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

із дисципліни «*Технології розроблення програмного забезпечення*»

Вступ до паттернів проектування

Аудіоредактор

Виконала
студентка групи ІА–34
Кузьменко В.С.

Перевірив
викладач
Мягкий М.Ю.

Київ 2025

ЗМІСТ

Тема: Вступ до паттернів проектування.

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

Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати один з розглянутих шаблонів за обраною темою.

• Реалізувати не менше 3-х класів відповідно до обраної теми.

• 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.

Тема роботи:

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).

Теоретичні відомості

Шаблони проєктування - це формалізовані рішення завдань у розробці, які описують, як ефективно вирішити проблему і де краще застосувати це рішення. Вони базуються на багаторічному досвіді розробників і спрощують роботу з системами. Наприклад, застосування шаблонів робить систему більш зрозумілою, стійкою до змін і легкою для подальшої інтеграції та підтримки. Водночас це є уніфікованою мовою для спілкування між розробниками. Шаблони корисні, але їх слід застосовувати обдумано, оскільки надмірне використання може погіршити дизайн.

Шаблон Singleton забезпечує створення тільки одного екземпляра класу і надає глобальну точку доступу до нього. Його використовують, коли потрібен єдиний об'єкт для управління спільними ресурсами, наприклад, налаштуваннями програми.

Шаблон Iterator дозволяє перебирати елементи колекції без розкриття її внутрішньої реалізації. Він виділяє логіку обходу в окремий клас, що робить код колекції простішим. Наприклад, для складних структур, як дерева, ітератор забезпечує послідовний обхід, навіть якщо спосіб обходу змінюється.

Шаблон Proxy створює об'єкт-замінник, який виконує додаткову логіку до або після звернення до реального об'єкта. Це корисно для відкладеної ініціалізації або доступу до ресурсоємних об'єктів. Наприклад, платіжна картка є проксі для готівки: забезпечує ту саму функцію, але зручніше у використанні.

Шаблон State змінює поведінку об'єкта залежно від його стану. Наприклад, банківська картка може нараховувати різні відсотки залежно від свого типу (Classic, Platinum). State допомагає уникнути великої кількості умов у коді, розділяючи стани в окремі класи.

Strategy (Стратегія) - це патерн проєктування, який дозволяє визначити сімейство схожих алгоритмів, інкапсулювати кожен з них у власний клас і зробити їх взаємозамінними. Стратегія дозволяє легко додавати нові алгоритми

або змінювати існуючі, не змінюючи клієнтський код, що використовує ці алгоритми.

Хід роботи

1. Ознайомитись з теоретичними відомостями.
2. Реалізувати частину функціоналу програми у вигляді класів та їх взаємодії.
3. Застосувати один із шаблонів проєктування.
4. Реалізувати щонайменше три класи за обраною темою.
5. Підготувати звіт з діаграмою класів і фрагментами коду.

Шаблон Singleton

У даній роботі використано шаблон Singleton, який забезпечує створення лише одного екземпляра конвертора протягом роботи програми. Завдяки цьому під час конвертації аудіофайлів не створюються нові копії об’єкта - використовується єдиний спільний екземпляр. Такий підхід запобігає помилкам, пов’язаним із множинними копіями, та забезпечує централізований контроль доступу до конвертора.

