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 №7

Технології розробки програмного забезпечення

ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Виконала
студентка групи ІА-24
Тильна Марія Сергіївна

Перевірив:
Мягкий М.Ю.

Київ 2024

Зміст

Тема: ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Короткі теоретичні відомості

Шаблон Mediator (Посередник) є поведінковим шаблоном, який зменшує взаємодії між об'єктами, спрямовуючи всі комунікації через єдиний центральний об'єкт-посередник. Це дозволяє знизити зв'язність між компонентами та спростити їх взаємодію, замінюючи прямі зв'язки між ними на посередницькі. Такий підхід часто використовується в ситуаціях, коли необхідно знизити складність системи з багатьма взаємодіючими елементами.

Facade (Фасад) — це структурний шаблон, який забезпечує спрощений інтерфейс для роботи зі складною підсистемою. Завдяки фасаду клієнт може взаємодіяти з підсистемою через один простий інтерфейс, при цьому внутрішня складність системи приховується. Цей шаблон дозволяє знизити складність використання системи, коли клієнт не потребує доступу до всіх її компонентів.

Шаблон Bridge (Міст) дозволяє розділити абстракцію від її реалізації, що дає можливість змінювати ці дві частини незалежно одна від одної. Це дозволяє створювати гнучкі та масштабовані системи, де нові абстракції та їх реалізації можна додавати без порушення існуючого коду. Це особливо корисно, коли необхідно працювати з різними варіантами реалізації абстракцій, наприклад, для різних платформ чи пристроїв.

Template Method (Шаблонний метод) — це поведінковий шаблон, який визначає загальну структуру алгоритму, дозволяючи підкласам змінювати деякі його частини. В основному класі задається загальна послідовність кроків алгоритму, а підкласи можуть реалізовувати конкретні етапи. Цей шаблон допомагає знизити дублювання коду та забезпечує однакову структуру виконання алгоритмів, що є корисним у фреймворках і бібліотеках.

Хід роботи

..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.

Реалізація шаблону проєктування для майбутньої системи

В цій лабораторній роботі я реалізувала шаблон **Template Method**. Він дозволяє централізувати контроль за алгоритмом обробки сторінок, забезпечуючи можливість налаштувати специфічні етапи без ризику порушення загальної структури. Це зручно для підтримки та розширення проекту, оскільки додає гнучкість у реалізації нових сценаріїв обробки сторінок.

Шаблон Template Method дозволяє чітко визначити загальний алгоритм обробки веб-сторінки в методі handlePage(). Оскільки мій веб-сканер має кілька етапів обробки сторінки (завантаження, перевірка, збереження, обробка термінів, оновлення статистики, збір нових URL), Template Method дає змогу задати цю структуру в одному місці.

Переваги використання Template Method у проекті

Модульність: Алгоритм обробки сторінки розбитий на чіткі етапи, кожен із яких можна тестувати та модифікувати окремо.

Повторне використання коду: Загальний алгоритм зберігається в базовому класі, уникаючи дублювання логіки між різними реалізаціями.

Гнучкість: Легко додавати нові типи обробників сторінок, реалізуючи дочірні класи з іншими способами збереження чи обробки.

Простота підтримки: Чітка структура алгоритму та розділення обов'язків робить код легшим для розуміння і внесення змін.

```java
public abstract class AbstractPageHandler {
    2 usages
    protected final PageFetcher pageFetcher;
    2 usages
    protected final PageProcessingChain pageProcessingChain;
    1 usage
    public AbstractPageHandler(PageFetcher pageFetcher, PageProcessingChain pageProcessingChain) {
        this.pageFetcher = pageFetcher;
        this.pageProcessingChain = pageProcessingChain;
    }
    1 usage
    public final void handlePage(String url, Session session) {
        try {
            Document doc = fetchPage(url);
            if (doc != null) {
                processPageContent(doc);
                if (isRelevantPage(doc)) {
                    Page page = savePage(url, doc, session);
                    processTerms(doc, page);
                    updateStatistics(page);
                    collectNewUrls(doc);
                }
            }
        } catch (Exception e) {
            handleError(e);
        }
    }
```

