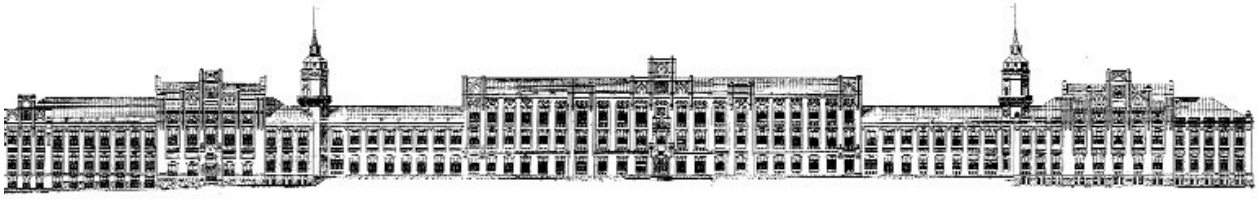

Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №7

з дисципліни «Технології розроблення

програмного забезпечення»

Тема: «Патерни проектування»

Виконала
студентка групи ІА-32:
Павлюк В. О.

Зміст

Table of Contents

Вступ

Темою роботи є «Вступ до патернів проєктування». Мета цієї лабораторної це вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

Як об’єкт проєктування обрано тему «Web crawler»: програмний сканер, що розпізнає структуру сторінок сайту, переходить за посиланнями, збирає інформацію за заданим терміном, видаляє несемантичні елементи (рекламу, скрипти тощо), зберігає очищені дані у вигляді структурованого набору HTML-файлів і веде статистику відвідань та метадані.

1 Теоретичні відомості

1.1. Шаблон «Mediator»

Призначення патерну: Шаблон «Mediator» (посередник) використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного). Даний шаблон схожий на шаблон «команда», проте в даному випадку замість зберігання даних про конкретну дію, зберігаються дані про взаємодії між компонентами. Даний шаблон зручно застосовувати у випадках, коли безліч об'єктів взаємодіє між собою деяким структурованим чином, однак складним для розуміння. У такому випадку вся логіка взаємодії виноситься в окремий об'єкт. Кожен із взаємодіючих об'єктів зберігає посилання на об'єкт «медіатор». «Медіатор» нагадує диригента при управлінні оркестром. Диригент стежить за тим, щоб кожен інструмент грав в правильний час і в злагоді з іншими інструментами. Функції «медіатора» повністю це повторюють.

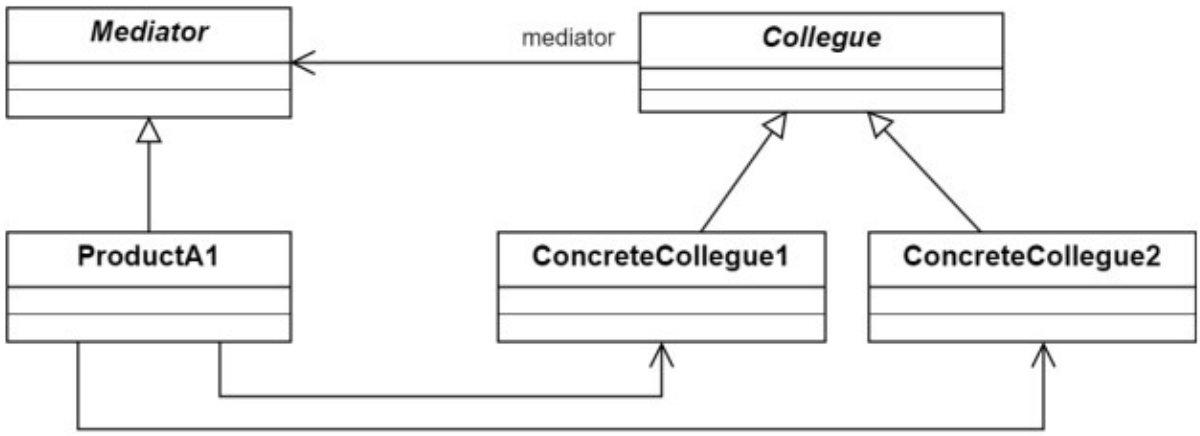

Рисунок 1.1. Структура патерна «Медіатор»

1.2. Шаблон «Facade»

Призначення патерну: Шаблон «Facade» (фасад) передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми . Оскільки підсистема може складатися з безлічі класів, а кількість її функцій – не більше десяти, то щоб уникнути створення «спагетікоду» (коли все тісно пов'язано між собою) виділяють один загальний інтерфейс доступу, здатний правильним чином звертатися до внутрішніх деталей. Це також відволікає користувачів від змін в підсистемі (внутрішня реалізація може змінюватися, а наданої послуги немає), що також скоротить кількість змін в використовуваних фасад класах (без фасаду довелося б

змінювати вихідні коди в безлічі точок). 82 Звичайно, твердої умови повного закриття внутрішніх класів підсистеми не стоїть – при необхідності можна звертатися до окремих класів безпосередньо, минаючи об'єкт фасад.

