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 №4

Технології розроблення програмного забезпечення

ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»

Варіант №5

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

Київ 2023

Тема: Шаблони *«Singleton», «Iterator», «Proxy», «State», «Strategy»*

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).

Тепер можна перейти до завдання даної лабораторної роботи:

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.

Singleton

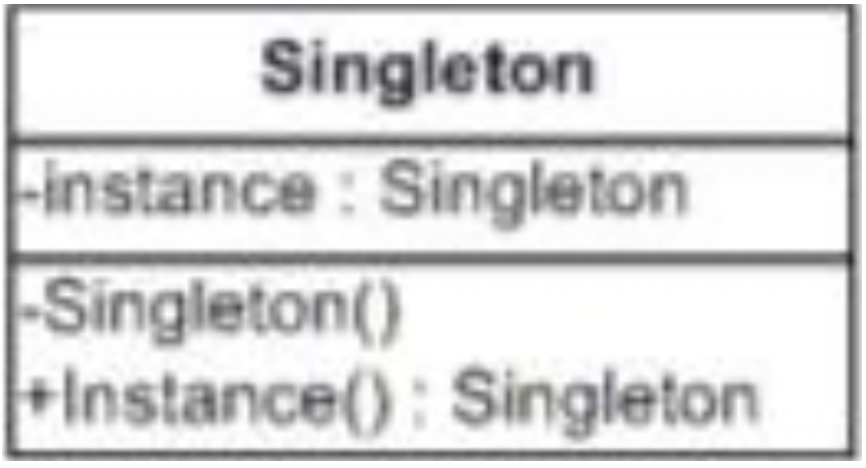

Рис. 1 - Структура “Одинака”

Даний паттерн являє собою клас в термінах ООП, який може мати не більше одного об'єкта. Також кількість об'єктів можна задати (тобто не можна створити

більш n об'єктів даного класу). Даний об'єкт найчастіше зберігається як статичне поле в самому класі.

Переваги та недоліки

В даний час «одинака» багато хто вважає т.зв. «анти-шаблоном», тобто поганою практикою проектування. Це пов'язано з тим, що «одинаки» представляють собою глобальні дані (як глобальна змінна), що мають стан. Стан глобальних об'єктів важко відслідковувати і підтримувати коректно; також глобальні об'єкти важко тестуються і вносять складність в програмний код (у всіх ділянках коду виклик в одне єдине місце з «одинаком»; при зміні підходу доведеться змінювати масу коду).

При цьому реалізація контролю доступу можлива за допомогою статичних змінних, замикань, мютексов та інших спеціальних структур.

+ Гарантує наявність єдиного екземпляра класу.

+ Надає до нього глобальну точку доступу.

- Порушує принцип єдиної відповідальності класу.

- Маскує поганий дизайн.

