Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 4

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Вступ до паттернів проектування»

«Web crawler»

Виконала
студентка групи – ІА-31
Горлач Дар’я Дмитрівна

Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Вступ до патернів проектування.

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

Тема проєкту: Web crawler (proxy, chain of responsibility, memento, template method, composite, p2p) Веб-сканер повинен вміти розпізнавати структуру сторінок сайту, переходити за посиланнями, збирати необхідну інформацію про зазначений термін, видаляти не семантичні одиниці (рекламу, об'єкти javascript і т.д.), зберігати знайдені дані у вигляді структурованого набору html файлів вести статистику відвіданих сайтів і метадані.

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.

Зміст

Теоретичні відомості

Поняття шаблону проєктування

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 [5]. Крім того, патерн проєктування обов’язково має загальновживане найменування. Правильно сформульований патерн проєктування дозволяє,

відшукавши одного разу вдале рішення, користуватися ним знову і знову. Варто підкреслити, що важливим початковим етапом при роботі з патернами є адекватне моделювання розглянутої предметної області. Це є необхідним як для отримання належним чином формалізованої постановки задачі, так і для вибору відповідних патернів проєктування.

Відповідне використання патернів проєктування дає розробнику ряд незаперечних переваг. Наведемо деякі з них. Модель системи, побудована в межах патернів проєктування, фактично є структурованим виокремленням тих елементів і зв’язків, які значимі при вирішенні поставленого завдання. Крім цього, модель, побудована з використанням патернів проєктування, більш проста і наочна у вивченні, ніж стандартна модель. Проте, не дивлячись на простоту і наочність, вона дозволяє глибоко і всебічно опрацювати архітектуру розроблюваної системи з використанням спеціальної мови.

Застосування патернів проєктування підвищує стійкість системи до зміни вимог та спрощує неминуче подальше доопрацювання системи. Крім того, важко переоцінити роль використання патернів при інтеграції інформаційних систем організації. Також слід зазначити, що сукупність патернів проєктування, по суті, являє собою єдиний словник проєктування, який, будучи уніфікованим засобом, незамінний для спілкування розробників один одним.

Таким чином шаблони представляють собою, підтверджені роками розробок в різних компаніях і на різних проєктах, «ескізи» архітектурних рішень, які зручно застосовувати у відповідних обставинах.

Шаблон «Proxy»

Призначення: «Proxy» (Проксі) – 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 [5].

Проксі об'єкти використовувалися в більш ранніх версіях інтернет-браузерів, наприклад, для відображення картинки: поки картинка завантажується, користувачеві відображається «заглушка» картинки.

Проблема: Ви супроводжуєте систему, одна із частин якої працює з зовнішнім сервісом підписання документів, нарпиклад, DocuSign. Періодично
клієнти в вашій системі формують звіти и форматі pdf, далі ваша система відправляє їх на підпис в сервіс DocuSign і потім періодично перевіряєте чи документ вже підписаний. Якщо документ підписаний, ви викачуєте його з сервісу і поміщаєте в своїй базі.

За останній рік кількість користувачів вашої системи виросло суттєво і почали приходити великі рахунки від DocuSign. Після аналізу ви розумієте, що рахунки зросли через велику кількість запитів з відправкою

документів на підписання та запитів на перевірку чи документ вже підписаний. Після обговорення з бізнес-аналітиками та користувачам зрозуміли, що критичний інтервал доставки документів на підписання – 2 години і такий самий час критичності перевірки що документ підписаний і можна було б групувати всі запити на відправку та на отримання і відправляти пакетом раз на годину. На даний момент клієнтський код працює з класом DocSignManager через інтерфейс IDocSignManager.

Рішення: Для вирішення проблеми можна застосувати патерн "Замісник". Ви реалізовуєте клас замісник, який також реалізовує інтерфейс IDocSignManager, але він накопичує запити на відправку файлів на підписання і відправляє їх раз на годину, також він приблизно раз на годину отримує підписані документи, а на запити від клієнтів відповідає на основі інформації взятої з бази. Таким чином старий клас DocSignManager так само використовується для роботи з DocuSign сервісом, але вже набагато рідже, а клієнтський код взаємодіє з додатковим проміжним рівнем DocSignManagerProxy, хоча з точки зору клієнського коду нічого не змінилося і він працює з тим самим об'єктом IDocSignManager. Реалізувавши такий підхід ви тепер економите 40% від попередньої вартості використання DocuSign сервіса і тепер основний вплив на вартість робить розмір файлів, що передаються на підпис, а не кількість запитів до служби.

