Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота № 4

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Вступ до паттернів проектування»

Виконав
Студент групи ІА-31:
Козир Я. О.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. **Мета:**

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

2. **Теоретичні відомості:**

Singleton: Забезпечує єдиний екземпляр класу з глобальним доступом. Використовується для унікальних ресурсів (наприклад, конфігураційні файли), але вважається антипатерном через глобальний стан.
Iterator: Дозволяє послідовний доступ до елементів колекції без розкриття її внутрішньої структури. Підтримує різні методи обходу (наприклад, у глибину, випадково).
Proxy: Діє як замінник або заглушка для іншого об'єкта, додаючи функціонал (наприклад, ледаче завантаження, контроль доступу). Зменшує кількість запитів до зовнішніх сервісів (наприклад, оптимізація DocuSign).
State: Дозволяє змінювати поведінку об'єкта залежно від його стану (наприклад, типи карток або режими системи). Використовує окремі класи для кожного стану.
Strategy: Уможливлює заміну алгоритмів поведінки об'єкта (наприклад, методи сортування чи маршрути). Відокремлює логіку алгоритмів від контексту для гнучкості.

3. **Хід роботи:**

Тема :

21. **Online radio station** (iterator, adapter, factory method, facade, visitor, client-server)

Додаток повинен служити сервером для радіостанції з можливістю мовлення на радіостанцію (64, 92, 128, 196, 224 kb/s) в потоковому режимі; вести облік підключених користувачів і статистику відвідувань і прослуховувань; налаштувати папки з піснями і можливість вести списки програвання або playlists (не відтворювати всі пісні).

Для системи реалізовано патерн Iterator для керування чергою відтворення. DJ формує плейлист => PlaybackQueue => IPlaylistIterator обходить треки => StreamingService відтворює по черзі.

