Яке призначення шаблону «Стан»?

Призначення шаблону «Стан» (State) полягає в тому, щоб дозволити об'єкту змінювати свою поведінку при зміні його внутрішнього стану. Зовні це виглядає так, ніби змінюється сам клас об'єкта. Шаблон "Стан" інкапсулює поведінку, пов'язану з кожним станом, в окремі класи, а сам об'єкт-контекст делегує виконання операцій об'єкту, який представляє його поточний стан.

7. Нарисуйте структуру шаблону «Стан».



8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

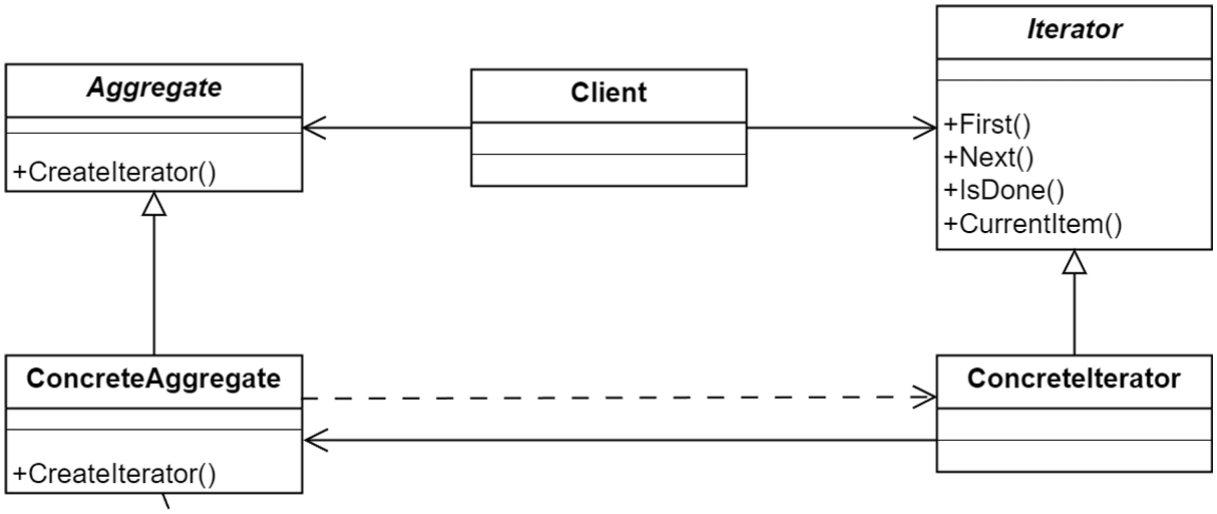
У шаблон «Стан» входять три основні класи:

1. Context (Контекст): Клас, який містить посилання на об'єкт поточного State. Він має методи, поведінка яких залежить від поточного стану.
2. State (Інтерфейс Стану): Інтерфейс або абстрактний клас, який оголошує методи, що визначають поведінку, пов'язану з різними станами.
3. ConcreteState (Конкретний Стан): Класи, які реалізують інтерфейс State, надаючи конкретну поведінку для певного стану Context. Взаємодія полягає в тому, що Context делегує запити об'єкту ConcreteState, а ConcreteState може змінювати стан об'єкта Context, встановлюючи нове посилання на інший ConcreteState.

9. Яке призначення шаблону «Ітератор»?

Призначення шаблону «Ітератор» (Iterator) полягає у наданні способу послідовного доступу до елементів складеного об'єкта (колекції), не розкриваючи його внутрішнього представлення (чи то масив, чи список, чи дерево). Він відокремлює логіку обходу від самої колекції, дозволяючи кільком обходам відбуватися одночасно і надаючи єдиний інтерфейс для обходу різних типів колекцій.

10. Нарисуйте структуру шаблону «Ітератор».