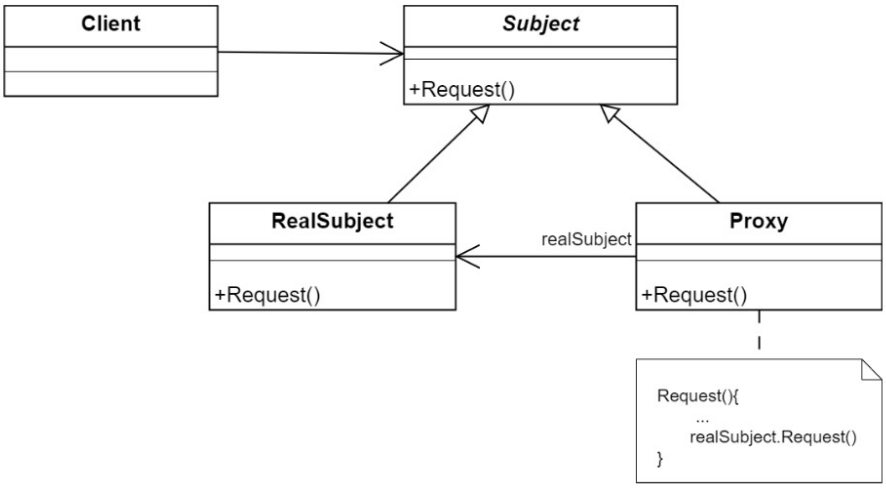


Рисунок 1. Структура патерну Proxy

Переваги та недоліки:

+ Легкість впровадження проміжного рівня без переробки клієнтського коду.
+ Додаткові можливості по керуванню життєвим циклом об’єкту.
- Існує ризик падіння швидкості роботи через впровадження додаткових операцій.
- Існує ризик неадекватної заміни відповіді клієнтському коду

Хід роботи

Реалізація патерну проєктування

Для реалізації Web Crawler використано патерн проєктування Proxy, оскільки він забезпечує контрольований доступ до завантажувача сторінок та додаткову функціональність кешування та пакувальної обробки запитів. Це дозволяє оптимізувати роботу веб-краулера шляхом зменшення кількості реальних HTTP-запитів та пакетної обробки URL.

Патерн Proxy реалізовано у класах PageFetcher, HttpPageFetcher та PageFetcherProxy, які відповідають за інтелектуальне управління процесом завантаження веб-сторінок. Такий підхід є особливо ефективним у роботі веб-краулера, де необхідно балансувати між продуктивністю та актуальністю даних.

```
public class HttpPageFetcher implements PageFetcher {

    3 usages
    private final Map<String, String> downloadedPages = new HashMap<>();

    3 usages
    @Override
    public void fetchPage(String url) {
        System.out.println("Завантаження сторінки " + url);
        downloadedPages.put(url, "<html>Content of " + url + "</html>");
    }

    3 usages
    @Override
    public boolean isPageFetched(String url) {
        System.out.println("Перевірка статусу сторінки " + url);
        return downloadedPages.containsKey(url);
    }

    2 usages
    @Override
    public String getPageContent(String url) {
        return downloadedPages.get(url);
    }
}
```

Рис. 2 – Код класу HttpPageFetcher

```java
public interface PageFetcher {
    3 usages  2 implementations
    void fetchPage(String url);

    3 usages  2 implementations
    boolean isPageFetched(String url);

    2 usages  2 implementations
    String getPageContent(String url);
}
```