Iterator

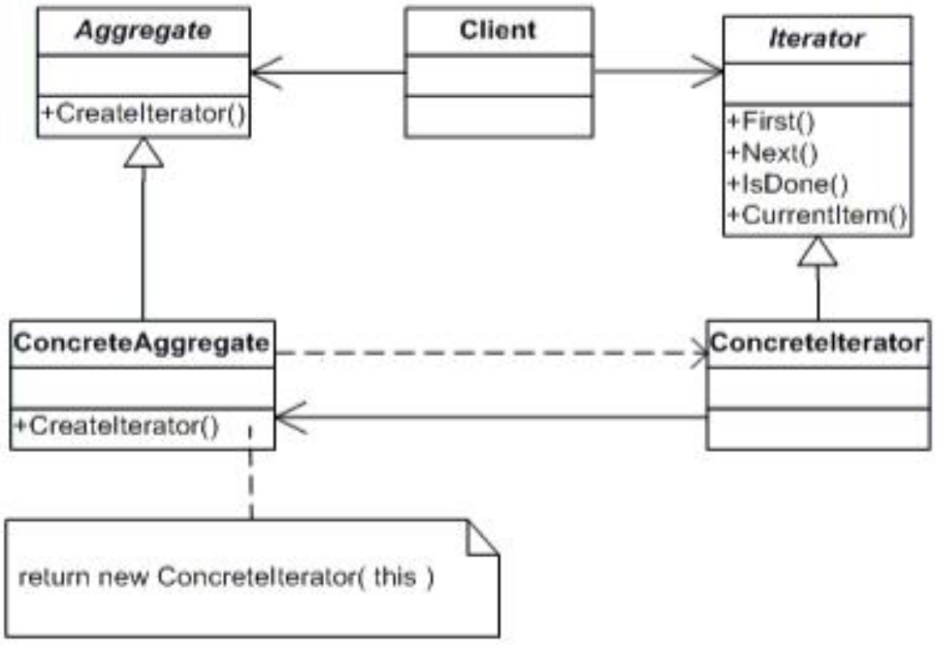

Рис. 2 - Структура паттерна Iterator

Iterator являє собою шаблон реалізації об'єкта доступу до набору (колекції, агрегату) елементів без розкриття внутрішніх механізмів реалізації. Ітератор виносить функціональність перебору колекції елементів з самої колекції, таким чином досягається розподіл обов'язків: колекція відповідає за зберігання даних, ітератор - за прохід по колекції.

При цьому алгоритм ітератора може змінюватися - при необхідності пройти в зворотньому порядку використовується інший ітератор. Можливо також написання такого ітератора, який проходить список спочатку по парних позиціях (2,4,6-й елементи і т.д.), потім по непарних. Тобто, шаблон ітератор дозволяє реалізовувати різноманітні способи проходження по колекції незалежно від виду і способу представлення даних в колекції.

Ідея патерна Ітератор полягає в тому, щоб винести поведінку обходу колекції з самої колекції в окремий клас.

Об'єкт-ітератор буде відстежувати стан обходу, поточну позицію в колекції і скільки елементів ще залишилося обійти. Одну і ту ж колекцію зможуть одночасно обходити різні ітератори, а сама колекція не буде навіть знати про це.

Переваги та недоліки:

Цей шаблон дозволяє уніфікувати операції проходження по наборам об'єктів для всіх наборів. Тобто, незалежно від реалізації (масив, зв'язаний список, незв'язаний список, дерево та ін.), кожен з наборів може використовувати будь-який з реалізованих ітераторів.

+ Дозволяє реалізувати різні способи обходу структури даних.

+ Спрощує класи зберігання даних

- Не виправданий, якщо можна обійтися простим циклом.

Proxy

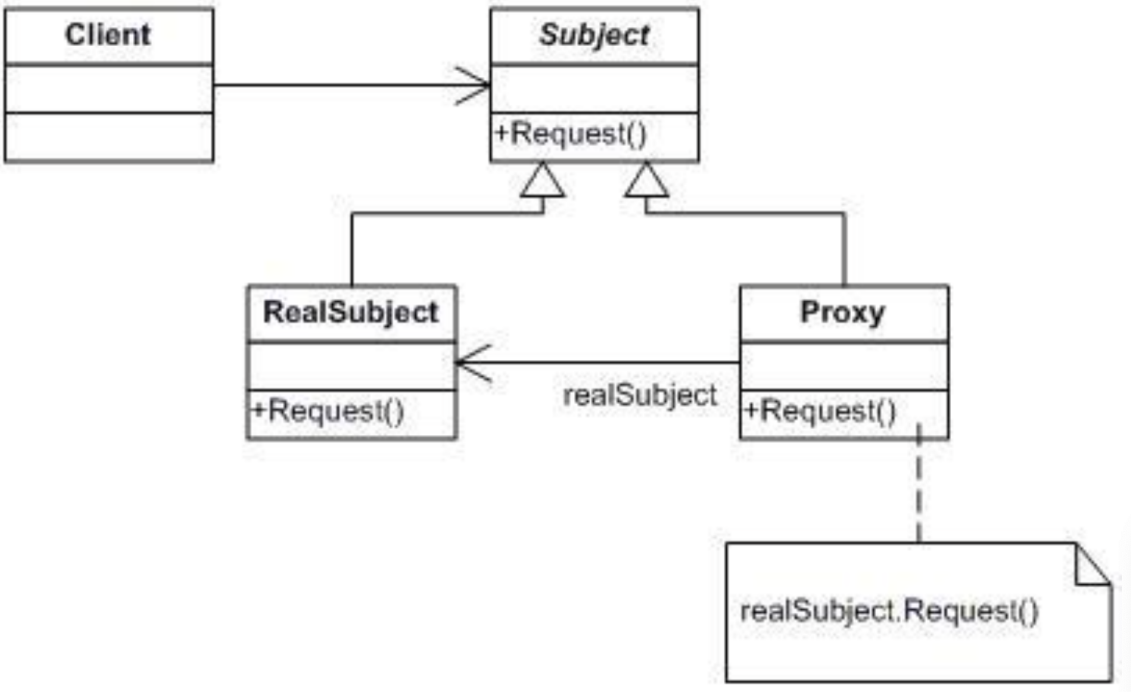

Рис. 3 - Структура паттерна Proxy

У цьому шаблоні об'єкти є об'єктами-заглушками або двійниками/замінниками для об'єктів конкретного типу. Зазвичай, проксі об'єкти вносять додатковий функціонал або спрощують взаємодію з реальними об'єктами.