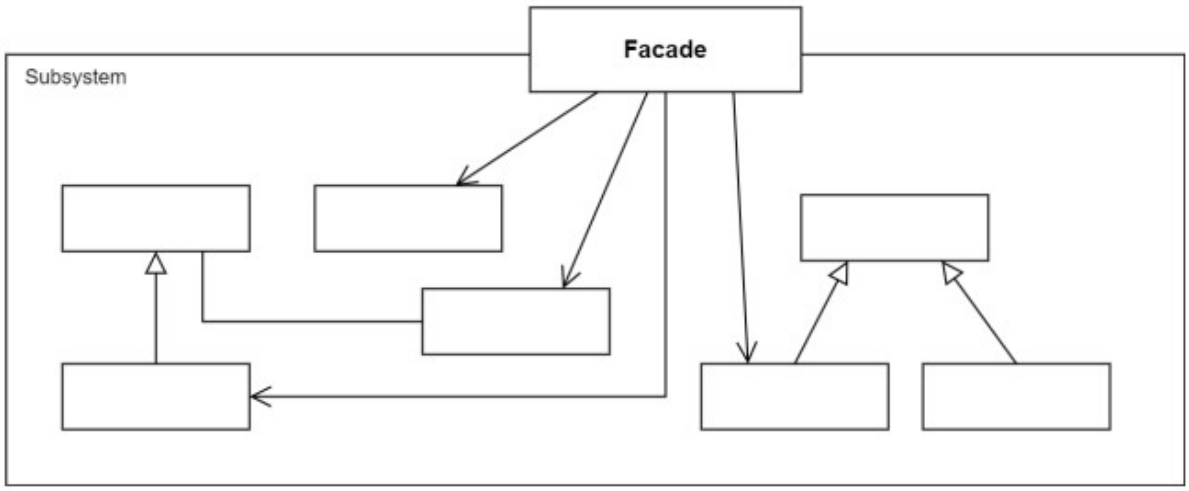

Рисунок 1.2. Структура патерну «Фасад»

1.3. Шаблон «Bridge»

Призначення патерну: Шаблон «Bridge» (міст) використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами

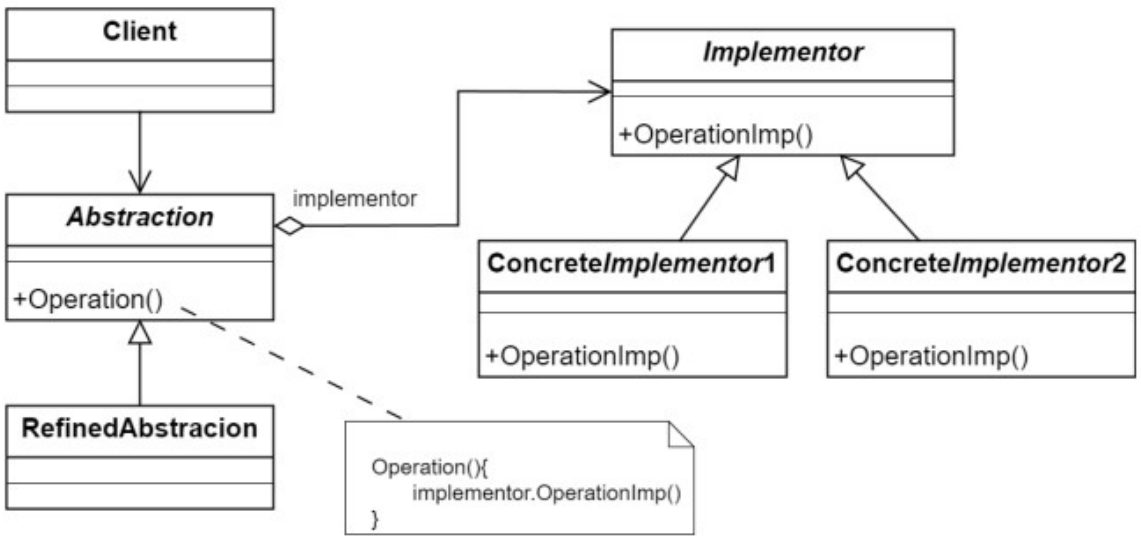

Рисунок 1.3. Структура патерну «Міст»

