



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:	3
2. Теоретичні відомості:.....	3
3. Хід роботи:.....	3
4. Висновок	8
5. Контрольні питання:	8

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 заміник 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: Уможливорює 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).

- 1) Ознайомитись з короткими теоретичними відомостями.
- 2) Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
- 3) Реалізувати один з розглянутих шаблонів за обраною темою.
- 4) Реалізувати не менше 3-х класів відповідно до обраної теми.

- 5) Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

Станція (RadioStationEntity) містить складну колекцію Playbacks (зв'язуюча таблиця), але клієнтській частині (сторінці прослуховування) потрібен простий послідовний список треків (Track). Ми не хочемо розкривати внутрішню структуру списку List<PlaybackQueue> клієнту. Реалізовано ітератор PlaybackQueueIterator, який дозволяє послідовно отримувати об'єкти Track без необхідності знати, як вони зберігаються в пам'яті. Клієнт (Listen.cshtml) використовує єдиний інтерфейс IPlaylistIterator для відображення плейлиста та навігації."

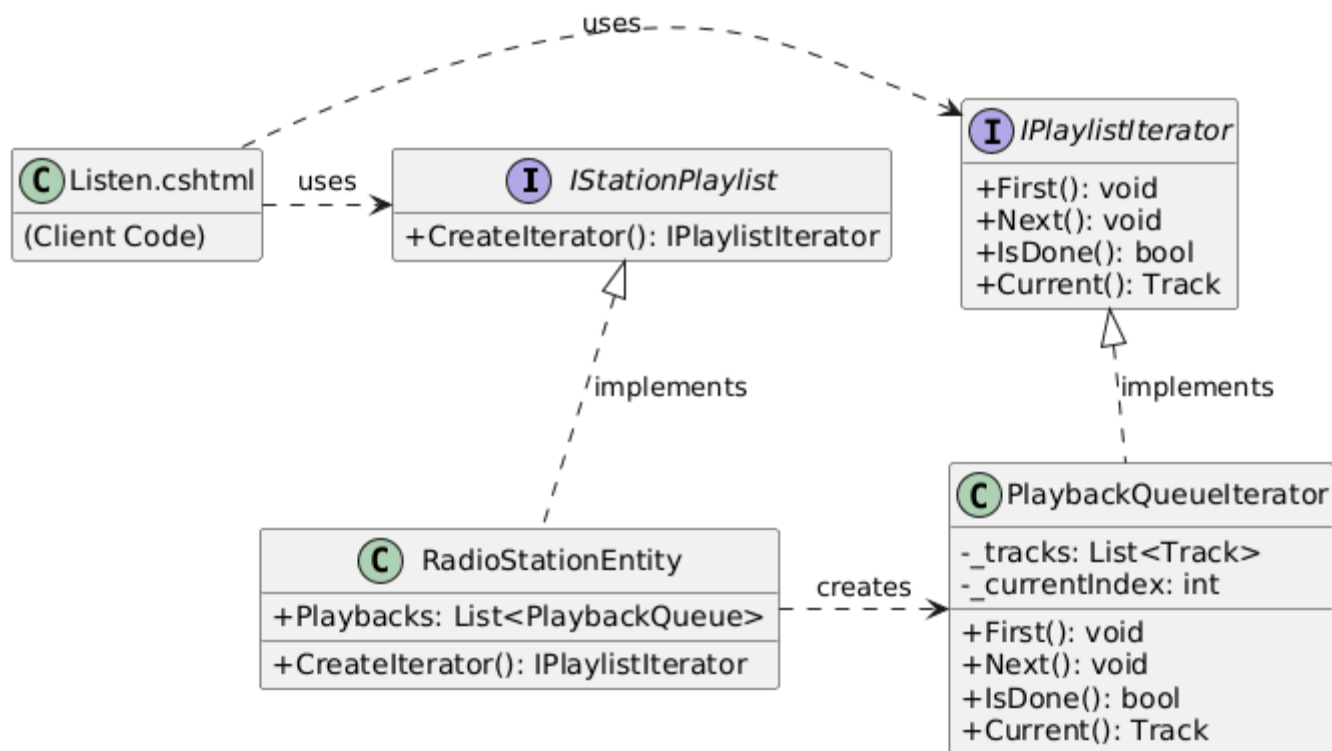


Рисунок 1 - Структура патерну Iterator

IStationPlaylist (Aggregate): Інтерфейс, що оголошує метод створення ітератора.

RadioStationEntity (ConcreteAggregate): Конкретна сутність станції, яка зберігає колекцію записів і створює для неї ітератор.