Проксі об'єкти використовувалися в більш ранніх версіях інтернет-браузерів, наприклад, для відображення картинки.

Даний патерн пропонує створити новий клас-дублер, який має той же інтерфейс, що і оригінальний службовий об'єкт. При отриманні запиту від клієнта об'єкт-заступник сам би створював екземпляр службового об'єкта і переадресовував б йому всю реальну роботу. Завдяки однаковому інтерфейсу, об'єкт-заступник можна передати в будь-який код, який чекає сервісний об'єкт.

Переваги та недоліки:

+ Дозволяє контролювати сервісний об'єкт непомітно для клієнта.

+ Може працювати, навіть якщо сервісний об'єкт ще не створений.

- Ускладнює код програми через введення додаткових класів.

- Збільшує час відгуку від сервісу.

State

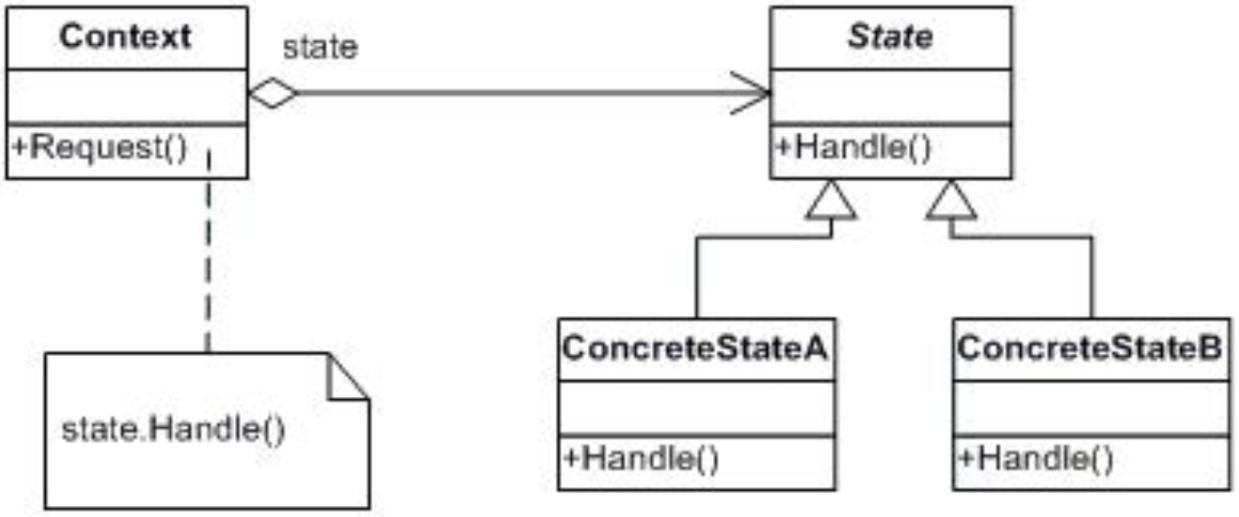

Рис. 4 - Структура паттерна State

Цей шаблон дозволяє змінювати логіку роботи об'єктів у випадку зміни їх внутрішнього стан.

Реалізація даного шаблону полягає в наступному: пов'язані зі станом поля, властивості, методи і дії виносяться в окремий загальний інтерфейс (State); кожен стан являє собою окремий клас (ConcreteStateA, ConcreteStateB), які реалізують загальний інтерфейс.

Об’єкти, що мають стан (Context), при зміні стану просто записують новий об'єкт в поле state, що призводить до повної зміни поведінки об'єкта.

Це дозволяє легко додавати в майбутньому і обробляти нові стани, відокремлювати залежні від стану елементи об'єкта в інших об'єктах, і відкрито проводити заміну стану (що має сенс у багатьох випадках).