1.4. Шаблон «Template Method»

Призначення патерну: Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам. Можна привести в приклад формування вебсторінки: необхідно додати заголовки, вміст сторінки, файли, що додаються, і нижню частину сторінки. Код для додавання вмісту сторінки може бути абстрактним і реалізовуватися в різних класах – AspNetCompiler, HtmlCompiler, PhpCompiler і т.п. Додавання всіх інших елементів виконується за допомогою вихідного абстрактного класу з алгоритмом.

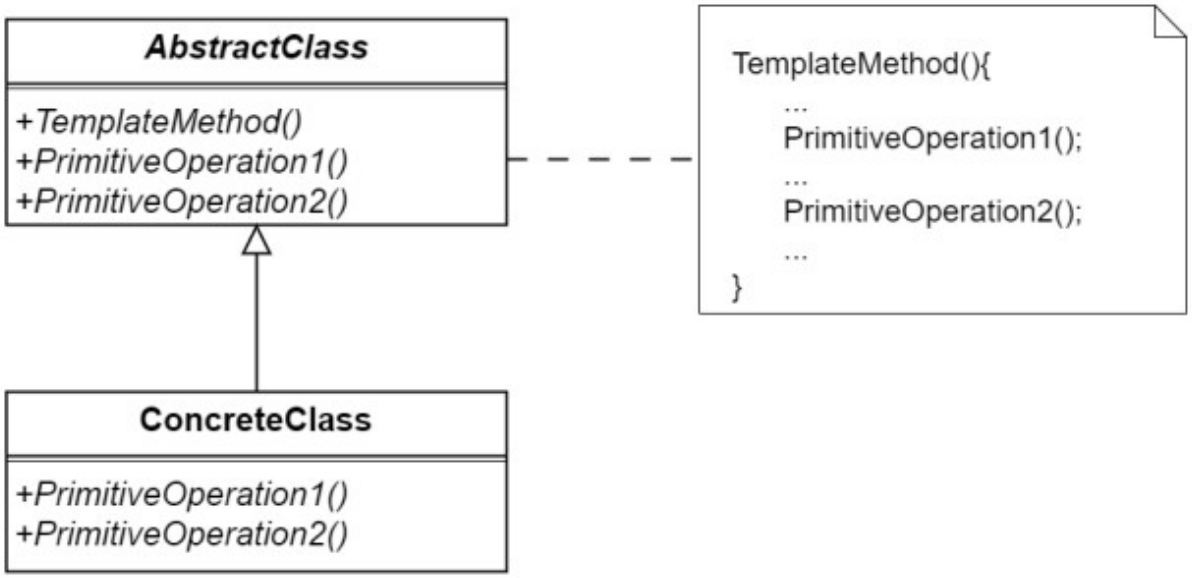

Рисунок 1.4. Структура патерну «Шаблонний метод»

2 Хід роботи

2.1. Реалізувати шаблонний метод, який задає фіксовану структуру алгоритму crawl та дозволяє розширювати окремі кроки.

Завдання

- Винести структуру обходу в базовий клас AbstractCrawlTemplate.
- Реалізувати конкретний варіант DefaultCrawlTemplate.
- Передбачити hook-методи для розширення (використано в Lab 9).

2.2 Діаграма класів (фрагмент)

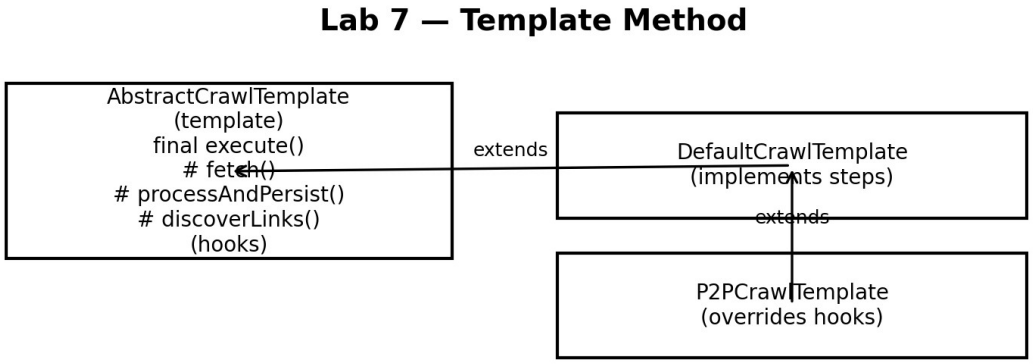

Рисунок 2.2 - Діаграма класів (фрагмент)

