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

Технології розроблення програмного забезпечення

«*Вступ до паттернів проектування. Паттерн «Iterator»*»

«10. VCS all-in-one»

Виконав
студент групи ІА–32:
Варивода К. С.

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

Зміст

Варіант

10. VCS all-in-one (iterator, adapter, factory method, facade, visitor, p2p)

Клієнт для всіх систем контролю версій повинен підтримувати основні команди і дії (commit, update, push, pull, fetch, list, log, patch, branch, merge, tag) для 3-х основних систем управління версіями (svn, git, mercurial), а також мати можливість вести реєстр репозиторіїв (і їх типів) і відображати дерева фіксації графічно.

Теоретичні відомості

Будь-який патерн проєктування, використовуваний при розробці інформаційних систем, являє собою формалізований опис, який часто зустрічається в завданнях проєктування, вдале рішення даної задачі, а також рекомендації по застосуванню цього рішення в різних ситуаціях. Крім того, патерн проєктування обов'язково має загальновживане найменування. Правильно сформульований патерн проєктування дозволяє, відшукавши одного разу вдале рішення, користуватися ним знову і знову.

Шаблон «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.

Проблема: Більшість колекцій виглядають як звичайний список елементів. Але є й екзотичні колекції, побудовані на основі дерев, графів та інших складних структур даних.

Але як би не була структурована колекція, користувач повинен мати можливість послідовно обходити її елементи, щоб виробляти з ними якісь дії.

Але яким способом слід переміщатися по складній структурі даних? Наприклад, сьогодні може бути достатнім обхід дерева в глибину, але завтра буде потрібно можливість переміщатися по дереву в ширину. А на наступному тижні і того гірше – знадобиться обхід колекції у випадковому порядку.

Додаючи все нові алгоритми в код колекції, ви потроху розмиваєте її основне завдання, яке полягає в ефективному зберіганні даних.

Рішення: 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.

До того ж, якщо вам знадобиться додати новий спосіб обходу, ви зможете створити окремий клас ітератора, не змінюючи існуючий код колекції.

Шаблонний ітератор містить:

- first() – установка покажчика перебору на перший елемент колекції;
- next() – установка покажчика перебору на наступний елемент колекції;
- hasNext() – метод, який віддає булевське поле, яке встановлюється як true коли наявний наступний елемент в списку
- CurrentItem() – поточний об'єкт колекції.

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.

Релевантність патерну для проекту.

Патерн «Ітератор» є чудовим доповненням для проекту, так як дозволяє сховати всю логіку отримання списку комітів, дає можливість отримувати ту кількість комітів, яку треба користувачу, без необхідності отримувати всі коміти одразу в список та дає можлиість отримувати дані напряму з комплексної структури проекту по одному, замість отримання всього одразу та складання в список, по якому буде проходитися вже вбудований ітератор.

