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 розроблення програмного забезпечення

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

Виконала
студентка групи ІА–22:
Фоменко Альона

Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

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

Мета: Ознайомитися з короткими теоретичними відомостями. Реалізувати частину функціоналу робочої програми у вигляді класів та їх взаємодій для досягнення конкретних функціональних можливостей. Застосування одного з розглянутих шаблонів при реалізації програми

Теоретичні відомості

Принципи проектування:

1. Don't Repeat Yourself (DRY):
    - Уникайте повторень коду на всіх рівнях.
    - Переваги:
        - Код стає компактнішим і легшим для читання.
        - Просте виправлення помилок без ризику пропустити їх в іншому місці.
        - Зменшення дублювання функціональності та помилок.
    - Виправлення повторень:
        - Використовуйте техніки рефакторингу, як винесення методів, інтерфейсів або класів.
2. Keep it simple, Stupid! (KISS):
    - Простота — ключ до надійності та читабельності коду.
    - Маленькі прості компоненти працюють краще, ніж складна система.
    - Приклад:
        - Спільнота Unix дотримується принципу: "Do one thing right".
3. You only load it once! (YOLO):
    - Ініціалізаційні та конфігураційні змінні слід завантажувати один раз.
    - Мета: уникнення зниження продуктивності через повторне завантаження.
4. Принцип Парето (80/20):
    - Більшість результатів досягається мінімумом зусиль:
        - 80% навантаження створює 20% додатка.
        - 80% багів можна усунути, виправивши 20% помилок.
    - Висновок: для вирішення більшості проблем потрібна мала частина ресурсів.
5. You ain’t gonna need it (YAGNI):

- Уникайте створення надмірно універсальних рішень.
- Переваги:
    - Простота реалізації.
    - Мінімізація зайвої гнучкості, що захаращує код.

Шаблон «Mediator» (Посередник):

1. Призначення:
    - Замість прямої взаємодії між об'єктами вони спілкуються через посередника.
    - Логіка взаємодії виноситься в окремий об'єкт.
2. Структура:
    - Посередник зберігає інформацію про взаємодію компонентів, а не про конкретні дії.
3. Проблема:
    - Наприклад, у діалозі створення профілю взаємодія елементів напряму ускладнює повторне використання компонентів.
4. Рішення:
    - Об'єкти взаємодіють через посередника.
    - Всі компоненти залежать лише від посередника, а не один від одного.
5. Переваги:
    - Усуває залежності між компонентами.
    - Спрощує взаємодію та повторне використання.
    - Централізує управління.
6. Недоліки:
    - Посередник може стати надто складним.
7. Приклад з життя:
    - Диспетчер літаків координує їх дії, зменшуючи ризик катастроф.

Шаблон «Facade» (Фасад):

1. Призначення:
    - Єдиний інтерфейс для взаємодії зі складною підсистемою.
    - Ховає внутрішні деталі реалізації підсистеми.
2. Проблема:
    - Робота з великою кількістю об'єктів складної бібліотеки ускладнює бізнес-логіку.
3. Рішення:

- Фасад надає простий інтерфейс для використання лише необхідного функціоналу підсистеми.

4. Переваги:
    - Ізолює клієнтів від складної підсистеми.
    - Спрощує інтеграцію та підтримку.
5. Недоліки:
    - Ризик перетворення фасада на "божественний об'єкт", занадто прив'язаний до багатьох компонентів.
6. Приклад з життя:
    - Співробітник магазину є фасадом до системи створення замовлень, платіжної системи та служби доставки.

Шаблони «Bridge» (Міст) і «Template Method» (Шаблонний метод) вирішують різні завдання, але обидва підвищують гнучкість і масштабованість коду. Давайте коротко підсумуємо їхні ключові аспекти:

Шаблон «Bridge»

Призначення:

- Розділяє абстракцію (інтерфейс) і її реалізацію, дозволяючи їх змінювати незалежно.
- Особливо корисний, коли потрібно поєднувати множинні ієрархії (наприклад, типи об'єктів і способи їх реалізації).