Рис 1. Код абстрактного класу AbstarctPageHandler

```java
protected Document fetchPage(String url) throws Exception {
    return pageFetcher.fetchPage(url);
}
1 usage
protected void processPageContent(Document doc) throws Exception {
    pageProcessingChain.process(doc);
}
1 usage  1 implementation
protected abstract boolean isRelevantPage(Document doc);
1 usage  1 implementation
protected abstract Page savePage(String url, Document doc, Session session);
1 usage  1 implementation
protected abstract void processTerms(Document doc, Page page);
1 usage  1 implementation
protected abstract void updateStatistics(Page page);
1 usage  1 implementation
protected abstract void collectNewUrls(Document doc);
1 usage  1 implementation
protected abstract void handleError(Exception e);
```

Рис 1.1 Продовження коду абстрактного класу AbstarctPageHandler

Абстрактний клас, що реалізує шаблонний метод для обробки веб-сторінок. Визначає загальний алгоритм обробки сторінки, дозволяючи підкласам перевизначати конкретні кроки цього алгоритму.

```java
public class DefaultPageHandler extends AbstractPageHandler {
    2 usages
    private static final int MAX_METADATA_LENGTH = 255;
    1 usage
    private static final List<String> keywords = Arrays.asList("technology", "data", "machine", "science");
    6 usages
    private final Set<String> visitedUrls = new HashSet<>();
    5 usages
    private final Queue<String> urlQueue = new LinkedList<>();
    2 usages
    private final PageRepository pageRepository;
    2 usages
    private final TermRepository termRepository;
    3 usages
    private final StatisticRepository statisticRepository;
    4 usages
    private final ContentFilter contentFilter;
    1 usage
    public DefaultPageHandler(
            PageFetcher pageFetcher,
            PageProcessingChain pageProcessingChain,
            PageRepository pageRepository,
            TermRepository termRepository,
            StatisticRepository statisticRepository,
            ContentFilter contentFilter) {
        super(pageFetcher, pageProcessingChain);  // Викликаємо конструктор батьківського класу
        this.pageRepository = pageRepository;
        this.termRepository = termRepository;
        this.statisticRepository = statisticRepository;
        this.contentFilter = contentFilter;
```

Low disk space
Less than 50 MiB is left on the system

Рис 2. Код класу DefaultPageHandler

```java
    @Override
    protected boolean isRelevantPage(Document doc) { return contentFilter.isSemanticContent(doc); }
    1 usage
    @Override
    protected Page savePage(String url, Document doc, Session session) {
        String truncatedMetadata = truncateMetadata(doc.title());

        Page page = new Page();
        page.setUrl(url);
        page.setSemantic(true);
        page.setMetadata(truncatedMetadata);
        page.setSession(session);

        return pageRepository.save(page);
    }
    1 usage
    @Override
    protected void processTerms(Document doc, Page page) {
        String pageText = contentFilter.extractText(doc);
        List<String> foundTerms = contentFilter.extractTerms(doc, keywords);

        for (String term : foundTerms) {
            Term termEntity = new Term();
            termEntity.setSession(page.getSession());
            termEntity.setTerm(term);
            termEntity.setPage(page);
            termEntity.setOccurrences(Collections.frequency(Arrays.asList(pageText.split( regex: " ")), term));
            termEntity.setSemantic(true);
            termRepository.save(termEntity);
```

Рис 2.1 Продовження коду класу DefaultPageHandler

```java
@Override
protected void updateStatistics(Page page) {
    com.example.webcrawler_back.models.Statistic statistic =
        statisticRepository.findTopByOrderByCreatedAtDesc();
    if (statistic == null) {
        statistic = new com.example.webcrawler_back.models.Statistic();
    }

    statistic.setTotalScans(statistic.getTotalScans() + 1);
    statistic.setPagesVisited(statistic.getPagesVisited() + 1);
    if (page.getSemantic()) {
        statistic.setSemanticPages(statistic.getSemanticPages() + 1);
    }

    statisticRepository.save(statistic);
}

1 usage
@Override
protected void collectNewUrls(Document doc) {
    for (Element link : doc.select( cssQuery: "a[href]")) {
        String newUrl = link.absUrl( attributeKey: "href");
        if (newUrl != null && !newUrl.isEmpty() && !visitedUrls.contains(newUrl)) {
            urlQueue.add(newUrl);
        }
    }
}
```