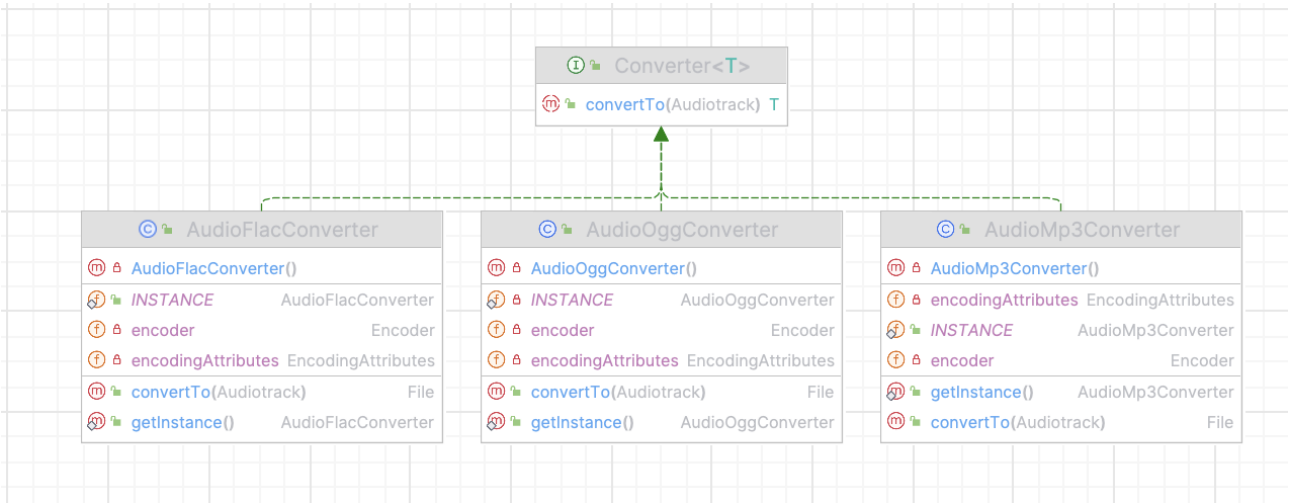


Рис. 1 Singleton для трьох класів-конвертерів

Converter<T> - це інтерфейс, який визначає метод convertTo(Audiotrack audiotrack), спільний для всіх типів конвертерів. Кожен конвертер містить такі елементи:

- INSTANCE - статичний екземпляр класу, що гарантує створення лише одного об’єкта, реалізуючи принцип шаблону Singleton.
- encoder та encodingAttributes - спільні ресурси для кодування аудіофайлів, збережені на рівні класу, щоб уникнути повторної ініціалізації.
- getInstance() - метод, який перевіряє наявність створеного екземпляра та створює його лише за потреби.

Усі конвертери реалізують власний метод convertTo(Audiotrack audiotrack), що виконує конвертацію аудіофайлів у відповідні формати - FLAC, MP3 або OGG.

Код системи

Лістинг 1- Interface Converter

```
package org.audioeditor.converter;

import org.audioeditor.audiotrack.Audiotrack;

public interface Converter<T> {
    T convertTo(Audiotrack audiotrack);
}
```

Лістинг 2- Class AudioFlacConverter

```
package org.audioeditor.converter;

import it.sauronsoftware.jave.Encoder;
import it.sauronsoftware.jave.EncoderException;
import it.sauronsoftware.jave.EncodingAttributes;
import org.audioeditor.audiotrack.Audiotrack;

import java.io.File;

public class AudioFlacConverter implements Converter<File> {
    public static AudioFlacConverter INSTANCE;
    private EncodingAttributes encodingAttributes;
    private Encoder encoder;

    private AudioFlacConverter() {
        encodingAttributes = new EncodingAttributes();
        encodingAttributes.setFormat("flac");
        encoder = new Encoder();
    }
```

```
    public static AudioFlacConverter getInstance() {
        if (INSTANCE == null) {
            INSTANCE = new AudioFlacConverter();
        }
        return INSTANCE;
    }

    @Override
    public File convertTo(Audiotrack audiotrack) {
        encodingAttributes.setAudioAttributes(audiotrack.getAudioAttributes());

        File convertedFlac = new File(audiotrack.getFileLink().getParent() +
"\\" + audiotrack.getFileLink().getName() + " (converted to flac).flac");
        try {
            encoder.encode(audiotrack.getFileLink(), convertedFlac,
encodingAttributes);
        } catch (EncoderException e) {
            throw new RuntimeException(e);
        }

        return convertedFlac;
    }
}
```