Приклад: Розділення фігур і кольорів. Замість створення комбінацій класів (СинєКоло, ЧервонийКвадрат) створюються окремі ієрархії:

1. Класи фігур: Коло, Квадрат тощо.
2. Класи кольорів: Синій, Червоний тощо. Фігури зберігають посилання на об'єкти кольорів і делегують їм відповідні завдання.

Переваги:

- Спрощує розширення.
- Реалізує принцип відкритості/закритості.
- Зменшує кількість класів, необхідних для підтримки коду.

Недоліки:

- Ускладнює початкову реалізацію через додаткові класи.

Сфера застосування:

- Побудова незалежних від платформи систем.
- Об'єкти, які мають різні аспекти, що часто змінюються незалежно.

Шаблон «Template Method»

Призначення:

- Створює основу алгоритму в абстрактному класі, залишаючи реалізацію окремих кроків підкласам.
- Використовується для стандартизації алгоритму з можливістю його адаптації.

Приклад: Парсери для документів:

1. Базовий клас визначає основні кроки парсингу: відкрити, зчитати, закрити.
2. Кожен підклас реалізує специфічні дії для свого формату (PDF, DOC, CSV).

Переваги:

- Уніфікація коду завдяки спільному алгоритму.
- Легко додаються нові варіанти поведінки через підкласи.

Недоліки:

- Жорстка структура алгоритму.
- Може порушити принцип Лісков, якщо підкласи змінюють базову поведінку.
- Великий шаблонний метод може стати важким для підтримки.

Сфера застосування:

- Реалізація стандартного алгоритму з варіативними кроками.
- Побудова систем з однаковими загальними принципами і різними деталями.

Хід роботи

Тема 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.

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

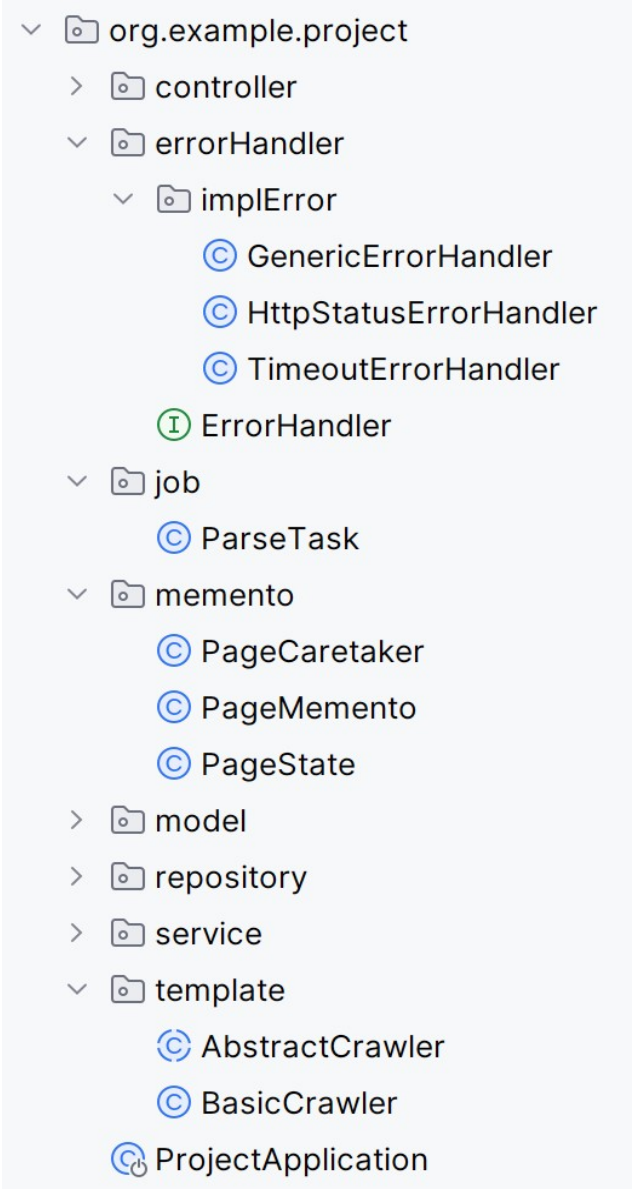


Рис. 1: Структура проекту

У ході лабораторної роботи була реалізована система кравлера, де основним завданням було використання патерну template method.