IPlaylistIterator (Iterator): Інтерфейс для обходу елементів (First, Next, IsDone, Current).

PlaybackQueueIterator (ConcreteIterator): Реалізує логіку обходу, перетворюючи внутрішню структуру PlaybackQueue на послідовність Track.

Listen.cshtml (Client): Клієнтський код, який використовує ітератор для відображення списку пісень користувачу.

Програмна реалізація:

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

Рисунок 2 - Інтерфейс ітератора IPlaylistIterator.cs

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

Рисунок 3 - Агрегат (Інтерфейс) IStationPlaylist.cs

```

namespace OnlineRadioStation.Domain
{
    3 references
    public class PlaybackQueueIterator : IPlaylistIterator
    {
        3 references
        private readonly List<Track> _tracks;
        4 references
        private int _currentIndex = 0;
        2 references
        public PlaybackQueueIterator(List<Track> tracks)
        {
            _tracks = tracks;
        }
        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 4 - Конкретний ітератор PlaybackQueue.cs

```

public class RadioStationEntity : IStationPlaylist
{
    [Key]
    15 references
    public Guid StationId { get; set; }
    16 references
    public string StationName { get; set; } = string.Empty;
    13 references
    public string Description { get; set; } = string.Empty;
    4 references
    public Guid CreatedById { get; set; }
    0 references
    public User CreatedBy { get; set; } = null!;
    5 references
    public ICollection<PlaybackQueue> Playbacks { get; set; } = new List<PlaybackQueue>();
    0 references
    public ICollection<SavedStation> SavedByUsers { get; set; } = new List<SavedStation>();
    0 references
    public ICollection<DjStream> DjStreams { get; set; } = new List<DjStream>();

    1 reference
    public IPlaylistIterator CreateIterator()
    {
        var tracks = this.Playbacks
            .OrderBy(p => p.QueuePosition)
            .Select(p => p.Track)
            .ToList();
        return new PlaybackQueueIterator(tracks);
    }
}

```

Рисунок 6 – RadioStationEntity.cs

```

if (tracksList != null)
{
    OnlineRadioStation.Domain.IPlaylistIterator iterator =
        new OnlineRadioStation.Domain.PlaybackQueueIterator(tracksList);

    iterator.First();
    int position = 1;

    while (!iterator.IsDone())
    {
        var Track? track = iterator.Current();

        if (!string.IsNullOrEmpty(track.HlsUrl))
        {
            int rating = userRatings.ContainsKey(track.TrackId) ? userRatings[track.TrackId] : 0;
            var (int Likes, int Dislikes) votes = _userService.GetTrackVotesAsync(track.TrackId).Result;
            playlistForJs.Add(new {
                trackId = track.TrackId.ToString(),
                title = track.Title,
                url = track.HlsUrl,
                id = position,
                userRating = rating,
                likesCount = votes.Likes,
                dislikesCount = votes.Dislikes
            });
            position++;
        }