Хід роботи

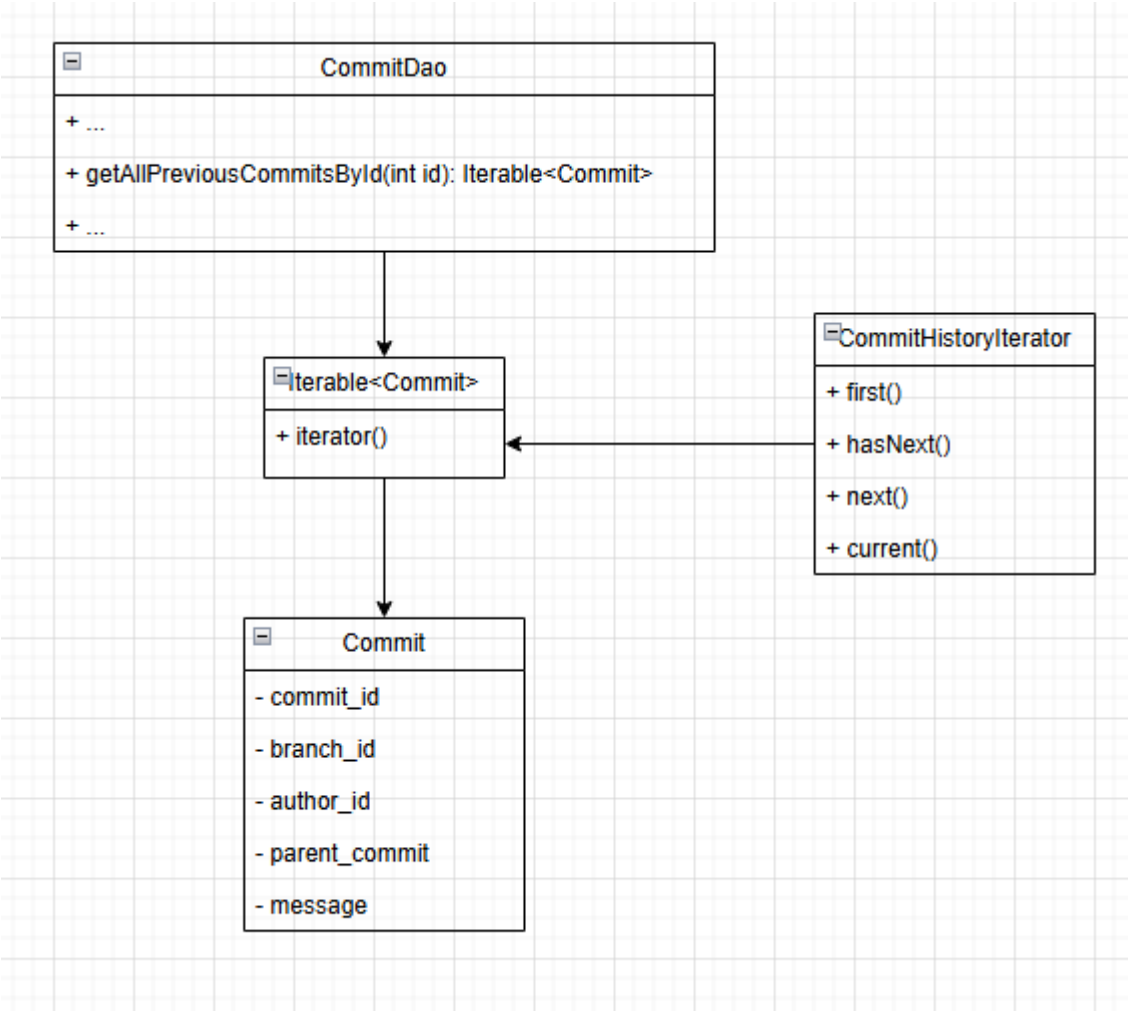


Рис 1 – Діаграма класів модулю з використанням паттерну «Ітератор»

```java
package vcs.repository.iterators;

import vcs.repository.classes.Commit;
import vcs.repository.dao.impl.CommitDao;

import java.util.Iterator;

public class CommitHistoryIterator implements Iterator<Commit> {  5 usages  mavr1iq

    private final CommitDao commitDao;  2 usages
    private final Commit firstCommit;  2 usages
    private int nextCommitId;  5 usages
    private Commit currentCommit;  3 usages

    public CommitHistoryIterator(int startId, CommitDao commitDao) {  1 usage  mavr1iq
        this.commitDao = commitDao;

        Commit commit = commitDao.getById(startId);
        this.firstCommit = commit;
        this.currentCommit = commit;

        if (commit != null) {
            this.nextCommitId = commit.getParent_commit();
        } else {
            this.nextCommitId = 0;
        }
    }

    @Override  mavr1iq
    public boolean hasNext() {
        return nextCommitId > 0;
    }

    @Override  mavr1iq
    public Commit next() {
        if (hasNext()) {
            Commit commit = commitDao.getById(nextCommitId);
            nextCommitId = commit.getParent_commit();
            this.currentCommit = commit;
            return commit;
        }
        return null;
    }

    public Commit current() {  1 usage  mavr1iq
        return currentCommit;
    }

    public Commit first() {  mavr1iq
        return firstCommit;
    }
}
```

