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

Технологія розроблення програмного забезпечення

«ШАБЛОНИ «SINGLETON»,
«ITERATOR», «PROXY», «STATE»,
«STRATEGY» »

Варіант 11

Виконала
студентка групи ІА-24
Тильна Марія Сергіївна

Перевірив:
Мягкий М.Ю.

Київ 2024р.

Зміст

Тема: Шаблони «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»

Завдання.
1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми.

..11 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.

https://github.com/mashunchik/Webcrawler.git

Теоретичні відомості

Патерни проеткування

Патерн проектування — це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм.

На відміну від готових функцій чи бібліотек, патерн не можна просто взяти й скопіювати в програму. Патерн являє собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми.

Патерни часто плутають з алгоритмами, адже обидва поняття описують типові рішення відомих проблем. Але якщо алгоритм — це чіткий набір дій, то патерн — це високорівневий опис рішення, реалізація якого може відрізнятися у двох різних програмах.

Якщо провести аналогії, то алгоритм — це кулінарний рецепт з чіткими кроками, а патерн — інженерне креслення, на якому намальовано рішення без конкретних кроків його отримання.

Найбільш низькорівневі та прості патерни — *ідіоми*. Вони не дуже універсальні, позаяк мають сенс лише в рамках однієї мови програмування.

Найбільш універсальні — архітектурні патерни, які можна реалізувати практично будь-якою мовою. Вони потрібні для проектування всієї програми, а не окремих її елементів.

Крім цього, патерни відрізняються і за призначенням. Існують три основні групи патернів:

- **Породжуючі патерни** піклуються про гнучке створення об’єктів без внесення в програму зайвих залежностей.
- **Структурні патерни** показують різні способи побудови зв’язків між об’єктами.
- **Поведінкові патерни** піклуються про ефективну комунікацію між об’єктами.

Одинак (Singleton)

Шаблон проєктування "Одинак" гарантує, що клас матиме лише один екземпляр, і забезпечує глобальну точку доступу до цього екземпляра. Це корисно, коли потрібно контролювати доступ до деяких спільних ресурсів, наприклад, підключення до бази даних або конфігураційного файлу. Основна ідея полягає у тому, щоб закрити доступ до конструктора класу і створити статичний метод, що повертає єдиний екземпляр цього класу. У мовах, які підтримують багатопоточність, також важливо синхронізувати метод доступу, щоб уникнути створення кількох екземплярів в різних потоках. Один із способів реалізації одинак у Java – використання статичної ініціалізації, яка автоматично забезпечує безпечність у багатопоточному середовищі. Деякі розробники вважають одинак антипатерном, оскільки він створює глобальний стан програми, що ускладнює тестування та підтримку. Використання цього шаблону має бути обґрунтованим і обмеженим певними обставинами.

Ітератор (Iterator)

Шаблон "Ітератор" надає спосіб послідовного доступу до елементів колекції без розкриття її внутрішньої структури. Він особливо корисний для обходу різних типів колекцій, таких як списки, множини або деревовидні структури, незалежно від того, як вони реалізовані. Ітератор інкапсулює поточний стан перебору, тому може зберігати інформацію про те, який елемент є наступним. Цей шаблон дозволяє відокремити логіку роботи з колекцією від логіки обходу, що робить код більш гнучким і модульним. У Java цей шаблон реалізований у вигляді інтерфейсу Iterator, який надає методи hasNext() та next() для послідовного перебору елементів. Ітератор також можна використовувати для видалення елементів під час обходу колекції. Завдяки цьому шаблону можна використовувати поліморфізм для однакового доступу до елементів різних колекцій.

Проксі (Proxy)

Шаблон "Проксі" створює замісник або посередника для іншого об’єкта, що контролює доступ до цього об’єкта. Проксі може виконувати додаткову роботу перед передачею викликів реальному об'єкту, як-от перевірку прав доступу або відкладену ініціалізацію. Існує декілька типів проксі, серед яких захисний проксі, який контролює доступ, і віртуальний проксі, який затримує створення об'єкта, поки він не буде потрібен. У Java цей шаблон часто використовується для створення динамічних проксі за допомогою інтерфейсів, де проксі-клас реалізує той самий інтерфейс, що й реальний об’єкт. Проксі ефективний для оптимізації роботи з ресурсами або для контролю доступу до важких у створенні об'єктів. Це дозволяє зберігати оригінальний об'єкт захищеним і надає додатковий шар для маніпуляцій.

Стан (State)

Шаблон "Стан" дозволяє об'єкту змінювати свою поведінку при зміні внутрішнього стану, надаючи йому різні стани для різних контекстів. Він ефективно інкапсулює різні стани об'єкта як окремі класи і делегує дії поточному стану. Наприклад, кнопка може мати різні дії залежно від того, чи вона активована чи деактивована. Це дозволяє замість довгих умовних операторів використовувати поліморфізм, де кожен клас стану реалізує свою поведінку, визначену інтерфейсом стану. У Java цей шаблон може бути реалізований як клас з інтерфейсом для станів, де кожен стан є окремим підкласом, що відповідає за певну поведінку. Це полегшує масштабування та тестування, оскільки додавання нового стану не вимагає змін у вихідному коді.