Лістинг 3- Class AudioMp3Converter

```
package org.audioeditor.converter;

import it.sauronsoftware.jave.Encoder;
import it.sauronsoftware.jave.EncoderException;
import it.sauronsoftware.jave.EncodingAttributes;
import org.audioeditor.audiotrack.Audiotrack;

import java.io.File;
import java.io.IOException;

public class AudioMp3Converter implements Converter<File> {
    private static AudioMp3Converter INSTANCE;
    private EncodingAttributes encodingAttributes;
    private Encoder encoder;

    private AudioMp3Converter() {
        encodingAttributes = new EncodingAttributes();
        encodingAttributes.setFormat("mp3");
        encoder = new Encoder();
    }

    public static AudioMp3Converter getInstance() {
        if (INSTANCE == null) {
            INSTANCE = new AudioMp3Converter();
        }
        return INSTANCE;
    }

    @Override
    public File convertTo(Audiotrack audiotrack) {
        encodingAttributes.setAudioAttributes(audiotrack.getAudioAttributes());

        try {
            File convertedMp3 = new File(audiotrack.getFileLink().getParent() +
"\\" + audiotrack.getFileLink().getName() + " (converted to mp3).mp3");
            convertedMp3.createNewFile();
            encoder.encode(audiotrack.getFileLink(), convertedMp3,
```

```
encodingAttributes);
            return convertedMp3;
        } catch (EncoderException | IOException e) {
            throw new RuntimeException(e);
        }

    }
}
```

Лістинг 4- Class AudioOggConverter

```
package org.audioeditor.converter;

import it.sauronsoftware.jave.Encoder;
import it.sauronsoftware.jave.EncoderException;
import it.sauronsoftware.jave.EncodingAttributes;
import org.audioeditor.audiotrack.Audiotrack;

import java.io.File;

public class AudioOggConverter implements Converter<File> {
    private static AudioOggConverter INSTANCE;
    private EncodingAttributes encodingAttributes;
    private Encoder encoder;

    private AudioOggConverter() {
        encodingAttributes = new EncodingAttributes();
        encodingAttributes.setFormat("ogg");
        encoder = new Encoder();
    }

    public static AudioOggConverter getInstance() {
        if (INSTANCE == null) {
            INSTANCE = new AudioOggConverter();
        }
        return INSTANCE;
    }

    @Override
    public File convertTo(Audiotrack audiotrack) {
        encodingAttributes.setAudioAttributes(audiotrack.getAudioAttributes());

        File convertedOgg = new File(audiotrack.getFileLink().getParent() + "\\"
+ audiotrack.getFileLink().getName() + " (converted to ogg).ogg");
        try {
            encoder.encode(audiotrack.getFileLink(), convertedOgg,
encodingAttributes);
        } catch (EncoderException e) {
            throw new RuntimeException(e);
        }

        return convertedOgg;
    }
}
```

Висновок

У ході виконання лабораторної роботи було досліджено та реалізовано шаблон проєктування Singleton на прикладі класів-конвертерів аудіофайлів. Було проаналізовано принципи його роботи, основні переваги та обмеження при застосуванні в програмних системах.

У результаті реалізовано три класи-конвертери - AudioFlacConverter, AudioMp3Converter і AudioOggConverter, що забезпечують конвертацію аудіофайлів у відповідні формати. Завдяки використанню шаблону Singleton для кожного конвертера створюється лише один екземпляр, що дозволяє оптимізувати споживання ресурсів та забезпечити централізоване керування процесом конвертації.

Базовий узагальнений інтерфейс Converter<T> забезпечив уніфікацію реалізації та розширюваність системи для майбутніх форматів аудіо.

Використання Singleton має низку переваг: гарантію створення єдиного екземпляра класу, спрощений доступ через метод *getInstance()* та можливість централізованого контролю стану об’єкта. Водночас шаблон може ускладнювати тестування і потребує додаткових механізмів синхронізації в багатопотокових додатках.

Загалом, застосування шаблону Singleton у цій роботі продемонструвало його ефективність для організації стабільної, керованої та ресурсоефективної взаємодії між об’єктами системи. Отримані результати та набуті навички можуть бути використані при подальшій розробці програмного забезпечення, що потребує єдиних точок доступу до важливих компонентів.