Рис 2 – імплементація класу CommitHistoryIterator, який використовує патерн «Ітератор»

	commit_id	author_id	branch_id	parent_commit	message
1	1	1	1	<null>	Init commit
2	2	1	1	1	Test commit 1
3	3	1	1	2	Test commit 2
4	4	1	1	3	Test commit 3
5	5	1	1	4	Test commit 4

Рис 3 – Тестові дані в базі даних

```java
package vcs.repository;

import vcs.repository.classes.Commit;
import vcs.repository.dao.db.DatabaseContext;
import vcs.repository.dao.impl.CommitDao;
import vcs.repository.iterators.CommitHistoryIterator;


public class VcsAllInOne {  mavr1iq
    public static void main(String[] args) {  mavr1iq
        DatabaseContext dbContext = new DatabaseContext();
        CommitDao commitDao = new CommitDao(dbContext);

        Iterable<Commit> commits = commitDao.getAllPreviousCommitsById(5);
        CommitHistoryIterator iterator = (CommitHistoryIterator) commits.iterator();

        System.out.println(iterator.current());
        System.out.println(iterator.hasNext());
        System.out.println(iterator.first());

        for (Commit commit : commits) {
            System.out.println(commit);
        }
    }
}
```

Рис 4 – код для перевірки роботи патерну

```
Commit [commit_id=5, branch_id=1, message=Test commit 4, author_id=1, parent_commit=4]
true
Commit [commit_id=5, branch_id=1, message=Test commit 4, author_id=1, parent_commit=4]
Commit [commit_id=4, branch_id=1, message=Test commit 3, author_id=1, parent_commit=3]
Commit [commit_id=3, branch_id=1, message=Test commit 2, author_id=1, parent_commit=2]
Commit [commit_id=2, branch_id=1, message=Test commit 1, author_id=1, parent_commit=1]
Commit [commit_id=1, branch_id=1, message=Init commit, author_id=1, parent_commit=0]

Process finished with exit code 0
```

Рис 5 – вивід програми

Висновок

Отже, після виконання даної лабораторної роботи було вивчено паттерн програмування «Ітератор», його сильні та слабкі сторони, тонкощі імплементації та use кейси. Для закріплення отриманих знань було розроблено паттерн у існуючому проекті та продемонстровано його роботу

Відповіді на питання

1. Що таке шаблон проєктування?

Шаблон проєктування (Design Pattern) — це перевірене, багаторазово використовуване рішення типової проблеми, яка часто виникає в рамках проєктування програмного забезпечення. Це не готовий код, а скоріше опис або шаблон взаємодії об'єктів та класів для вирішення конкретного завдання, який можна адаптувати для своєї програми.

2. Навіщо використовувати шаблони проєктування?

Перевірені рішення: Вони пропонують рішення, які вже були перевірені часом та іншими розробниками, що допомагає уникнути "винаходу велосипеда".