Рис. 3 – Код інтерфейсу PageFetcher

```java
public class PageFetcherProxy implements PageFetcher {

    4 usages
    private final PageFetcher realFetcher;

    3 usages
    private final Queue<String> fetchQueue = new LinkedList<>();
    3 usages
    private final Map<String, String> localCache = new HashMap<>();

    1 usage
    public PageFetcherProxy(PageFetcher realFetcher) {
        this.realFetcher = realFetcher;
    }

    3 usages
    @Override
    public void fetchPage(String url) {
        System.out.println("Додаємо URL у чергу " + url);
        fetchQueue.add(url);
    }

    3 usages
    @Override
    public boolean isPageFetched(String url) {
        System.out.println("Перевірка з локального кешу");
        return localCache.containsKey(url);
    }

    2 usages
    @Override
    public String getPageContent(String url) {
        return localCache.get(url);
    }
```

Рис. 4 – Код класу PageFetcherProxy

Використання цього патерну надає:

1. Контрольований доступ до ресурсів: Proxy виступає як проміжний шар між клієнтом та реальним завантажувачем, що дозволяє додавати додаткову логіку (кешування, чергування, обмеження запитів) без змін у вихідному коді HttpPageFetcher.
2. Кешування результатів: Клас PageFetcherProxy містить локальний кеш (localCache), який зберігає вже завантажені сторінки. Це значно зменшує кількість мережевих запитів при повторних зверненнях до однакових URL.
3. Пакетна обробка запитів: Замість негайного виконання кожного запиту, Proxy збирає URL у чергу (fetchQueue) та обробляє їх пакувально за допомогою методу processBatch(). Це дозволяє оптимізувати використання мережевих ресурсів.
4. Спрощення коду клієнта: Клас WebCrawlerApp взаємодіє з Proxy так само, як і з реальним об'єктом, не потребуючи знання про додаткову логіку кешування чи чергування.

У нашому випадку патерн Proxy забезпечує ефективне управління мережевими ресурсами веб-краулера. Це дозволяє зменшити навантаження на цільові сервери, покращити швидкодію програми за рахунок кешування та уникнути блокування через надмірну кількість запитів.