Рис 2.2 Продовження коду класу DefaultPageHandler

```java
@Override
protected void handleError(Exception e) {
    System.err.println("Error during page processing: " + e.getMessage());
    e.printStackTrace();
}
1 usage
private String truncateMetadata(String metadata) {
    if (metadata.length() > MAX_METADATA_LENGTH) {
        return metadata.substring(0, MAX_METADATA_LENGTH);
    }
    return metadata;
}
2 usages
public Queue<String> getUrlQueue() { return urlQueue; }
1 usage
public Set<String> getVisitedUrls() { return visitedUrls; }
3 usages
public void setUrlQueue(Queue<String> queue) {
    this.urlQueue.clear();
    this.urlQueue.addAll(queue);
}
1 usage
public void setVisitedUrls(Set<String> urls) {
    this.visitedUrls.clear();
    this.visitedUrls.addAll(urls);
}
1 usage
public void addVisitedUrl(String url) {
    this.visitedUrls.add(url);
}
```

Рис 2.3 Продовження коду класу DefaultPageHandler

Конкретна реалізація обробника сторінок. Реалізує всі абстрактні методи з AbstractPageHandler. Клас DefaultPageHandler розширює клас AbstractPageHandler і надає конкретну реалізацію для кожного з абстрактних методів базового класу

```
private void crawl() {
    try {
        while (running && !urlQueue.isEmpty()) {
            String currentUrl = urlQueue.poll();
            pageHandler.setUrlQueue(urlQueue);  // Синхронізуємо чергу URL

            if (visitedUrls.contains(currentUrl)) {
                continue;
            }

            try {
                // Використовуємо шаблонний метод для обробки сторінки
                pageHandler.handlePage(currentUrl, currentSession);
                visitedUrls.add(currentUrl);
                pageHandler.addVisitedUrl(currentUrl);  // Додаємо URL до відвіданих в обробнику

                // Оновлюємо чергу URL з обробника
                urlQueue = new LinkedList<>(pageHandler.getUrlQueue());

                // Зберігаємо проміжний стан кожні 10 URL
                if (visitedUrls.size() % 10 == 0) {
                    CrawlerState state = new CrawlerState(
                        new HashSet<>(visitedUrls),
                        new LinkedList<>(urlQueue),
                        currentSession.getId().toString()
                    );
```

Рис 3. Приклад використання шаблону

Коли викликається метод handlePage() з конкретною сторінкою (URL), алгоритм обробки сторінки починається. Всі етапи, визначені в шаблонному методі, будуть виконуватися у фіксованій послідовності, але з конкретними реалізаціями для кожного етапу, які надаються в DefaultPageHandler.

Метод handlePage() є шаблонним методом, тому що визначає загальну структуру процесу, залишаючи певні кроки на реалізацію в підкласах. Це дозволяє кожному підкласу реалізувати конкретні деталі для кожного етапу обробки сторінки (збереження, перевірка релевантності, обробка термінів тощо).