Переваги та недоліки:

+ Позбавляє від безлічі великих умовних операторів машини станів.

+ Концентрує в одному місці код, пов'язаний з певним станом.

+ Спрощує код контексту.

- Може невиправдано ускладнити код, якщо станів мало і вони рідко змінюються.

Strategy

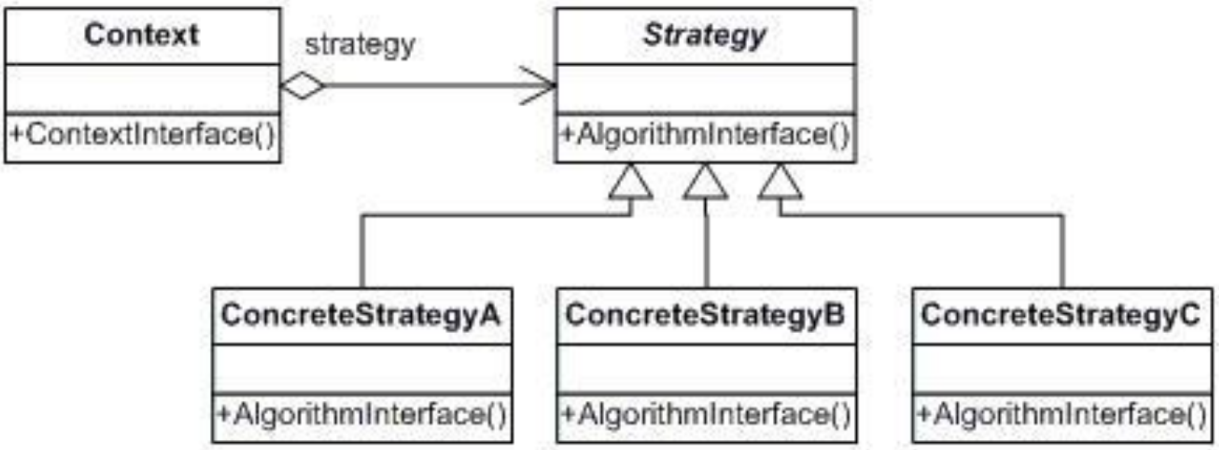

Рис. 5 - Структура паттерна Strategy

Шаблон «Strategy» (Стратегія) дозволяє змінювати деякий алгоритм поведінки об'єкта іншим алгоритмом, що досягає ту ж мету іншим способом. Прикладом можуть служити алгоритми сортування: кожен алгоритм має власну реалізацію і визначений в окремому класі; вони можуть бути взаємозамінними в об'єкті, який їх використовує.

Даний шаблон дуже зручний у випадках, коли існують різні «політики» обробки даних. По суті, він дуже схожий на шаблон «State» (Стан), проте використовується в абсолютно інших цілях - незалежно від стану об'єкта відобразити різні можливі поведінки об'єкта.

Патерн Стратегія пропонує визначити сімейство схожих алгоритмів, які часто змінюються або розширюються, і винести їх у власні класи, звані стратегіями.

Замість того, щоб початковий клас сам виконував той чи інший алгоритм, він буде грати роль контексту, посилаючись на одну із стратегій і делегуючи їй виконання роботи. Щоб змінити алгоритм, вам буде досить підставити в контекст інший об'єкт-стратегію.

Переваги та недоліки:

+ Гаряча заміна алгоритмів на льоту.

+ Ізолює код і дані алгоритмів від інших класів.

+ Відхід від спадкування до делегування.

+ Реалізує принцип відкритості / закритості.

- Ускладнює програму за рахунок додаткових класів.

- Клієнт повинен знати, в чому полягає різниця між стратегіями, щоб вибрати відповідну.

2. Застосування одного з розглянутих шаблонів при реалізації програми.

Як зазначено в завданні, необхідно реалізувати шаблон Singleton.

Під час аналізу теми та завдання проєкту стало зрозуміло, що найоптимальнішим рішенням буде застосування даного шаблону під час реалізації іншого, а саме client-server(Лабораторна робота №9). У паттерні client-server шаблон “одинак” реалізується неявно при створенні сервера - одному відкритому порту може відповідати лише один екземпляр сервера.

Висновки: Під час виконання даної лабораторної роботи ми вивчили інформацію про шаблони «singleton», «iterator», «proxy», «state», «strategy» та застосували один з них на практиці.