Структура патерну

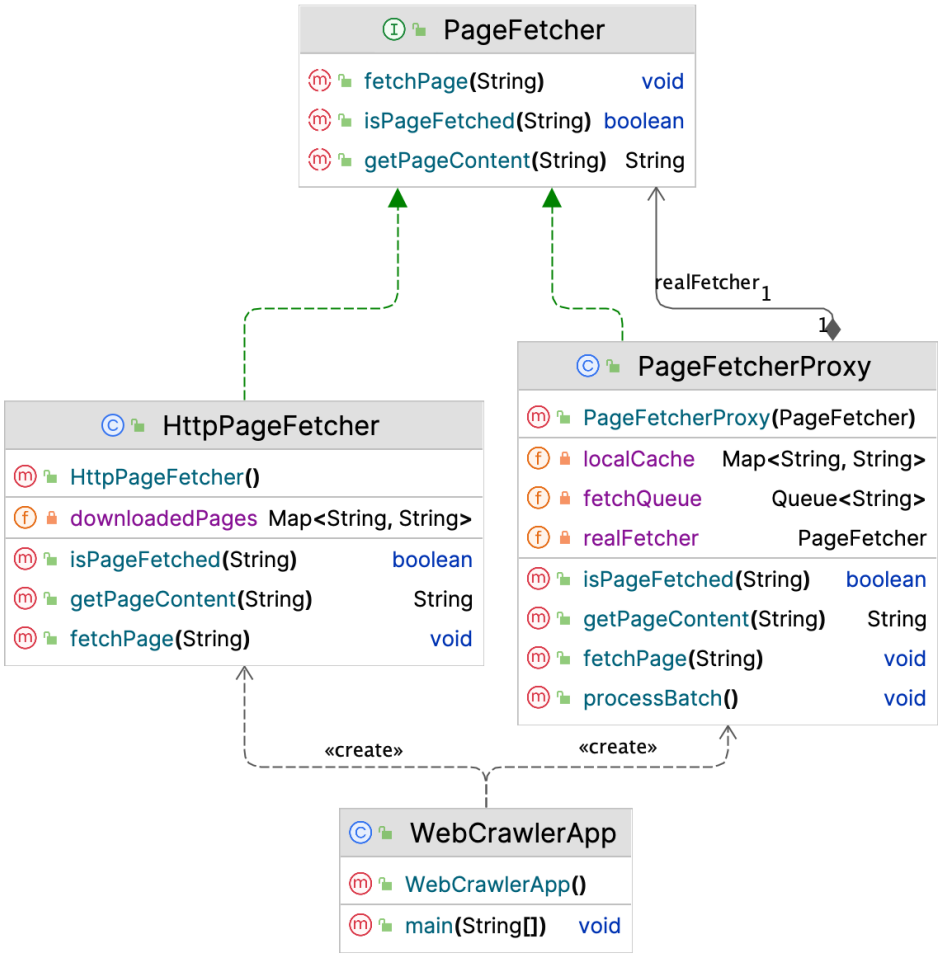


Рис. 5 – Структура патерну Proxy

Опис реалізації:

- PageFetcher - інтерфейс, що визначає контракт для завантаження сторінок

- HttpPageFetcher - реальний об'єкт, що виконує фактичне завантаження через HTTP

- PageFetcherProxy - проксі-об'єкт, що додає кешування та пакувальну обробку

- WebCrawlerApp - клієнтський код, який використовує проксі для доступу до веб-сторінок

Такий підхід особливо ефективний для веб-краулерів, де важливо обмежувати частоту запитів до одного домену, кешувати вже завантажений контент для повторного використання та додавати додаткову логіку без змін основного коду.

Посилання на репозиторій: https://github.com/dariahorlach/Lab-TRPZ

Висновки

Висновки: під час виконання лабораторної роботи, було реалізовано патерн Proxy для веб-краулера, що дозволило створити інтелектуальну систему завантаження веб-сторінок з оптимізованим використанням мережевих ресурсів. Клас PageFetcherProxy виступив як проміжний шар між клієнтом та реальним завантажувачем, додавши критично важливі функції кешування та пакувальної обробки запитів. Архітектура з Proxy дозволила розділити відповідальності, де HttpPageFetcher зосередився на базовій функції завантаження контенту, тоді як PageFetcherProxy взяв на себе управління ресурсами, кешуванням та оптимізацією запитів. Це підвищило модульність системи та спростило її тестування та розширення.

Питання до лабораторної роботи

1. Що таке шаблон проєктування?

Шаблон проєктування (Design Pattern) – це перевірене рішення типової задачі проєктування програмного забезпечення, що виникає у певному контексті.

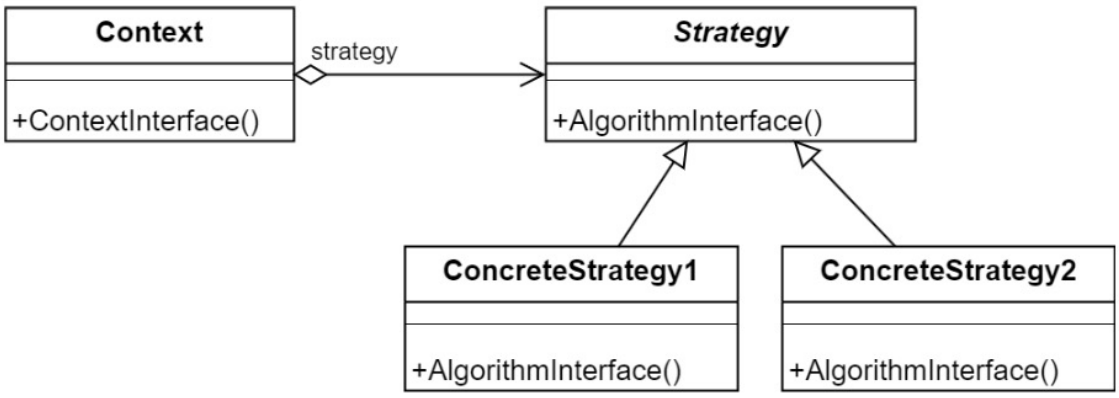
2. Навіщо використовувати шаблони проєктування?

1. Забезпечують повторне використання коду.
2. Полегшують підтримку та розширення системи.
3. Покращують зрозумілість і структуру коду.
4. Допомагають уникнути типових помилок.

3. Яке призначення шаблону «Стратегія»?

Шаблон «Стратегія» дозволяє змінювати алгоритм роботи об’єкта під час виконання програми, інкапсулюючи різні алгоритми в окремі класи.