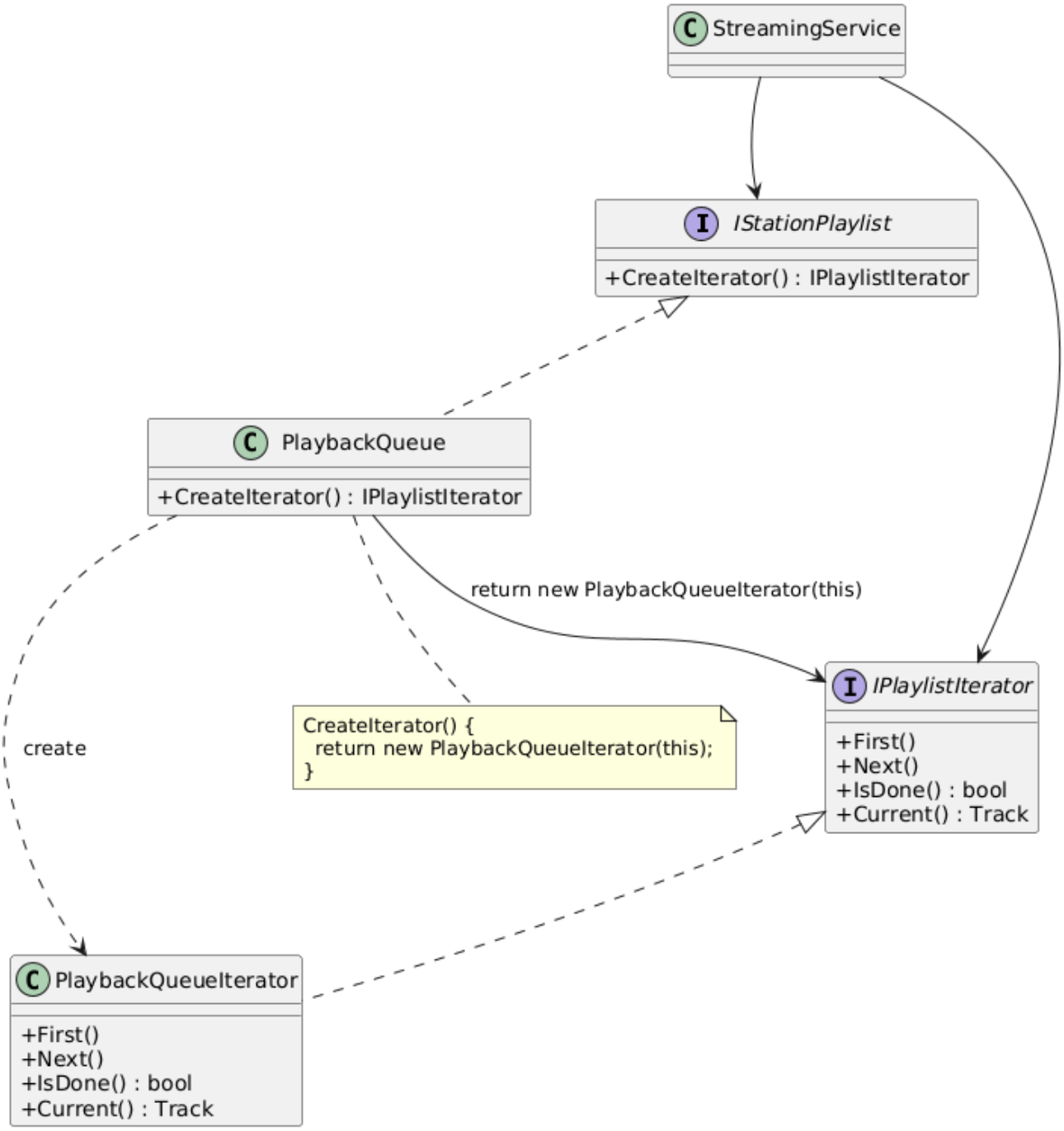


Рисунок 1 - Структура патерну Iterator

IStationPlaylist – це інтерфейс, який описує метод CreateIterator(). Він задає контракт для всіх колекцій, що підтримують ітерацію (наприклад, черги відтворення).

PlaybackQueue – це конкретна реалізація колекції. Вона містить список треків, відсортованих за позицією в черзі (QueuePosition), і повертає ітератор для послідовного обходу.

IPlaylistIterator – це інтерфейс ітератора, який визначає методи First(), Next(), IsDone() та Current() для керування процесом обходу.

PlaybackQueueIterator – це конкретна реалізація ітератора. Вона внутрішньо зберігає поточний індекс і забезпечує послідовний доступ до треків у PlaybackQueue.

StreamingService – це контекст (клієнт), який використовує ітератор. Він не знає, як саме влаштована колекція, а лише викликає методи ітератора для відтворення треків по черзі під час стримінгу.

В результаті можна легко обходити будь-яку колекцію треків (з БД, чи API), не змінюючи код стримінгу. Це і є суть Ітератора – відокремлення алгоритму обходу від структури даних, що забезпечує гнучкість і підтримуваність системи.

```
namespace OnlineRadioStation.Domain
{
    1 reference
    public interface IStationPlaylist
    {
        2 references
        IPlaylistIterator CreateIterator();
    }
}
```

Рисунок 2 - IStationPlaylist.cs

```
namespace OnlineRadioStation.Domain
{
    3 references
    public interface IPlaylistIterator
    {
        2 references
        void First();
        2 references
        void Next();
        3 references
        bool IsDone();
        2 references
        Track Current();
    }
}
```

Рисунок 3 - IPlaylistIterator.cs

```
using System;
using System.ComponentModel.DataAnnotations;

namespace OnlineRadioStation.Domain
{
    10 references
    public partial class PlaybackQueue
    {
        [Key]
        0 references
        public Guid QueueId { get; set; }
        0 references
        public Guid TrackId { get; set; }
        0 references
        public Guid StationId { get; set; }
        0 references
        public Guid AddedById { get; set; }
        0 references
        public int QueuePosition { get; set; }

        0 references
        public Track Track { get; set; } = null!;
        0 references
        public RadioStationEntity Station { get; set; } = null!;
        0 references
        public User AddedBy { get; set; } = null!;
    }
}
```

Рисунок 4 - PlaybackQueue.cs (partial)

```
namespace OnlineRadioStation.Domain
{
    2 references
    public class PlaybackQueueIterator : IPlaylistIterator
    {
        3 references
        private readonly List<Track> _tracks;
        4 references
        private int _currentIndex = 0;

        1 reference
        public PlaybackQueueIterator(List<Track> tracks)
        {
            _tracks = tracks.OrderBy(t => t.QueuePosition).ToList();
        }

        2 references
        public void First() => _currentIndex = 0;

        2 references
        public void Next() => _currentIndex++;

        3 references
        public bool IsDone() => _currentIndex >= _tracks.Count;

        2 references
        public Track Current()
        {
            if (IsDone())
                throw new InvalidOperationException("Кінець плейлиста");
            return _tracks[_currentIndex];
        }
    }
}
```