2. Реалізація шаблону template method

Паттерн template method забезпечує механізм визначення шаблону алгоритму в базовому класі з можливістю конкретизації окремих етапів у підкласах. Це дозволяє структурувати загальну логіку роботи, залишаючи специфічні деталі на рівні конкретної реалізації. У нашому проекті парсера цей паттерн можна застосувати для визначення стандартного алгоритму обробки веб-сторінки, де окремі етапи (наприклад, обробка посилань, зображень, тексту тощо) можуть бути змінені або розширені в підкласах. Це дозволяє легко адаптувати алгоритм для різних типів контенту або спеціальних умов без дублювання основної логіки.

Таке рішення забезпечує гнучкість і масштабованість, адже, навіть при зміні специфічних деталей обробки, загальна структура алгоритму залишається незмінною. Це також спрощує підтримку та розширення системи, адже нові варіанти обробки можна додавати, не змінюючи базового коду.

Опис класів

1. Клас ParseTask

```java
import org.example.project.template.BasicCrawler;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.util.List;

2 inheritors  alionafomenko
@Component
public class ParseTask {

    @Autowired
    private BasicCrawler crawler;



    alionafomenko
    @Scheduled(fixedDelay = 10000)
    public void parseSite() throws Exception {
        List<org.example.project.model.Document> documentList = crawler.documentService.getAllDocs();
        // System.out.println(documentList);
        for (org.example.project.model.Document document : documentList) {
            int level = document.getLevel();
            if (level < 5) {
                crawler.crawl(document.getUrl(), document.getSiteId(),  level: level + 1, document.getId());

            }
        }
    }
```

Рис. 2: Клас ParseTask

- Призначення:
  Організовує процес краулінгу сторінок, використовуючи функціонал класу BasicCrawler.
- Особливості:
  - Викликає метод crawl для кожної сторінки, отриманої з DocumentService.
  - Обробляє сторінки до п’ятого рівня вкладеності.

2. Клас AbstractCrawler

```
1 usage   1 inheritor   alionafomenko
public abstract class AbstractCrawler {

    @Autowired
    protected PictureService pictureService;

    @Autowired
    public DocumentService documentService;

    1 usage
    protected PageCaretaker caretaker = new PageCaretaker();
```

Рис. 3: Клас AbstractCrawler частина 1

```
1 usage   alionafomenko
public void crawl(String url, int siteId, int level, int documentId) throws Exception {
    StringBuilder content = new StringBuilder();
    String title = "";
    int httpStatus = 0;

    PageState pageState = new PageState();
    System.out.println(url);
    try {
        Document doc = Jsoup.connect(url)
                .userAgent( s: "AlionaCrawler")
                .timeout( i: 5000)
                .followRedirects( b: true)
                .referrer( s: "https://google.com")
                .get();

        pageState.setState(doc.html(), doc.title());
        caretaker.save(pageState.saveStateToMemento());

        Elements elements = doc.getAllElements();
        httpStatus = doc.connection().response().statusCode();

        for (Element element : elements) {
            processElement(element, siteId, level, url, content);
            if ("title".equals(element.tagName())) {
                title = element.ownText();
            }
        }

    } catch (IOException e) {
        httpStatus = handleIOException(e);
    } finally {
        documentService.saveContent(documentId, title, content.toString(),  status: "scanned", httpStatus);
    }
}
```

Рис. 4: Клас AbstractCrawler частина 2

```
1 usage  1 implementation  alionafomenko
protected abstract int handleIOException(Exception e);

1 usage  alionafomenko
private void processElement(Element element, int siteId, int level, String parentUrl, StringBuilder content) {
    String tagName = element.tagName();
    if ("a".equals(tagName)) {
        handleLink(element, siteId, level, parentUrl);
    } else if ("img".equals(tagName)) {
        handleImage(element, siteId, parentUrl);
    } else {
        String text = element.ownText();
        if (!text.isEmpty()) {
            content.append("<p>").append(text).append("</p>\n");
        }
    }
}

1 usage  alionafomenko
private void handleLink(Element element, int siteId, int level, String parentUrl) {
    String link = element.attr( attributeKey: "href");
    if (!link.endsWith(".jpg") && !link.endsWith(".png") && !link.endsWith(".jpeg") && !link.endsWith(".svg")) {
        if (!link.startsWith("http")) {
            if (link.startsWith("/")) {
                documentService.addDoc(siteId, link, parentUrl, status: "to_do", level);
            }
        } else {
            documentService.addDoc(siteId, link, parentUrl, status: "external_link", level: 0);
        }
    }
}
```