4. Нарисуйте структуру шаблону «Стратегія».



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

Context – використовує об’єкт Strategy для виконання алгоритму.

Strategy – абстрактний клас або інтерфейс для алгоритмів.

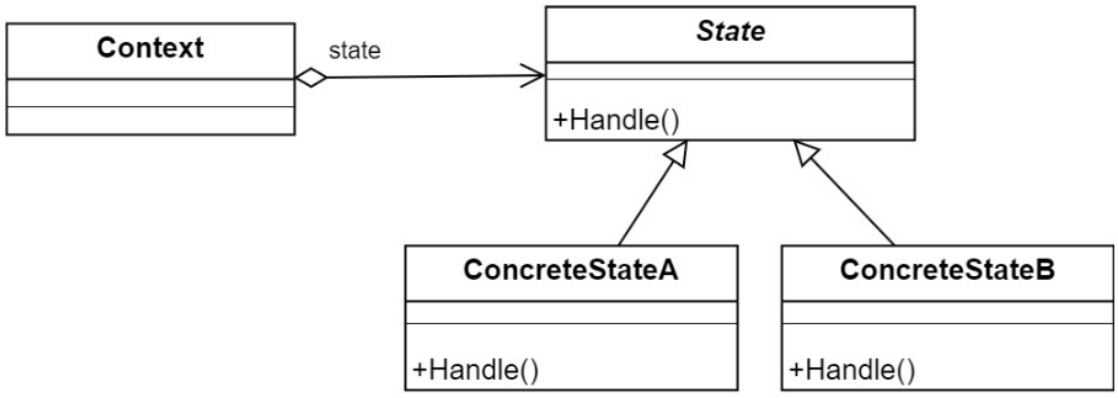
ConcreteStrategy – конкретна реалізація алгоритму.

Context делегує виконання алгоритму об’єкту Strategy, який можна змінювати динамічно.

6. Яке призначення шаблону «Стан»?

Шаблон «Стан» дозволяє змінювати поведінку об’єкта залежно від його внутрішнього стану, не змінюючи сам клас об’єкта.

7. Нарисуйте структуру шаблону «Стан».



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

Context – містить посилання на поточний стан та делегує йому поведінку.

State – абстрактний клас або інтерфейс стану.

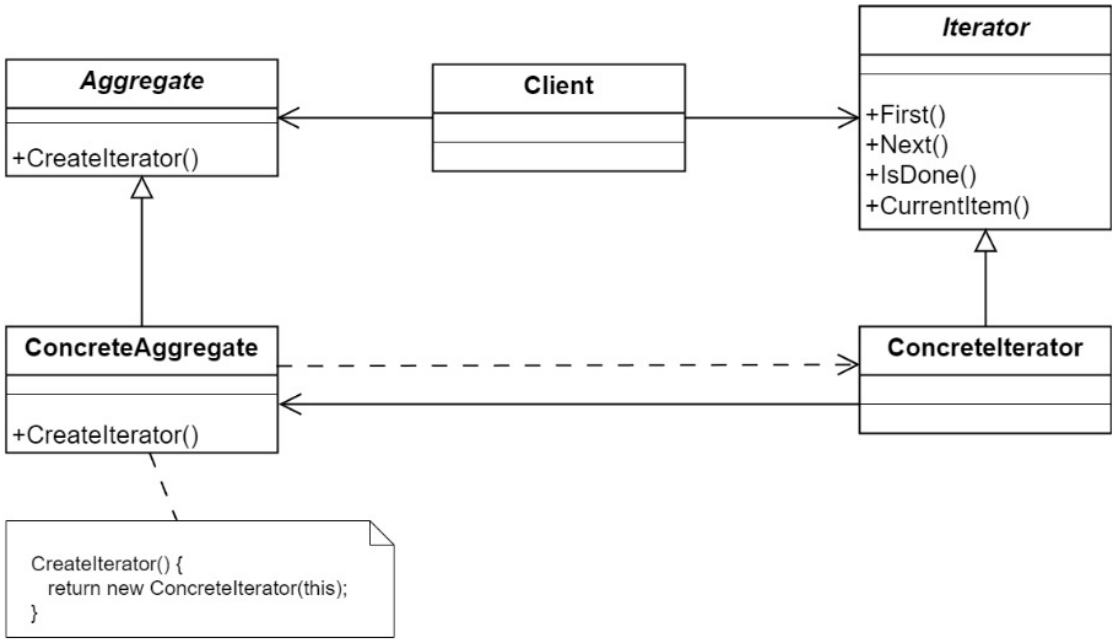
ConcreteState – реалізує конкретну поведінку для певного стану.