2.3. Фрагмент коду (Java)

AbstractCrawlTemplate: execute() (скелет алгоритму)

```
    public final ScanSession execute(CrawlProfile profile, CrawlRunOptions options)
throws Exception {
        ScanSession session;
        UrlScheduler scheduler = new UrlScheduler(profile.getDomainFilter(),
profile.getMaxDepth());
        int processed = 0;

        if (options != null && options.getResumeSessionId() != null) {
            Long resumeId = options.getResumeSessionId();
            session = sessionRepo.findById(resumeId)
                    .orElseThrow(() -> new IllegalArgumentException("No session found
```

```
with id=" + resumeId));
            CrawlCheckpoint checkpoint = checkpointStore.load(resumeId);
            if (checkpoint == null) {
                throw new IllegalStateException("No checkpoint found for session id=" +
resumeId);
            }
            scheduler.restore(checkpoint.getSchedulerMemento());
            processed = checkpoint.getPagesProcessed();
            session.markResumed();
            sessionRepo.save(session);
        } else {
            session = sessionRepo.createSession(profile.getName());
            scheduler.addStartUrls(profile.getStartUrls());
        }

        beforeCrawl(profile, session, scheduler);

        try {
            UrlTask task;
            while ((task = scheduler.nextTask()) != null) {
                String url = task.getUrl();
                int depth = task.getDepth();

                String html = fetch(url);
                PageData page = processAndPersist(session, profile, url, html);
                processed++;

                List<String> links = discoverLinks(html);
                scheduler.scheduleDiscoveredLinks(links, depth);
                onLinksDiscovered(profile, session, links, depth + 1);

                if (options != null && options.getPauseAfterPages() > 0 && processed >=
options.getPauseAfterPages()) {
                    checkpointStore.save(new CrawlCheckpoint(session.getId(),
profile.getId(), scheduler.createMemento(), processed));
                    session.markPaused();
                    sessionRepo.save(session);
                    afterCrawl(profile, session, scheduler);
                    return session;
                }
            }

            session.markCompleted();
            sessionRepo.save(session);
```

```
            checkpointStore.remove(session.getId());
            afterCrawl(profile, session, scheduler);
            return session;
        } catch (Exception e) {
            session.markFailed();
            sessionRepo.save(session);
```

DefaultCrawlTemplate: реалізація кроків

```
package ua.kpi.webcrawler.template;

import ua.kpi.webcrawler.core.DefaultContentProcessorFactory;
import ua.kpi.webcrawler.http.HttpClient;
import ua.kpi.webcrawler.model.CrawlProfile;
import ua.kpi.webcrawler.model.PageData;
import ua.kpi.webcrawler.model.ScanSession;
import ua.kpi.webcrawler.processing.PageHandler;
import ua.kpi.webcrawler.processing.PageProcessingContext;
import ua.kpi.webcrawler.memento.CrawlCheckpointStore;
import ua.kpi.webcrawler.repository.InMemoryPageRepository;
import ua.kpi.webcrawler.repository.InMemorySessionRepository;

import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;


public class DefaultCrawlTemplate extends AbstractCrawlTemplate {

    protected final HttpClient httpClient;
    protected final DefaultContentProcessorFactory processorFactory;

    public DefaultCrawlTemplate(HttpClient httpClient,
                                DefaultContentProcessorFactory processorFactory,
                                InMemorySessionRepository sessionRepo,
                                InMemoryPageRepository pageRepo,
                                CrawlCheckpointStore checkpointStore) {
        super(sessionRepo, pageRepo, checkpointStore);
        this.httpClient = httpClient;
        this.processorFactory = processorFactory;
    }

    @Override
    protected String fetch(String url) throws Exception {
```

```
        System.out.println("Fetching " + url);
        return httpClient.get(url);
    }

    @Override
    protected PageData processAndPersist(ScanSession session, CrawlProfile profile,
String url, String html) throws Exception {
        PageHandler chain = processorFactory.buildChain(profile);
        PageProcessingContext ctx = new PageProcessingContext(html);
        chain.handle(ctx);

        PageData pageData = pageRepo.create(session.getId(), url, ctx.getPlainText(),
ctx.getKeywordCounts());
        System.out.println("Saved page " + pageData.getId() + " keywordStats=" +
pageData.getKeywordCounts());
        return pageData;
    }

    @Override
    protected List<String> discoverLinks(String html) {
```

Висновки

У ході виконання лабораторної роботи було реалізовано шаблонний метод, який задає фіксовану структуру алгоритму crawl та дозволяє розширювати окремі кроки. Шаблонний метод дозволив стандартизувати алгоритм обходу, зменшити дублювання коду та зробити логіку розширюваною через перевизначення окремих кроків і хуків.