Відповіді на теоретичні питання

1. Що таке шаблон проєктування?

Шаблон проєктування - це типове, перевірене рішення для поширеної проблеми, яка виникає під час розробки програмного забезпечення. Він не є готовим кодом, а описує структуру та взаємодію класів і об’єктів, яку можна адаптувати під конкретну задачу.

2. Навіщо використовувати шаблони проєктування?

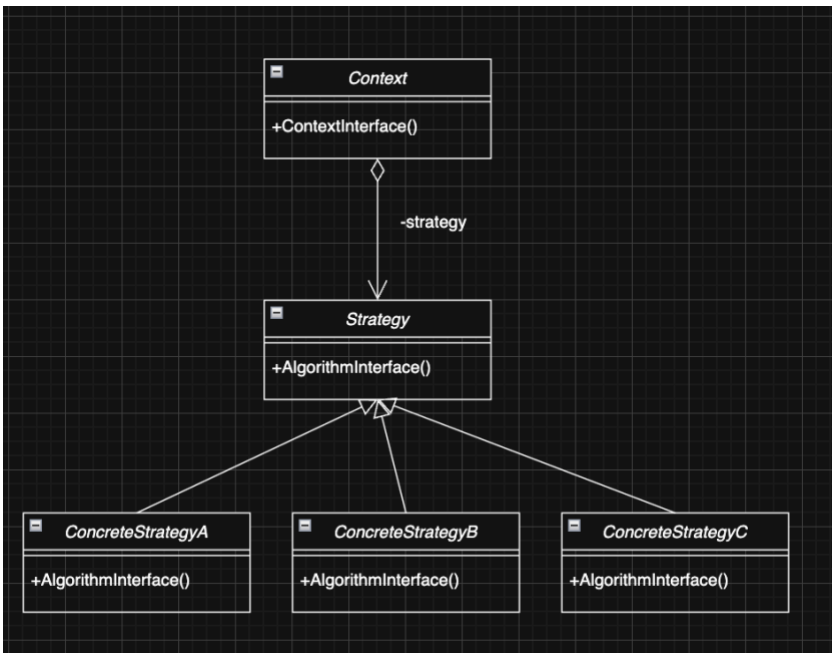
Шаблони:

- забезпечують повторне використання рішень і скорочують час розробки;
- покращують читабельність і підтримуваність коду;
- допомагають створювати гнучкі та розширювані системи;
- формують спільну мову між розробниками.

3. Яке призначення шаблону «Стратегія»?

Шаблон «Стратегія» (Strategy) дозволяє змінювати поведінку об’єкта під час виконання шляхом вибору однієї з кількох алгоритмічних стратегій, не змінюючи код самого об’єкта.

4. Нарисуйте структуру шаблону «Стратегія»:



- інтерфейс Strategy описує метод для виконання алгоритму;
- класи ConcreteStrategyA, ConcreteStrategyB, ConcreteStrategyC реалізують цей інтерфейс різними способами;
- клас Context зберігає посилання на стратегію й викликає її метод.