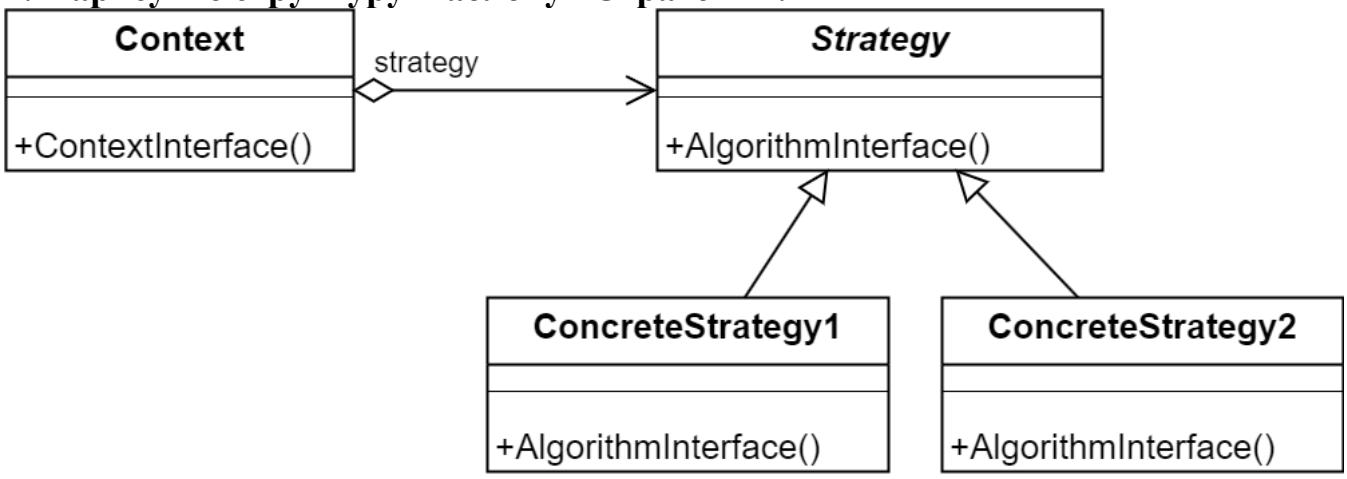
Спільна мова: Надають загальний словник для розробників. Легше сказати "тут використаємо Одинака", ніж пояснювати всю концепцію єдиного екземпляра.

Гнучкість та підтримка: Код, побудований на шаблонах, зазвичай є більш гнучким, читабельним та легким для підтримки та розширення.

3. Яке призначення шаблону «Стратегія»?

Призначення шаблону «Стратегія» (Strategy) — визначити сімейство алгоритмів, інкапсулювати кожен з них і зробити їх взаємозамінними. Це дозволяє обирати алгоритм під час виконання програми (at runtime) і змінювати поведінку об'єкта незалежно від клієнтського коду, який його використовує.

4. Нарисуйте структуру шаблону «Стратегія».



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

Context (Контекст): Клас, який потребує певної поведінки (алгоритму). Він зберігає посилання на об'єкт Strategy.

Strategy (Стратегія): Це інтерфейс (або абстрактний клас), що оголошує загальний метод, який мають реалізувати всі конкретні алгоритми.

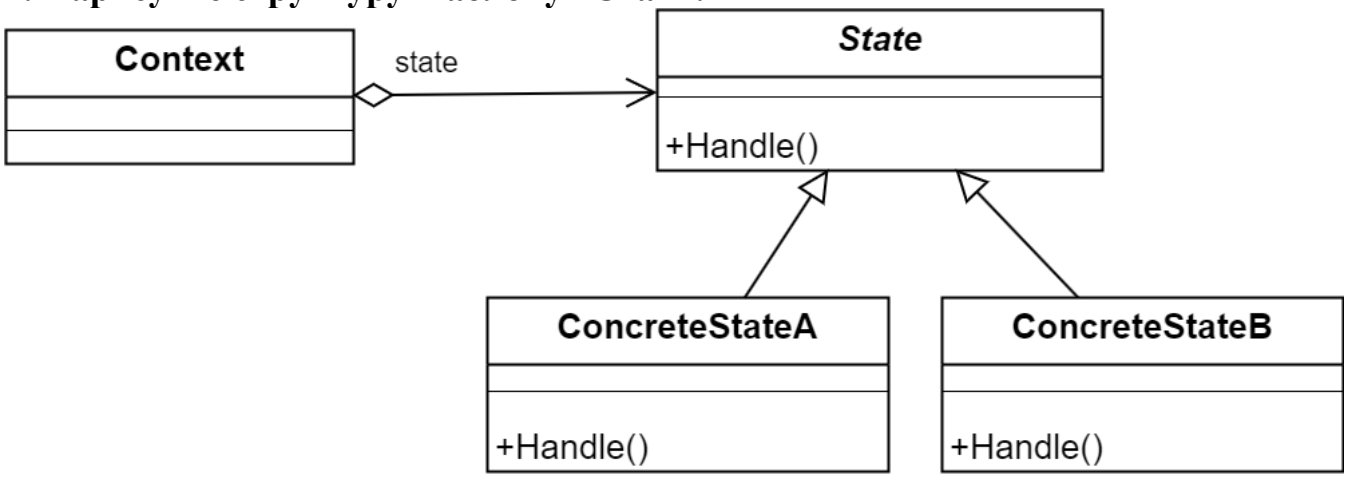
ConcreteStrategy (Конкретна Стратегія): Один або декілька класів, що реалізують інтерфейс Strategy. Кожен з них надає конкретну реалізацію алгоритму.