Контрольні запитання

1. Яке призначення шаблону «Посередник»?

Шаблон «Посередник» (Mediator) призначений для зменшення кількості прямих зв’язків між об’єктами, винісши їх взаємодію в окремий об’єкт-посередник. Це спрощує комунікацію між компонентами та зменшує залежності.

2. Нарисуйте структуру шаблону «Посередник».

Структура включає:

- Mediator – інтерфейс посередника;
- ConcreteMediator – конкретний посередник, який координує взаємодію;
- Colleague – базовий клас/інтерфейс учасників;
- ConcreteColleagueA/B – конкретні учасники, які спілкуються через Mediator.

3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

У шаблон входять Mediator/ConcreteMediator та Colleague/ConcreteColleague. ConcreteColleague не викликають один одного напряму: вони повідомляють Mediator про події, а Mediator вирішує, кого і як потрібно оповістити або який метод викликати в інших колег.

4. Яке призначення шаблону «Фасад»?

Шаблон «Фасад» (Facade) надає спрощений єдиний інтерфейс до складної підсистеми. Він приховує внутрішню складність і зменшує кількість залежностей клієнта від багатьох класів підсистеми.

5. Нарисуйте структуру шаблону «Фасад».

Структура включає:

- Facade – клас із простими методами для клієнта;
- SubsystemClass1/2/3 – класи підсистеми, які виконують реальну роботу;
- Client – використовує лише Facade, а не підсистему напряму.

6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

У шаблон входять Facade, набір класів підсистеми та Client. Client викликає методи Facade. Facade всередині викликає потрібні методи підсистеми у

правильному порядку та повертає результат клієнту, ізолюючи його від деталей реалізації.

7. Яке призначення шаблону «Міст»?

Шаблон «Міст» (Bridge) відокремлює абстракцію від реалізації так, щоб їх можна було змінювати незалежно. Це дозволяє уникнути «вибуху» кількості підкласів при поєднанні різних варіантів абстракції та реалізації.

8. Нарисуйте структуру шаблону «Міст».

Структура включає:

- Abstraction – абстракція, яка має посилання на Implementor;
- RefinedAbstraction – розширена абстракція;
- Implementor – інтерфейс реалізації;
- ConcreteImplementorA/B – конкретні реалізації;
- Abstraction делегує частину роботи Implementor.

9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

У шаблон входять Abstraction/RefinedAbstraction та Implementor/ConcreteImplementor. Abstraction зберігає посилання на Implementor і викликає його методи. ConcreteImplementor реалізують низькорівневі операції, а RefinedAbstraction визначає високорівневу логіку, комбінуючи виклики реалізації.

10. Яке призначення шаблону «Шаблонний метод»?

Шаблон «Шаблонний метод» (Template Method) визначає каркас алгоритму в базовому класі, але дозволяє підкласам перевизначати окремі кроки алгоритму без зміни його загальної структури.

11. Нарисуйте структуру шаблону «Шаблонний метод».

Структура включає:

- AbstractClass – базовий клас із templateMethod() та абстрактними/хуковими методами;
- ConcreteClassA/B – підкласи, що реалізують конкретні кроки алгоритму;
- templateMethod() викликає кроки у фіксованому порядку.

12. Які класи входять в шаблон «Шаблонний метод», та яка між ними взаємодія?

У шаблон входять AbstractClass і ConcreteClass. AbstractClass містить templateMethod(), який викликає окремі кроки (primitive operations).

ConcreteClass перевизначують ці кроки. Взаємодія: клієнт викликає templateMethod(), а фактична реалізація кроків залежить від конкретного підкласу.

13. Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

«Шаблонний метод» керує структурою алгоритму (послідовністю кроків) і дає підкласам змінювати окремі кроки. «Фабричний метод» керує створенням об’єктів і дозволяє підкласам визначати, який саме продукт створювати.

14. Яку функціональність додає шаблон «Міст»?

«Міст» додає можливість незалежно розширювати абстракцію і реалізацію, комбінувати їх у будь-яких поєднаннях та змінювати реалізацію під час виконання (через підстановку конкретного Implementor), зменшуючи кількість класів і залежностей.