Таким чином, контекст делегує виконання конкретній стратегії

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

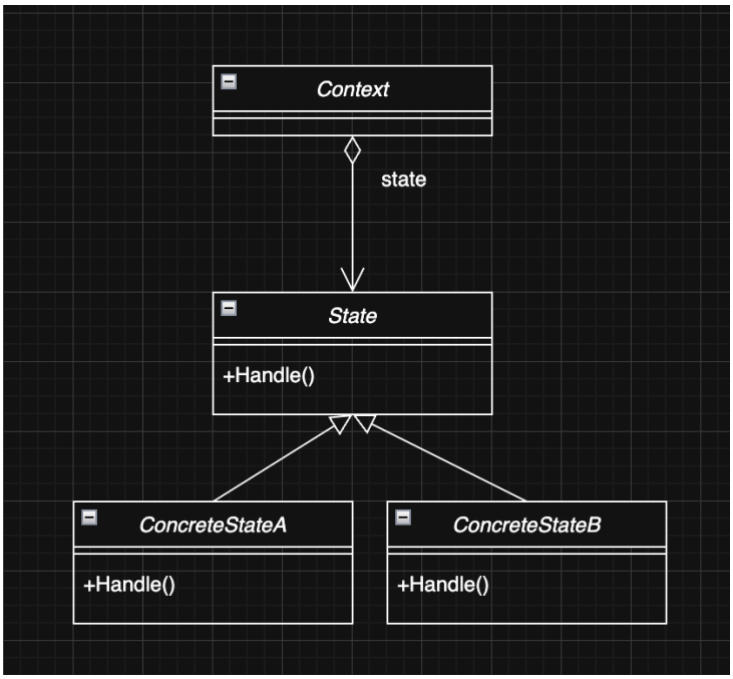
- IStrategy - інтерфейс або абстрактний клас, який визначає метод стратегії.
- ConcreteStrategyA / ConcreteStrategyB - конкретні реалізації алгоритмів.
- Context - клас, який містить посилання на стратегію та викликає її метод.

Взаємодія: Context делегує виконання завдання об’єкту-стратегії.

6. Яке призначення шаблону «Стан»?

Шаблон «Стан» (State) дозволяє змінювати поведінку об’єкта при зміні його внутрішнього стану, ніби об’єкт змінив свій клас. Кожен стан представлений окремим класом.

7. Нарисуйте структуру шаблону «Стан»:



- інтерфейс State описує поведінку для конкретного стану;
- класи ConcreteStateA, ConcreteStateB реалізують цей інтерфейс по-різному;

- клас Context зберігає посилання на поточний стан і делегує йому виконання

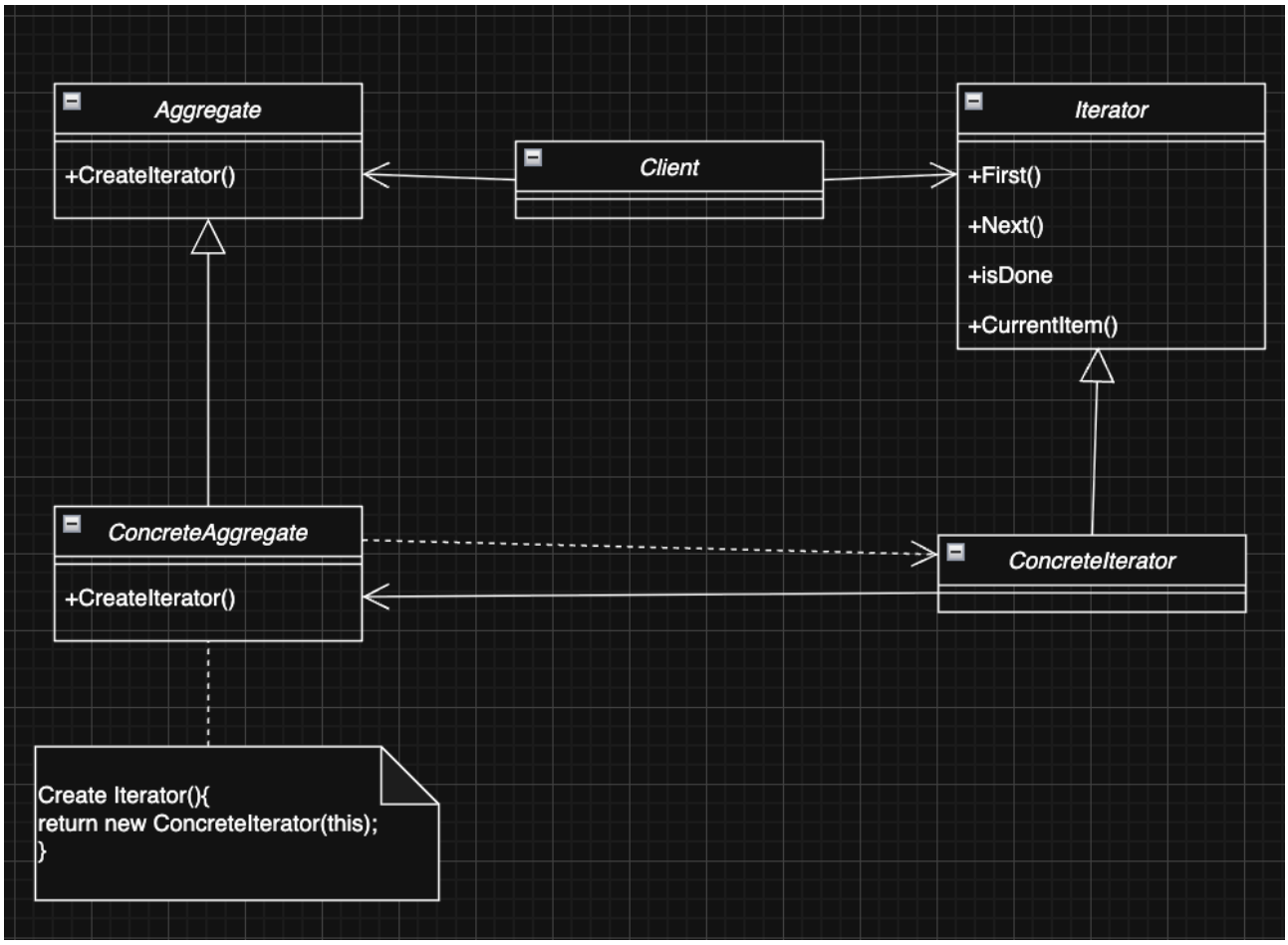
8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