Зображення структури шаблону

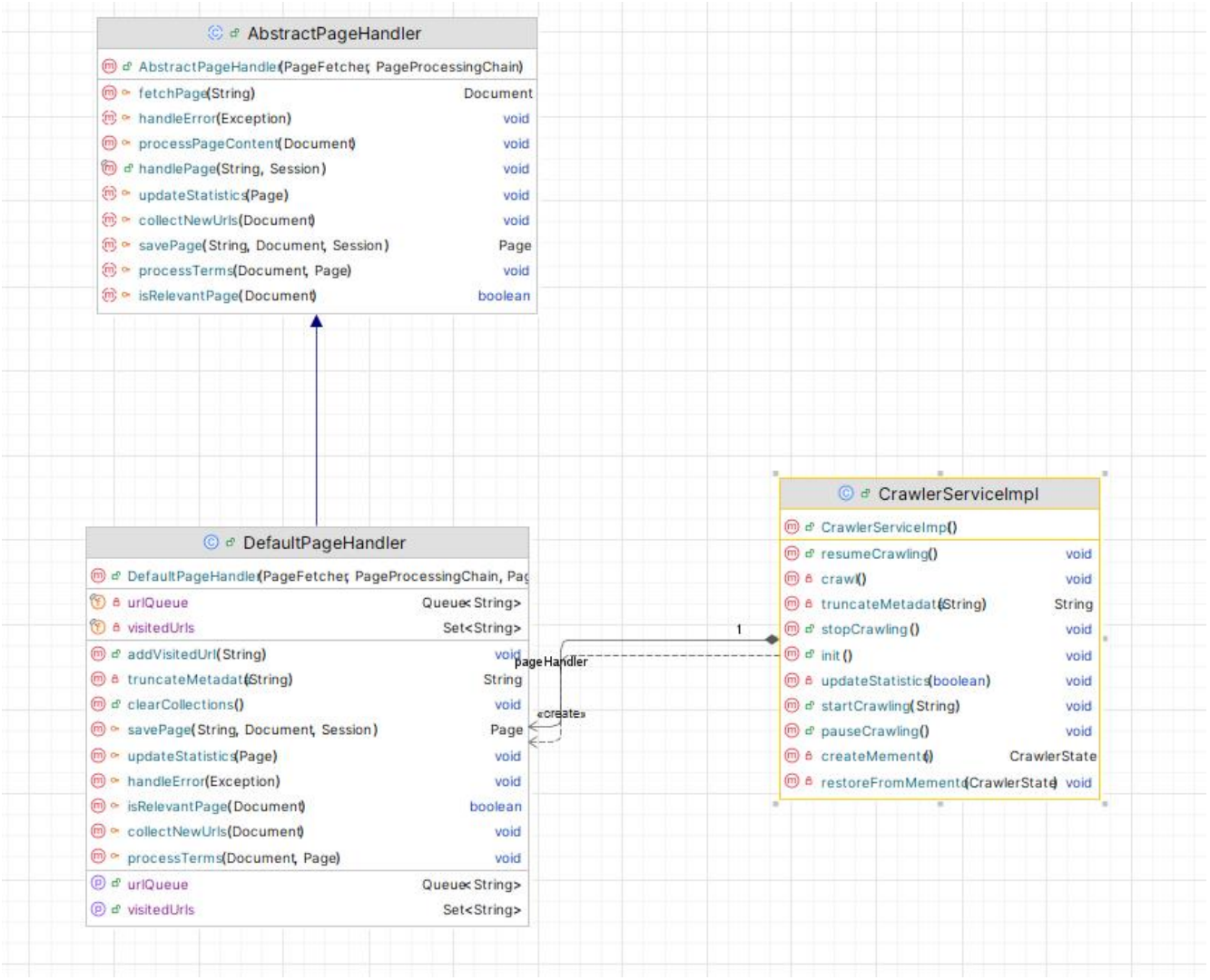

Рис 4. Структура шаблону

Посилання на репозиторій:

https://github.com/mashunchikWebcrawler

Висновок: На цій лабораторній роботі я ознайомилась з такими шаблонам проектування як «MEDIATOR»,«FACADE»,«BRIDGE»,«TEMPLATE METHOD». У своєму проекті я реалізувала шаблон TEMPLATE METHOD. Він дозволяє визначити чітку структуру для обробки веб-сторінок, при цьому даючи можливість гнучко реалізовувати специфічні кроки для кожного етапу. Це покращує підтримуваність коду, дозволяючи в майбутньому безперешкодно змінювати або доповнювати функціональність без порушення загального алгоритму. Це також спрощує підтримку, тестування та розширення функціональності проекту, забезпечуючи ефективне управління процесом сканування та обробки даних.