Рис. 5: Клас AbstractCrawler частина 3

```
    1 usage  alionafomenko
    private void handleImage(Element element, int siteId, String parentUrl) {
        String picLink = element.attr( attributeKey: "src");
        pictureService.addPicture(siteId, picLink, parentUrl);
    }
}
```

Рис. 6: Клас AbstractCrawler частина 4

- Призначення:
  Базовий клас для реалізації шаблону Template Method, що визначає загальну структуру алгоритму краулінгу.
- Особливості:
  - Шаблонний метод crawl:
    Задає стандартний процес обробки сторінки:
    1. Завантаження сторінки (Jsoup.connect).
    2. Збереження стану через PageMemento.
    3. Парсинг елементів сторінки (посилання, зображення, текст).
    4. Обробка помилок через метод handleIOException.
  - Абстрактний метод handleIOException:
    Використовується для обробки виключень (визначається в підкласах).
  - Методи для обробки елементів сторінки:

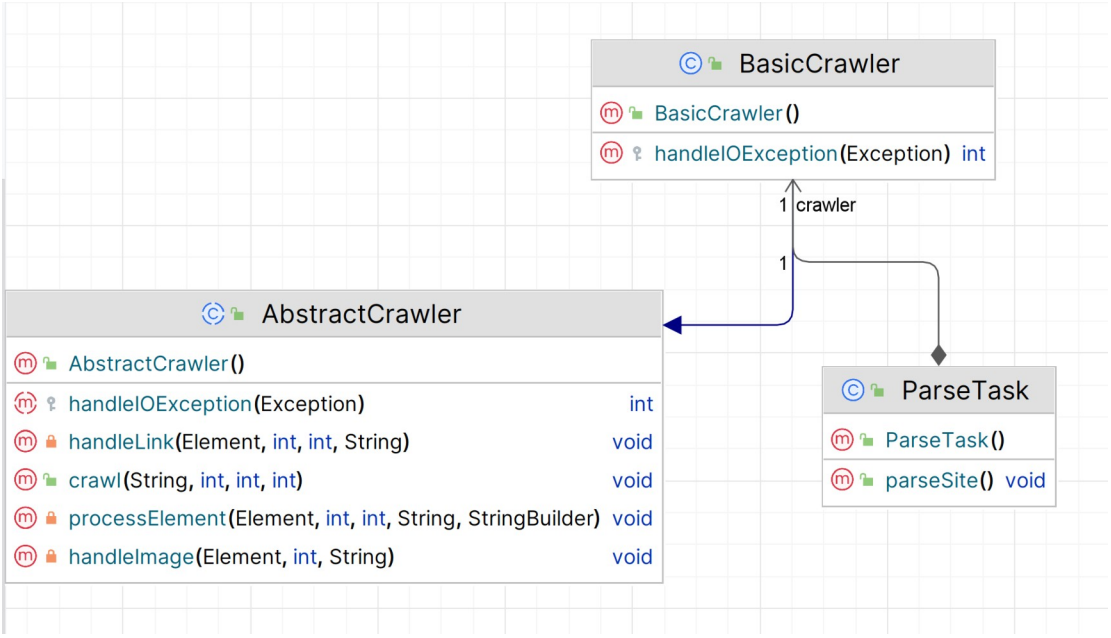
1. processElement — викликає відповідний метод обробки залежно від типу елемента (посилання, зображення, текст).
2. handleLink — обробляє посилання.
3. handleImage — обробляє зображення.