        iterator.Next();
    }
}

```

Рисунок 7 – Клієнт Listen.html

Поточний ефір

Черга відтворення на станції.

#	Сорт	Назва треку	Тривалість	Налаштування
1	▼	Yogamaf & Sorrybutwhy - vaibek	01:44	⌂ Скінути
2	▲ ▼	Weeknd - Lost in the fire	03:14	⌂ Скінути
3	▲	Travis Scott - Sicko Mode	05:17	⌂ Скінути

Рисунок 8 – Відображення треків в поточному ефірі від імені діджея (результат)

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 списку треків станції. Логіка обходу винесена в окремий клас PlaybackQueueIterator, що дозволило відокремити алгоритм проходження від моделі даних RadioStationEntity. Такий підхід дозволив використати універсальний механізм обходу при формуванні HTML-сторінки для слухача (Listen.cshtml), не прив'язуючись до внутрішньої реалізації зберігання черги в базі даних.

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.

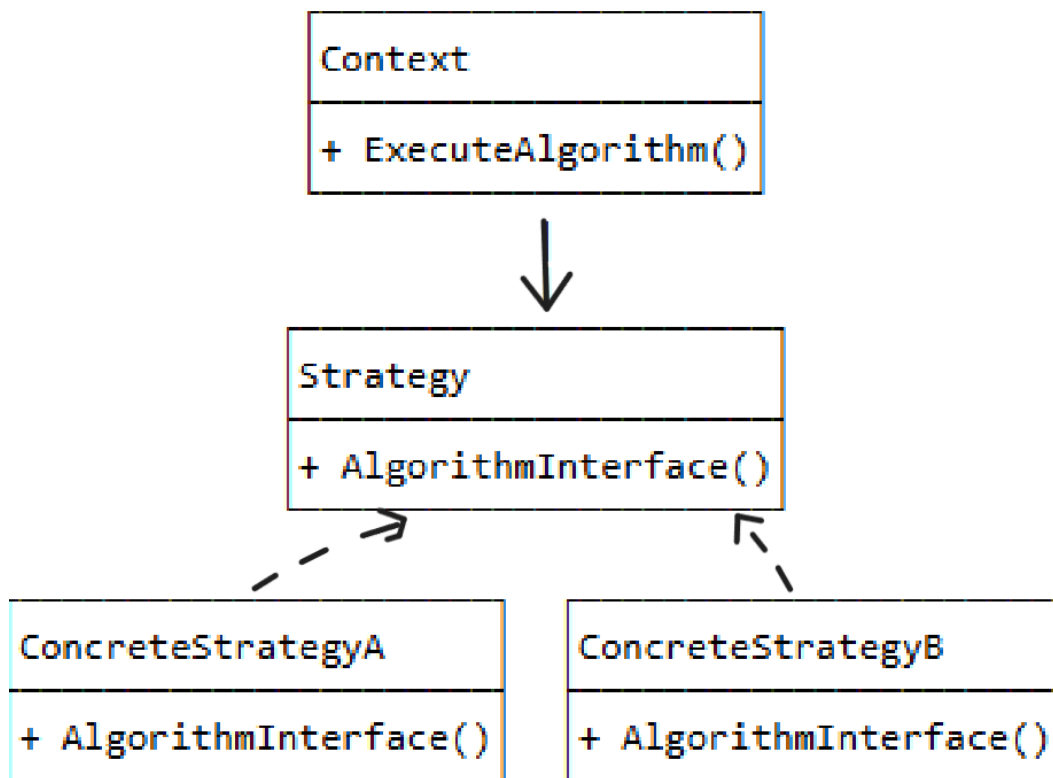
2. Навіщо використовувати шаблони проєктування?

- 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:

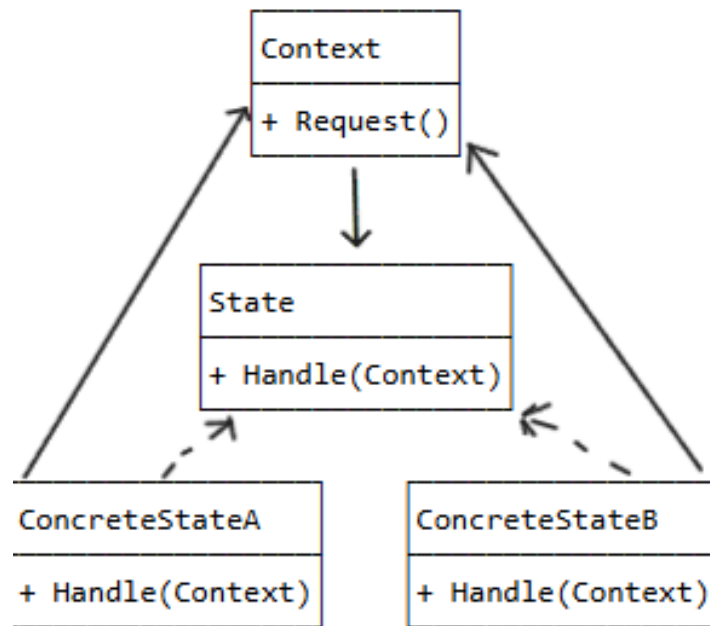
- Strategy: Інтерфейс із методом algorithm().
- ConcreteStrategy: Реалізації алгоритмів.
- Context: Використовує Strategy через setStrategy() і викликає algorithm().

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:

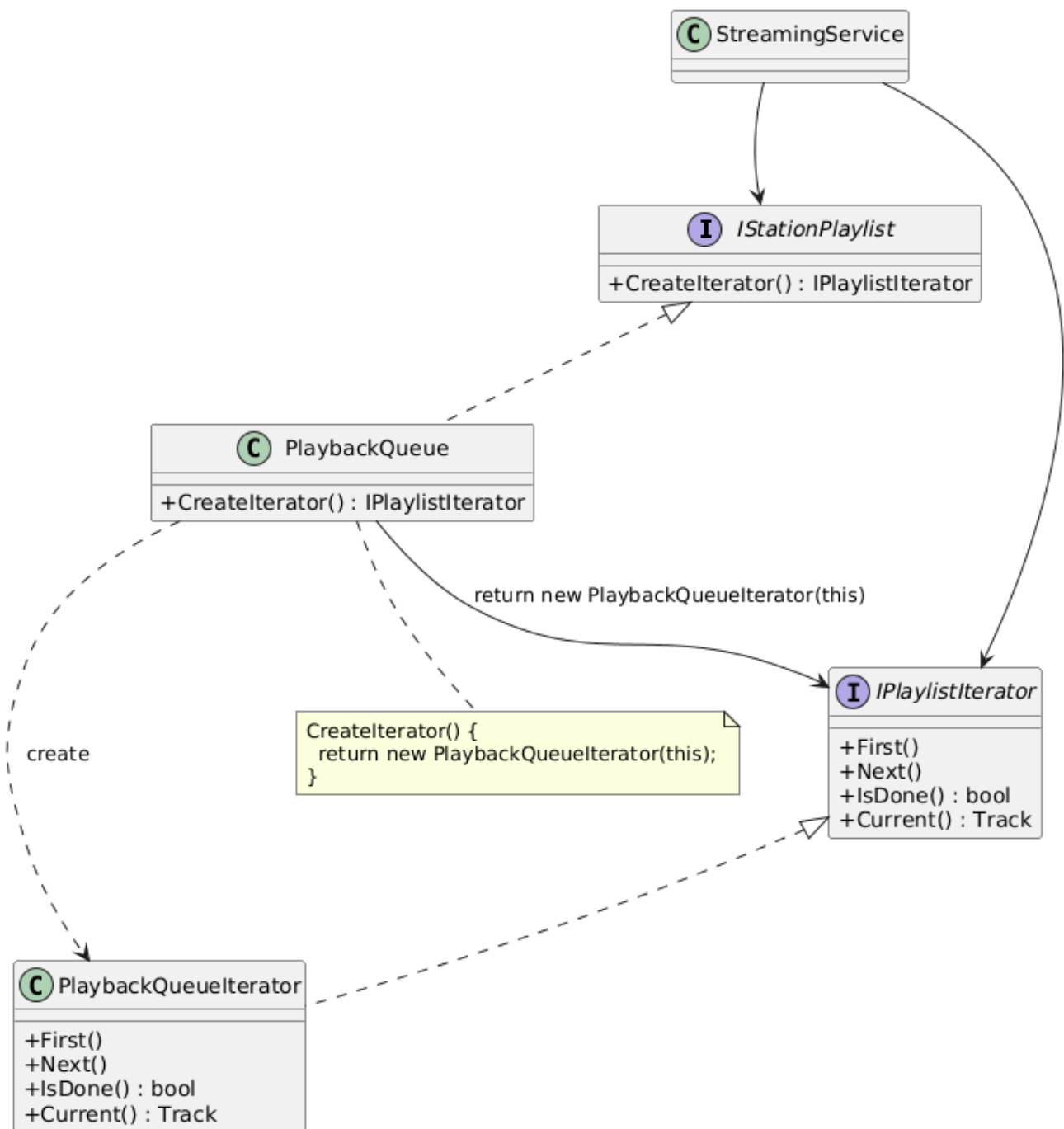
- State: Інтерфейс із методом handle().
- ConcreteState: Реалізації поведінки для стану.
- Context: Зберігає стан, делегує виконання handle().

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 шаблону «Одинак»?

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

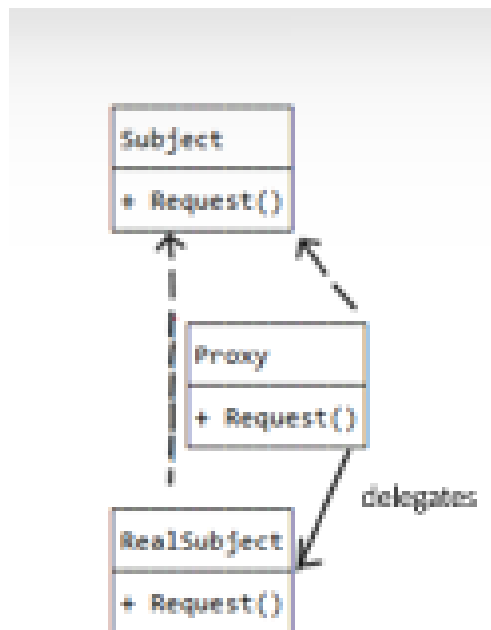
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

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.