Рисунок 5 - PlaybackQueueIterator.cs

```
public class StreamingService
{
    1 reference
    public void StartStreaming(PlaybackQueue queue)
    {
        var iterator = queue.CreateIterator();
        iterator.First();

        Console.WriteLine("=== Початок стримінгу ===");
        while (!iterator.IsDone())
        {
            var track = iterator.Current();
            Console.WriteLine($"Відтворюється: {track.Title} ({track.Duration})");
            // Тут буде FFmpeg
            iterator.Next();
        }
        Console.WriteLine("=== Стрім завершено ===");
    }
}
```

Рисунок 6 – StreamingService.cs

```
=== Початок стримінгу ===
Відтворюється: Song 1 (00:03:00)
Відтворюється: Song 2 (00:04:00)
=== Стрім завершено ===
```

Рисунок 7 – Результат виконання

4. Висновок

У ході виконання лабораторної роботи було розглянуто базові шаблони проєктування, зокрема Singleton, Iterator, Proxy, State та Strategy. Отримані знання допомогли зрозуміти їхню роль у створенні гнучкої та масштабованої архітектури. На прикладі веб-застосунку «Online Radio Station» було реалізовано шаблон Iterator для організації послідовного обходу черги відтворення (PlaybackQueue). Логіка обходу винесена в окремий ітератор (PlaybackQueueIterator), що дозволило відокремити алгоритм проходження від структури даних. Такий підхід спрощує керування плейлистом, робить код StreamingService чистішим і дає змогу легко розширювати функціонал (наприклад, додати ітерацію по БД чи кешу) без зміни основної логіки стримінгу. Це підтверджує ефективність патернів проєктування як інструменту для побудови зрозумілого, підтримуваного та готового до розвитку коду.

5. **Контрольні питання:**

1. Що таке шаблон проєктування?

Шаблон проєктування — це універсальне рішення для типових проблем у розробці ПЗ, яке описує структуру та взаємодію об’єктів.

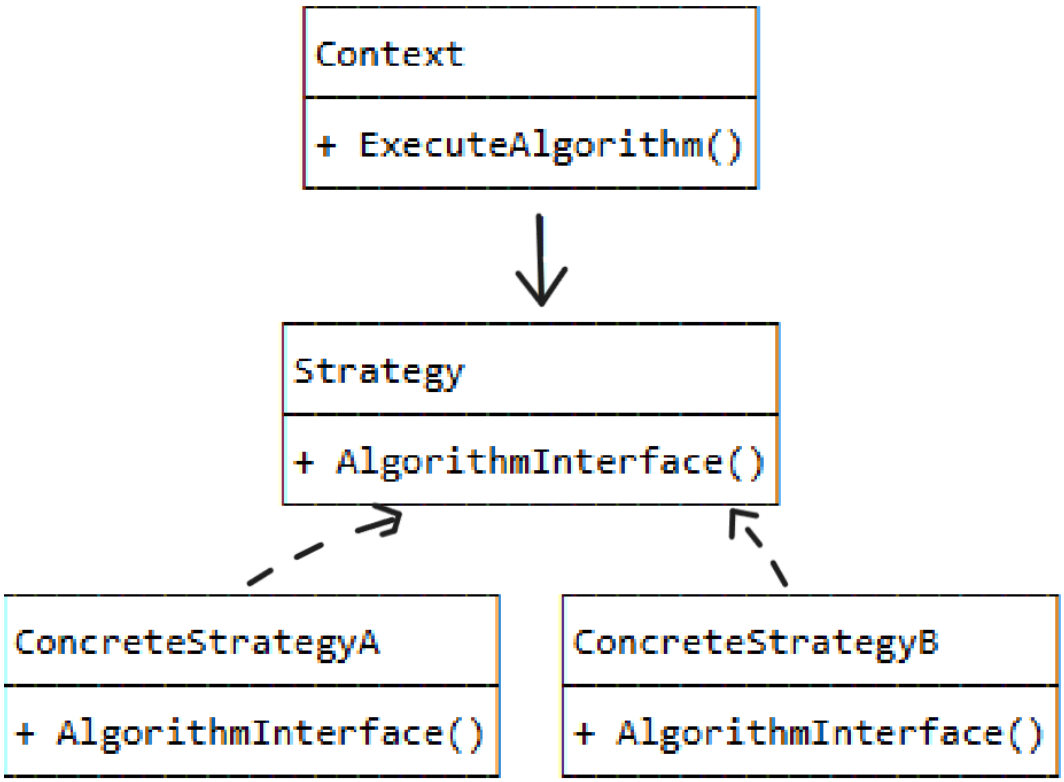
2. Навіщо використовувати шаблони проєктування?