3. **Клас BasicCrawler**

```java
package org.example.project.template;

import org.springframework.stereotype.Component;

new *
@Component
public class BasicCrawler extends AbstractCrawler {

    1 usage  new *
    @Override
    protected int handleIOException(Exception e) {
        if (e instanceof org.jsoup.HttpStatusException) {
            return ((org.jsoup.HttpStatusException) e).getStatusCode();
        } else if (e instanceof java.net.SocketTimeoutException) {
            return 0;
        } else if (e instanceof org.jsoup.UnsupportedMimeTypeException) {
            return 415;
        } else {
            return 500;
        }
    }
}
```

Рис. 7: Клас BasicCrawler

- Призначення:
  Конкретний клас, що реалізує обробку помилок для краулінгу веб-сторінок.
- Особливості:
  - Реалізує метод handleIOException, що обробляє:
    - HttpStatusException — повертає код HTTP-статусу.
    - SocketTimeoutException — повертає код 0.
    - UnsupportedMimeTypeException — повертає код 415.
    - Інші помилки — повертає код 500.



Фигура 1: Діаграма класів

Компоненти діаграми

1. AbstractCrawler (Абстрактний клас):
    - Є базовим класом для логіки краулінгу.
    - Методи:
        - handleIOException(Exception): int — обробка винятків під час роботи краулера (захищений метод).
        - handleLink(Element, int, int, String): void — обробляє посилання на елемент.
        - crawl(String, int, int, int): void — основний метод для краулінгу веб-ресурсів.
        - processElement(Element, int, int, String, StringBuilder): void — обробляє конкретний елемент.
        - handleImage(Element, int, String): void — обробка зображень у веб-сторінках.
    - Цей клас служить шаблоном (Template Method), визначаючи базову структуру краулінгу й залишаючи конкретну реалізацію для підкласів.
2. BasicCrawler (Клас, що успадковує AbstractCrawler):
    - Реалізує основний функціонал краулінгу, розширюючи метод handleIOException(Exception).
    - BasicCrawler() — конструктор.
    - handleIOException(Exception): int — обробляє винятки та повертає код результату.
3. ParseTask (Клас для задач парсингу):

- Використовує BasicCrawler для виконання парсингу.
- parseSite(): void — основний метод для запуску задачі парсингу сайту.
- Має агрегацію з BasicCrawler:
    - Відношення "один до одного" означає, що ParseTask використовує екземпляр BasicCrawler для роботи.

Проблема, яку вирішує патерн "Template Method"

Стандартизація алгоритму з можливістю його конкретизації

Патерн Template Method дозволяє визначити загальний алгоритм у базовому класі, а деталі його реалізації залишити для конкретних підкласів. Це особливо корисно, коли потрібно стандартизувати основну логіку роботи, але забезпечити гнучкість для різних варіантів реалізації.

Наприклад, у системі краулінгу (як у AbstractCrawler), шаблонний метод визначає загальну послідовність дій:

- підключення до сторінки;
- обробка елементів (посилань, зображень, тексту);
- збереження результатів;
- обробка помилок.

При цьому специфіка обробки помилок або типів контенту може бути змінена у підкласах (BasicCrawler). Це дозволяє створювати різні стратегії для різних сценаріїв, не змінюючи структуру основного алгоритму.

Переваги використання патерну "Template Method"

Узгодженість і стандартизація:

- Забезпечує єдину структуру алгоритму для всіх реалізацій.
- Уніфікує процес роботи, залишаючи можливість конкретизації окремих кроків.

Розширюваність:

- Нові варіанти реалізації можуть бути додані без зміни базового алгоритму.
- Легко розширити поведінку за рахунок створення підкласів із власними стратегіями.

Локалізація логіки:

- Базовий клас містить загальну логіку, тоді як підкласи зосереджені лише на конкретних деталях.

Висновок:
У даній лабораторній роботі я реалізувала патерн проєктування "Template Method" для стандартизації та узгодження алгоритму роботи системи. Це дозволило визначити загальну структуру процесу краулінгу, залишаючи можливість конкретизації окремих етапів у підкласах. Завдяки цьому патерну було забезпечено спільну логіку роботи для всіх реалізацій, що спрощує підтримку системи та додає їй гнучкості.

Крім того, використання "Template Method" дозволило створити єдину основу для розширення функціоналу. Додавання нових варіантів реалізації, таких як обробка специфічних помилок або додаткових типів контенту, тепер можливе без змін у базовому алгоритмі. Це зробило систему масштабованою та зручною для адаптації під нові вимоги.