Context змінює стан об’єкта, а ConcreteState визначає поведінку в цьому стані.

9. Яке призначення шаблону «Ітератор»?

Шаблон «Ітератор» дозволяє послідовно отримувати доступ до елементів колекції без розкриття її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор».



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

ConcreteAggregate – реалізація колекції.

Iterator – інтерфейс ітератора.

ConcreteIterator – реалізує послідовний доступ до елементів ConcreteAggregate.

Клас ConcreteIterator використовує ConcreteAggregate для обходу елементів.

12. В чому полягає ідея шаблона «Одинак»?

Шаблон «Одинак» (Singleton) забезпечує, що клас матиме лише один екземпляр, і надає глобальну точку доступу до нього.

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

Створює глобальний стан, що ускладнює тестування.

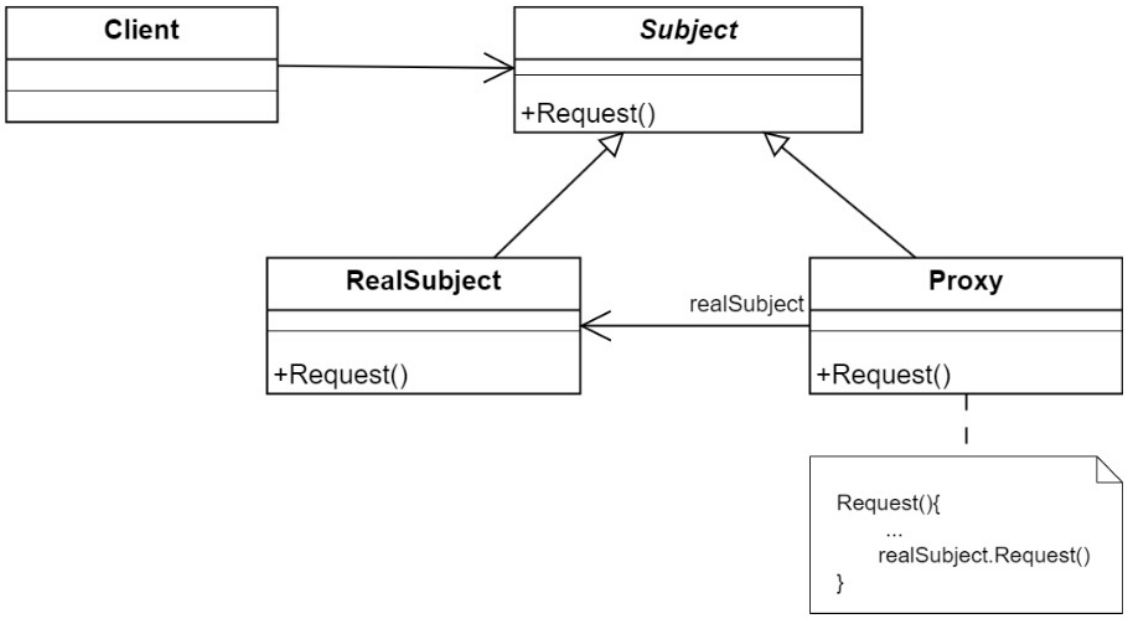
Порушує принцип інверсії залежностей.

Може призводити до жорстких залежностей між класами.

14. Яке призначення шаблону «Проксі»?

Шаблон «Проксі» забезпечує заміну об’єкта замінником, який контролює доступ до справжнього об’єкта.

15. Нарисуйте структуру шаблону «Проксі».



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

Subject – інтерфейс для RealSubject і Proxy.

RealSubject – реальний об’єкт, що виконує роботу.

Proxy – контролює доступ до RealSubject.
Proxy реалізує Subject і делегує виклики методів RealSubject, при необхідності додаючи додаткову поведінку.