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

У шаблон «Ітератор» входять чотири основні класи:

1. Iterator (Інтерфейс Ітератора): Оголошує інтерфейс для доступу та обходу елементів колекції (наприклад, hasNext(), next()).
2. ConcreteIterator (Конкретний Ітератор): Реалізує інтерфейс Iterator і відстежує поточне положення під час обходу конкретної колекції.
3. Aggregate (Інтерфейс Агрегату): Оголошує метод createIterator(), який повертає об'єкт Iterator.
4. ConcreteAggregate (Конкретний Агрегат): Реалізує інтерфейс Aggregate і повертає екземпляр відповідного ConcreteIterator. Взаємодія полягає в тому, що клієнт отримує Ітератор від Агрегату, а потім використовує методи Ітератора для послідовного доступу до елементів Агрегату.

12. В чому полягає ідея шаблона «Одинак»?

Ідея шаблону «Одинак» (Singleton) полягає у забезпеченні існування лише одного екземпляра певного класу протягом усього життєвого циклу програми та наданні глобальної точки доступу до цього єдиного екземпляра. Шаблон використовується для класів, які повинні мати лише один координуючий об'єкт, наприклад, менеджери конфігурації, пули з'єднань з базою даних або логери.

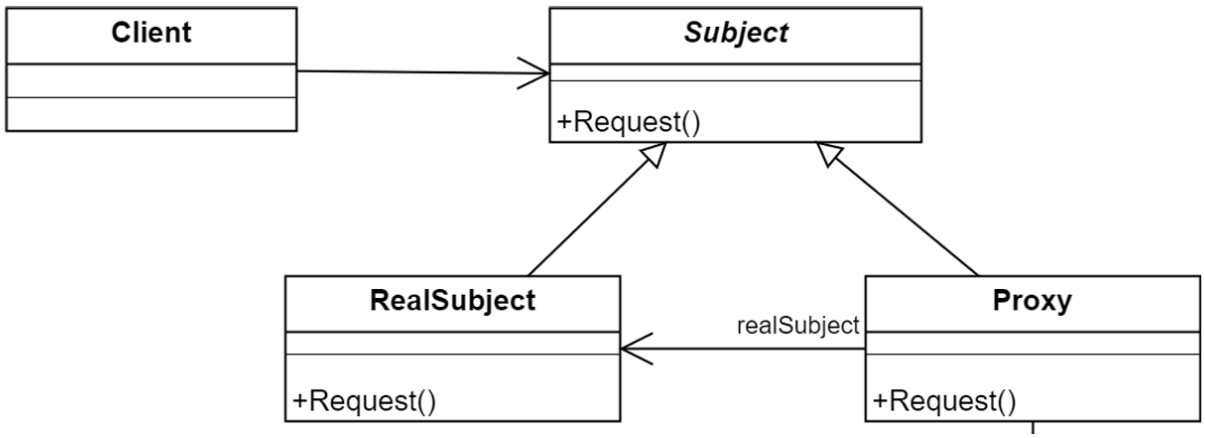
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Шаблон «Одинак» часто вважають «анти-шаблоном» тому, що він порушує принцип єдиної відповідальності (клас відповідає і за свою бізнес-логіку, і за управління єдиним екземпляром) і створює глобальний стан, який ускладнює тестування, особливо юніт-тестування (важко ізолювати код), та може призводити до прихованих залежностей і проблем у багатопотоковому середовищі, оскільки глобальний доступ ускладнює керування синхронізацією.

14. Яке призначення шаблону «Проксі»?

Призначення шаблону «Проксі» (Proxy) полягає у наданні об'єкта-заступника (суррогату) для іншого об'єкта, щоб контролювати доступ до нього. Проксі діє як інтерфейс, який обгортає оригінальний об'єкт (Service), дозволяючи додати додатковий функціонал (наприклад, ліниве завантаження, кешування, перевірка доступу або логування) перед передачею запиту до справжнього об'єкта.

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

У шаблон «Проксі» входять три основні класи:

1. Subject (Інтерфейс): Спільний інтерфейс для як RealSubject, так і Proxy, що дозволяє клієнту використовувати їх взаємозамінно.
2. RealSubject (Справжній Об'єкт): Клас, який містить основну бізнес-логіку та є оригінальним об'єктом, до якого потрібен доступ.
3. Proxy (Заступник/Проксі): Клас, який містить посилання на RealSubject і реалізує інтерфейс Subject. Він перехоплює запити від клієнта і, перш ніж передати їх RealSubject, може виконати додаткові дії (наприклад, перевірку доступу або ініціалізацію).