- Спрощують розробку.
- Покращують читабельність і масштабування коду.
- Допомагають уникнути помилок.
- Забезпечують гнучкість і повторне використання.

3. Яке призначення шаблону «Стратегія»?

Шаблон Стратегія дозволяє динамічно вибирати алгоритми, інкапсулюючи їх у взаємозамінні класи, щоб уникнути умовних конструкцій.

6. Нарисуйте структуру шаблону «Стратегія».



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

Класи:

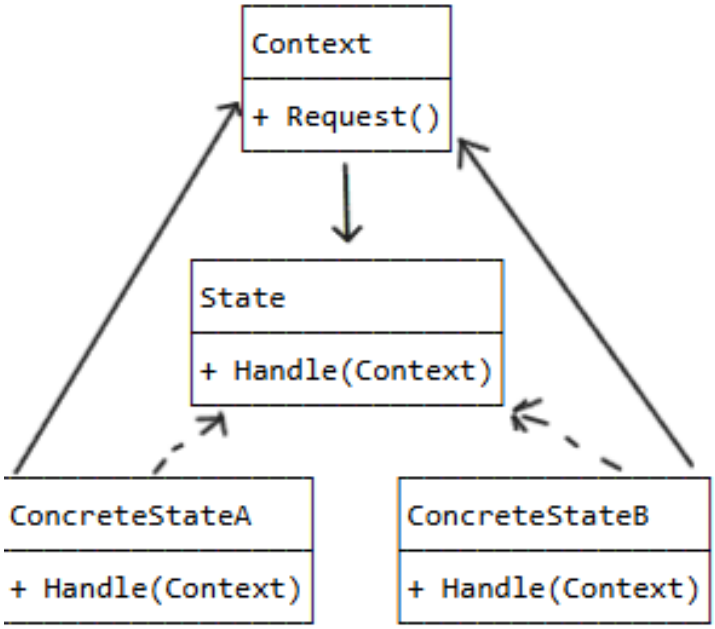
- Strategy: Інтерфейс із методом algorithm().
- ConcreteStrategy: Реалізації алгоритмів.
- Context: Використовує Strategy через setStrategy() і викликає algorithm().

Взаємодія: Контекст делегує виконання алгоритму об’єкту Strategy, який клієнт встановлює динамічно.

6. Яке призначення шаблону «Стан»?

Шаблон «Стан» дозволяє об’єкту змінювати поведінку залежно від стану, інкапсулюючи стани в окремі класи.

7. Нарисуйте структуру шаблону «Стан».



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

Класи:

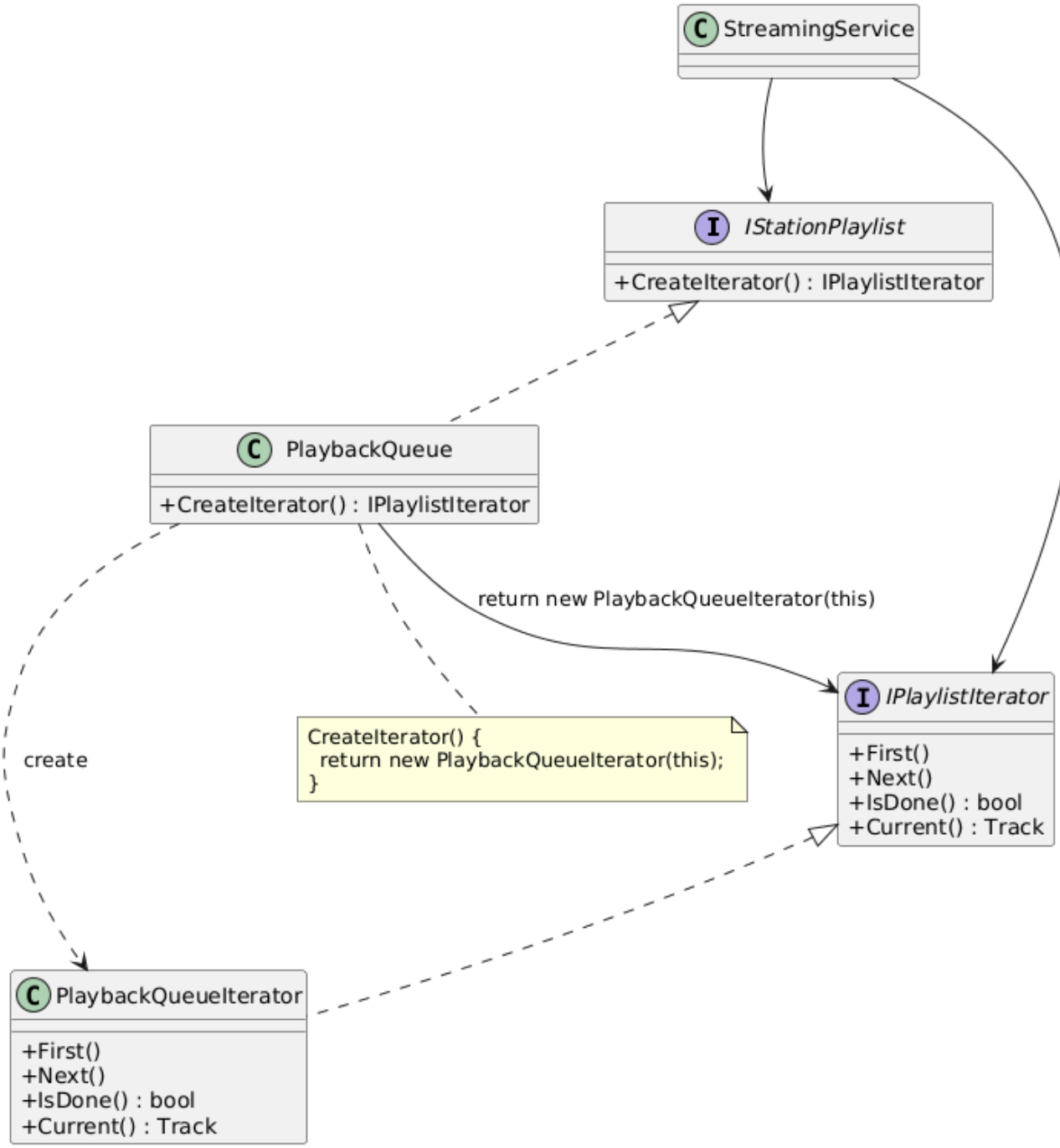
- State: Інтерфейс із методом handle().
- ConcreteState: Реалізації поведінки для стану.
- Context: Зберігає стан, делегує виконання handle().

Взаємодія: Контекст викликає handle() поточного стану, який може змінити стан через setState().

9. Яке призначення шаблону «Ітератор»?

Шаблон «Ітератор» забезпечує послідовний доступ до елементів колекції, приховуючи її внутрішню структуру.

10. Нарисуйте структуру шаблону «Ітератор».



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

Класи:

- Iterator: Інтерфейс із методами next(), hasNext().
- ConcreteIterator: Реалізація обходу.
- Aggregate: Інтерфейс із createIterator().
- ConcreteAggregate: Створює ConcreteIterator.

Взаємодія: Клієнт отримує ітератор через createIterator() і використовує його для обходу колекції.

12. В чому полягає ідея шаблона «Одинак»?

Шаблон «Одинак» забезпечує єдиний екземпляр класу з глобальним доступом до нього.

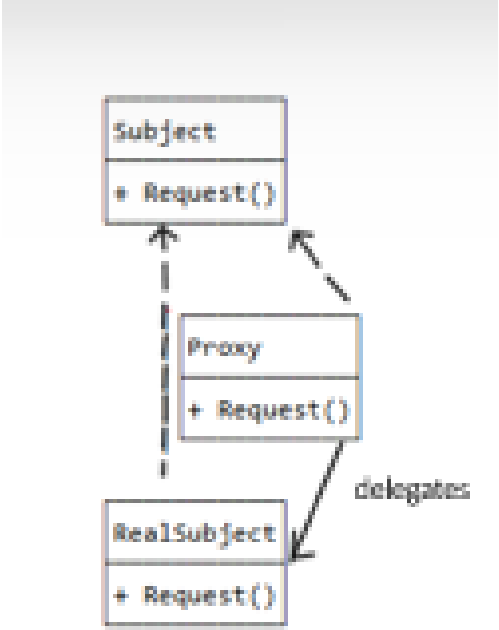
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Бо він порушує принципи ООП: ускладнює тестування, створює приховані залежності та глобальний стан

14. Яке призначення шаблону «Проксі»?

Шаблон «Проксі» контролює доступ до об’єкта, додаючи функціонал, як-от ледарська ініціалізація чи перевірка прав.

15. Нарисуйте структуру шаблону «Проксі».



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

Класи:

- Subject: Інтерфейс із методом request().
- RealSubject: Виконує основну роботу.
- Proxy: Контролює доступ до RealSubject.

Взаємодія: Клієнт викликає request() через Proxy, який делегує виклик до RealSubject або додає логіку.