Взаємодія: Context не виконує логіку сам, а делегує її об'єкту Strategy, який у нього є. Клієнт може "підмінити" об'єкт Strategy всередині Context у будь-який час, тим самим змінюючи його поведінку.

6. Яке призначення шаблону «Стан»?

Призначення шаблону «Стан» (State) — дозволити об'єкту змінювати свою поведінку, коли змінюється його внутрішній стан. Ззовні це виглядає так, ніби об'єкт змінив свій клас. Шаблон інкапсулює поведінку, пов'язану з кожним станом, в окремий об'єкт.

7. Нарисуйте структуру шаблону «Стан».



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

Context (Контекст): Клас, який може мати різні стани. Він зберігає посилання на свій поточний об'єкт State.

State (Стан): Інтерфейс (або абстрактний клас), що визначає методи поведінки, спільні для всіх станів.

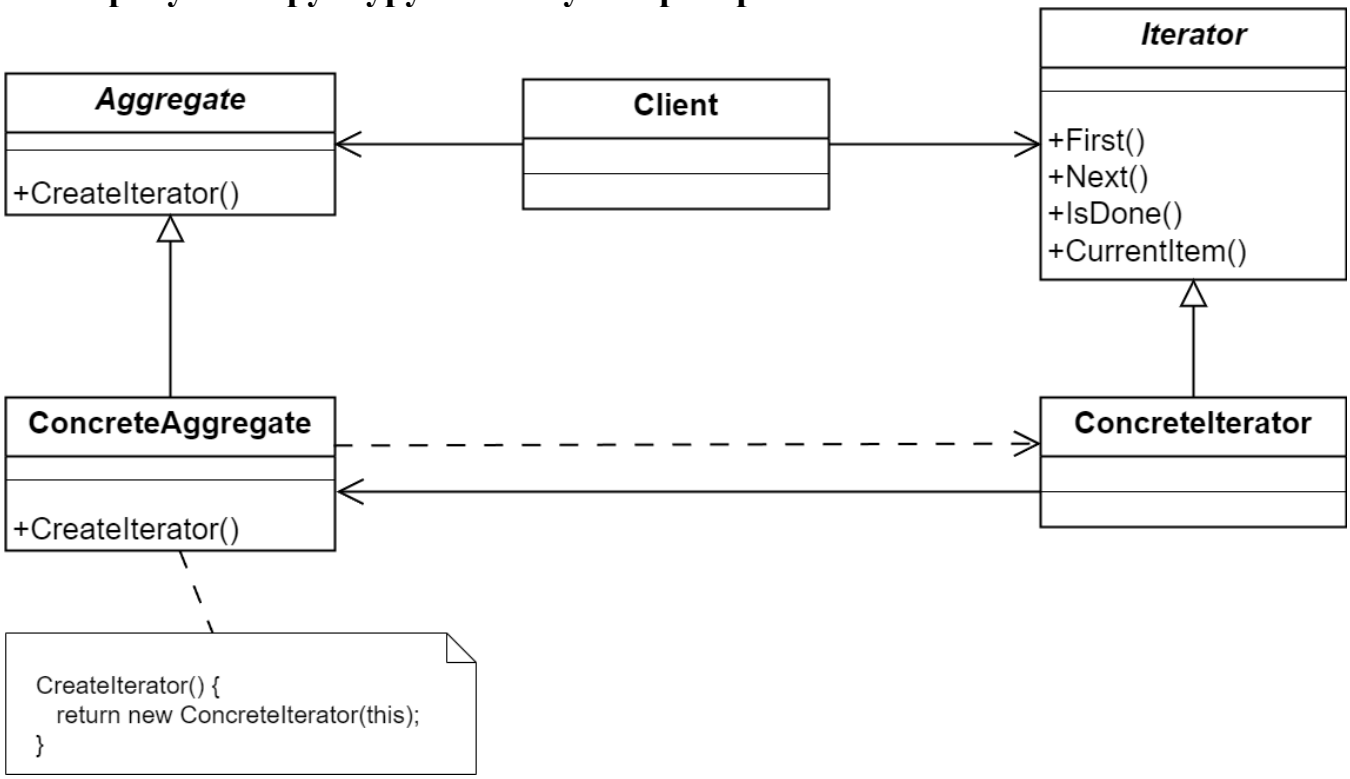
ConcreteState (Конкретний Стан): Один або декілька класів, що реалізують інтерфейс State. Кожен клас відповідає за поведінку об'єкта в одному конкретному стані.