- State - інтерфейс або абстрактний клас для всіх станів.
- ConcreteStateA / ConcreteStateB - конкретні стани, які реалізують різну поведінку.
- Context - об’єкт, що зберігає поточний стан і делегує йому виконання. Взаємодія: Context викликає метод поточного State; при потребі стан може змінитися на інший.

9. Яке призначення шаблону «Ітератор»?

Шаблон «Ітератор» (Iterator) надає уніфікований спосіб послідовного доступу до елементів колекції без розкриття її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор»:



- інтерфейс Iterator з методами First(), Next(), IsDone, CurrentItem;

- клас ConcreteIterator реалізує цей інтерфейс;
- інтерфейс Aggregate описує метод створення ітератора;
- клас ConcreteAggregate створює конкретний ітератор.

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

- Iterator - інтерфейс для обходу елементів.
- ConcreteIterator - конкретна реалізація ітератора.
- Aggregate - інтерфейс колекції, який створює ітератор.
- ConcreteAggregate - конкретна колекція.

Взаємодія: Ітератор отримує доступ до елементів колекції, рухаючись по ній послідовно.

12. В чому полягає ідея шаблону «Одинак»?

«Одинак» (Singleton) гарантує, що у програмі існує лише один екземпляр класу, і забезпечує глобальну точку доступу до нього.