Стратегія (Strategy)

Шаблон "Стратегія" дозволяє вибирати алгоритм або поведінку під час виконання, забезпечуючи взаємозамінність різних алгоритмів для конкретного завдання. Він передбачає інкапсуляцію різних варіантів поведінки в окремих класах, які реалізують один інтерфейс, що спрощує заміну і додавання алгоритмів. Клас контексту отримує об'єкт стратегії і викликає відповідні методи, не знаючи деталей реалізації конкретної стратегії. Наприклад, клас сортування може мати кілька стратегій: швидке сортування, сортування вставкою чи сортування вибором, і залежно від контексту обирається відповідний метод. У Java шаблон реалізується через інтерфейс стратегії, який мають різні класи конкретних стратегій.

Хід роботи

У цій лабораторній роботі я реалізувала шаблон Proxy. Шаблон Proxy у цьому проєкті забезпечує проміжний рівень між сканером і виконанням HTTP-запитів. Він додає кешування для зменшення повторних запитів, обмеження швидкості для сканування без перевантаження серверів, дотримується правил файлу robots.txt для етичного доступу та централізує логіку перевірок і обробки запитів. Це робить систему більш ефективною, безпечною та легкою для розширення.

Він обробляє всі складні веб-запити, зберігаючи основний код сканера чистим і зосередженим на своєму основному завданні – обробці веб-сторінок.

Як працює шаблон у Webcrawler:

WebCrawler -> HttpProxy -> BaseHttpClient -> Internet

- *WebCrawler* викликає *httpProxy.fetch(url)*
- *HttpProxy* перевіряє:
  1) Чи дозволений доступ до URL за правилами robots.txt?
  2) Чи знаходиться сторінка в кеші?
  3) Чи дотримуються обмеження швидкості запитів?
- Якщо всі перевірки успішні, запит передається до *BaseHttpClient*.
- *BaseHttpClient* виконує реальний HTTP-запит.
- Відповідь зберігається в кеші та повертається.

Структура проекту

```
3 usages  2 implementations
public interface IHttpClient {
    2 usages  2 implementations
    Document fetch(String url) throws IOException;
    1 usage  2 implementations
    boolean isAllowed(String url);
    2 implementations
    void clearCache();
}
```

Рис1. Інтерфейс

Інтерфейс IHttpClient служить угодою для операцій HTTP у веб-сканері. Він визначає основні операції, які має забезпечувати будь-яка реалізація HTTP-клієнта, чи то прямий клієнт, чи Proxy.

```java
public class RealHttpClient implements HttpClient {
    4 usages
    private Proxy proxy;
    2 usages
    private static final int TIMEOUT = 10000;

    2 usages
    @Override
    public Document fetchPage(String url) throws IOException {
        if (proxy != null) {
            return Jsoup.connect(url)
                    .proxy(proxy.getHost(), proxy.getPort())
                    .userAgent( s: "Mozilla/5.0")
                    .timeout(TIMEOUT)
                    .get();
        } else {
            return Jsoup.connect(url)
                    .userAgent( s: "Mozilla/5.0")
                    .timeout(TIMEOUT)
                    .get();
        }
    }

    2 usages
    @Override
    public void setProxy(Proxy proxy) { this.proxy = proxy; }

    @Override
    public boolean isAvailable() { return true; }
}
```

Рис2. Реальний об’єкт

Це справжній об’єкт, який виконує фактичну роботу (створення HTTP-запитів).

```java
public class ProxyHttpClient implements HttpClient {
    3 usages
    private final ProxyManager proxyManager;
    4 usages
    private final RealHttpClient realHttpClient;
    13 usages
    private Proxy currentProxy;
    3 usages
    private static final AtomicInteger requestCount = new AtomicInteger( initialValue: 0);
    1 usage
    private static final int REQUESTS_PER_PROXY = 50;

    1 usage
    public ProxyHttpClient(List<Proxy> initialProxies) {
        this.proxyManager = new ProxyManager(initialProxies);
        this.realHttpClient = new RealHttpClient();
    }
    private boolean shouldRotateProxy() {
        return currentProxy == null ||
               !currentProxy.isAvailable() ||
               requestCount.get() >= REQUESTS_PER_PROXY;
    }

    2 usages
    private void rotateProxy() {
        currentProxy = proxyManager.getNextProxy();
        realHttpClient.setProxy(currentProxy);
        requestCount.set(0);
        log.info("Rotated to new proxy: {}:{}", currentProxy.getHost(), currentProxy.getPort());
    }

    1 usage
    private void handleSuccess() {
        requestCount.incrementAndGet();
        if (currentProxy != null) {
            currentProxy.markSuccess();
            currentProxy.updateLastUsed();
        }
    }

    1 usage
    private void handleFailure() {
        if (currentProxy != null) {
            currentProxy.markFailed();
            if (!currentProxy.isAvailable()) {
                rotateProxy();
            }
        }
```

Рис.3 Реалізація Proxy

Це проксі, який контролює доступ до реального об'єкта.

Висновок : виконуючи цю лабораторну роботу, я ознайомилась з такими патернами, як Singleton, Iterator, Proxy, State та Strategy.Для реалізації свого проекту я використала шаблон Proxy, детально описавши його основну логіку та функціональність у контексті мого проєкту. Я зрозуміла, як ці патерни можуть підвищити модульність, зручність використання та читабельність коду, а також як вони сприяють ефективному управлінню об'єктами в програмуванні.