Взаємодія: Context делегує всі запити своєму поточному об'єкту State. Коли відбувається дія, ConcreteState не лише виконує логіку, але й може змінити стан Context, передавши йому новий об'єкт ConcreteState.

9. Яке призначення шаблону «Ітератор»?

Призначення шаблону «Ітератор» (Iterator) — надати стандартний спосіб послідовного доступу до елементів колекції (наприклад, списку, масиву чи дерева), не розкриваючи її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор».



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

Iterator (Ітератор): Інтерфейс, що визначає методи для обходу колекції (наприклад, hasNext() та next()).

ConcreteIterator (Конкретний Ітератор): Клас, що реалізує інтерфейс Iterator. Він знає, як обходити конкретну колекцію і відстежує поточну позицію.

Aggregate (Агрегат/Колекція): Інтерфейс, що визначає метод для створення об'єкта Iterator (наприклад, createIterator()).

ConcreteAggregate (Конкретний Агрегат): Клас, що реалізує Aggregate. Це і є сама колекція (наприклад, ArrayList), яка повертає екземпляр ConcreteIterator.

Взаємодія: Клієнтський код просить у ConcreteAggregate (колекції) дати йому Iterator. Після цього клієнт працює тільки з інтерфейсом Iterator (hasNext(), next()), не знаючи нічого про внутрішню будову колекції.

12. В чому полягає ідея шаблона «Одинак»?
Ідея шаблону «Одинак» (Singleton) полягає в тому, щоб гарантувати, що певний клас матиме тільки один екземпляр у всій програмі, і надати глобальну точку доступу до цього екземпляра.

13. Чому шаблон «Одинак» вважають «анти-шаблоном»?
Його часто вважають анти-шаблоном з кількох причин:

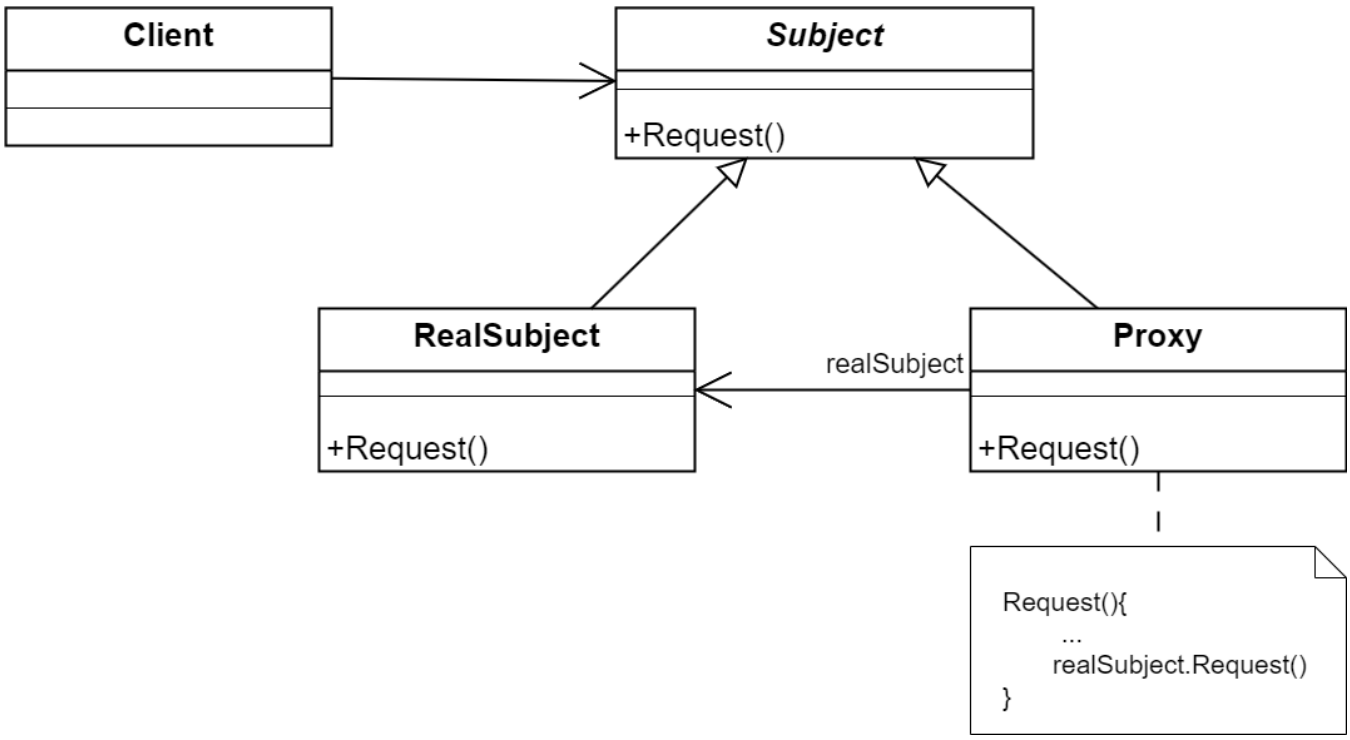
Глобальний стан: Він вводить глобальний стан у програму, що ускладнює відстеження залежностей і може призвести до неочікуваних побічних ефектів.

Проблеми з тестуванням: Унеможливлює ізольоване юніт-тестування. Важко "підмінити" (замокати) одинака, і тести, що змінюють його стан, можуть впливати на інші тести.

Порушення SRP (Принцип єдиної відповідальності): Клас починає відповідати за дві речі: за свою основну бізнес-логіку і за контроль над власним життєвим циклом (створенням).

14. Яке призначення шаблону «Проксі»?
Призначення шаблону «Проксі» (Proxy) — надати об'єкт-замісник (або "посередника") для іншого об'єкта, щоб контролювати доступ до нього. Це дозволяє додавати допоміжну логіку до або після звернення до реального об'єкта (наприклад, для логування, кешування, "лінивої" ініціалізації або перевірки прав доступу).

15. Нарисуйте структуру шаблону «Проксі».



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

Subject (Суб'єкт): Загальний інтерфейс, який реалізують і Proxy, і RealSubject. Це дозволяє клієнту працювати з Proxy так само, як і з реальним об'єктом.

RealSubject (Реальний Суб'єкт): "Справжній" об'єкт, який виконує основну роботу. Proxy приховує цей об'єкт від клієнта.

Proxy (Замісник): Клас, що реалізує інтерфейс Subject і зберігає посилання на RealSubject.

Взаємодія: Клієнт завжди звертається до Proxy. Proxy перехоплює виклик, виконує свою допоміжну логіку (наприклад, перевіряє кеш або права доступу), і лише після цього (якщо потрібно) делегує виклик об'єкту RealSubject.