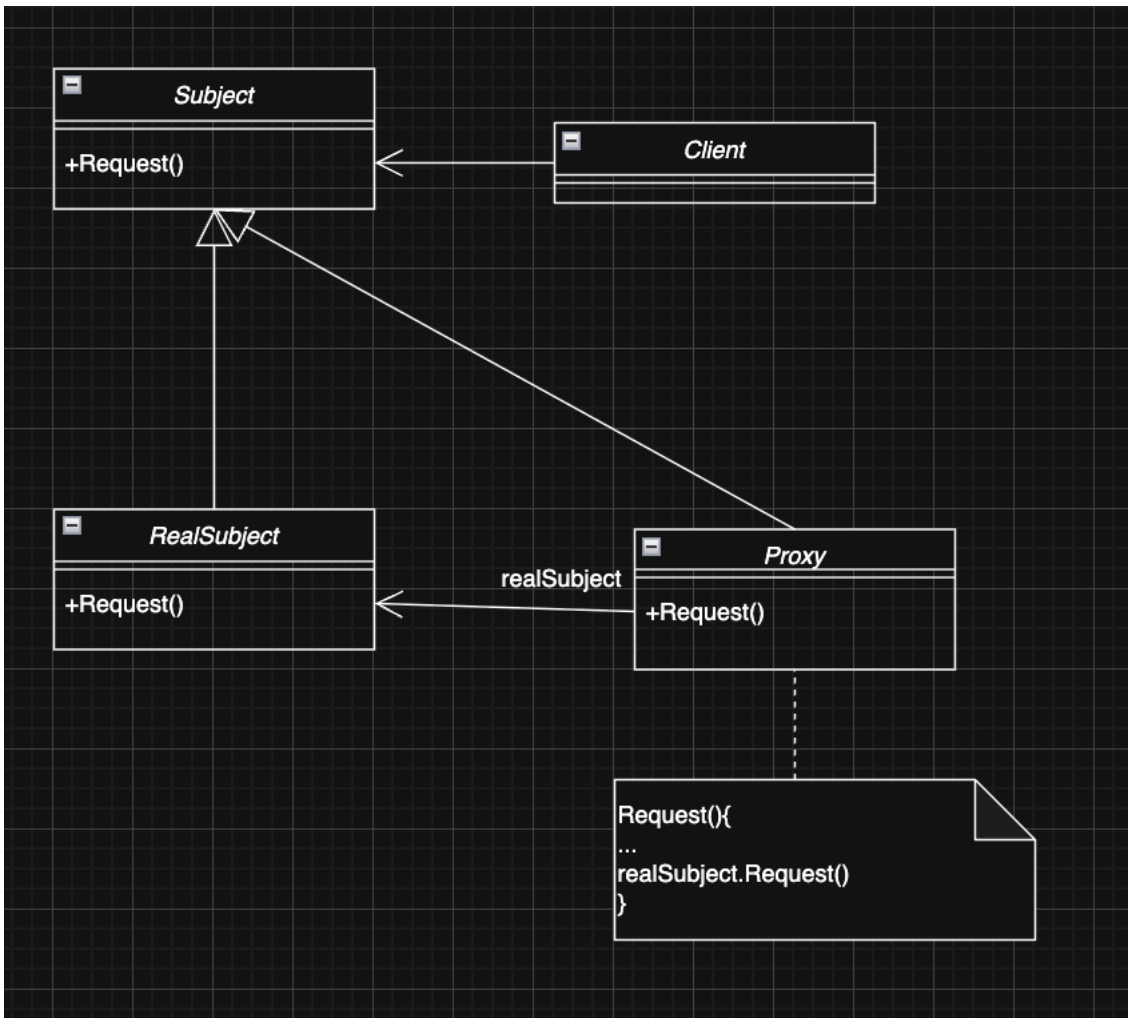
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

- створює глобальний стан, що ускладнює тестування та розширення;
- порушує принцип інверсії залежностей;
- може спричиняти проблеми в багатопотокових середовищах;
- з часом ускладнює підтримку та модульність коду.

14. Яке призначення шаблону «Проксі»?

Шаблон «Проксі» (Proxy) створює замісник або сурогатний об’єкт, який контролює доступ до реального об’єкта. Він дозволяє, наприклад, додати перевірку доступу, кешування чи відкладене створення.

15. Нарисуйте структуру шаблону «Проксі»:



- інтерфейс Subject описує загальні методи;
- клас RealSubject - реальний об’єкт, який виконує роботу;
- клас Proxy - містить посилання на RealSubject і викликає його методи, додаючи власну логіку;
- клієнт працює з інтерфейсом Subject, не знаючи, що перед ним проксі.

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

- Subject - спільний інтерфейс для реального об’єкта і проксі.
- RealSubject - реальний об’єкт, який виконує основну роботу.
- Proxy - замісник, який контролює доступ до RealSubject.
- Client - звертається до Proxy, не знаючи про існування RealSubject. Взаємодія: Client викликає метод Proxy, який у свою чергу виконує

додаткову логіку (наприклад, безпеку або кешування) і звертається до RealSubject за потреби.