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 №6

Технології розробки програмного забезпечення

ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Виконала
студентка групи ІА-24
Тильна Марія Сергіївна

Перевірив:
Мягкий М.Ю.

Київ 2024

Зміст

Короткі теоретичні відомості..... 3
Хід роботи..... 4
Реалізація шаблону проєктування для майбутньої системи.....4
Зображення структури шаблону..... 9
Посилання на репозиторій..... 10
Висновок..... 10

Тема: ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Короткі теоретичні відомості

Abstract Factory (Абстрактна фабрика) – це створювальний шаблон, який надає інтерфейс для створення сімейства взаємопов'язаних об'єктів без вказівки їх конкретних класів. Абстрактна фабрика дозволяє створювати продукти, які належать до певної сім'ї, не залежачи від того, яка саме реалізація буде обрана. Цей шаблон корисний, коли потрібно забезпечити взаємодію різних продуктів, але без залежності від їх конкретних реалізацій.

Factory Method (Фабричний метод) – це створювальний шаблон, який визначає інтерфейс для створення об'єктів, але дозволяє підкласам змінювати тип створюваного об'єкта. Відмінність від шаблону **Abstract Factory** полягає в тому, що **Factory Method** забезпечує створення одиничного продукту, а не всієї сім'ї продуктів. Цей шаблон корисний, коли потрібно делегувати створення об'єкта підкласам і не прив'язуватися до конкретних класів об'єктів.

Memento (Мементо) – це поведінковий шаблон, який дозволяє зберігати та відновлювати попередній стан об'єкта без порушення інкапсуляції. Мементо використовується, коли потрібно зберегти стан об'єкта для подальшого відновлення, наприклад, для реалізації функції "відкату" (undo). Шаблон складається з трьох основних компонентів: **Originator** (об'єкт, чий стан зберігається), **Memento** (об'єкт, що зберігає стан) та **Caretaker** (об'єкт, який зберігає мементо).

Observer (Спостерігач) – це поведінковий шаблон, який визначає залежність типу "один до багатьох" між об'єктами, де зміни в одному об'єкті автоматично сповіщають усіх залежних від нього об'єктів. Це дозволяє створювати події або реакції на зміни, не знаючи точно, які об'єкти повинні реагувати. Шаблон **Observer** часто використовується для реалізації систем подій або підписки.

Decorator (Декоратор) – це структурний шаблон, який дозволяє динамічно додавати нові функціональні можливості об'єктам, не змінюючи їх структуру. Декоратор надає можливість обгорнути об'єкт у новий клас, який додає або змінює його поведінку. Це дає змогу гнучко модифікувати функціональність об'єкта без створення численних підкласів. Шаблон **Decorator** часто застосовується для розширення функціональності елементів у графічних інтерфейсах або для реалізації патернів, що включають додаткові можливості для об'єктів.

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.

У моєму проекті веб-сканера важливо забезпечити гнучке управління процесом сканування. Процес може бути тривалим, і існує потреба призупиняти його, а потім відновлювати з того ж місця без втрати вже отриманих даних. Саме для вирішення цієї задачі було використано шаблон **Memento**.

Шаблон Memento забезпечує інкапсуляцію стану сканера, оскільки збереження і відновлення відбувається через об'єкт-знімок, не порушуючи структуру основного сервісу.

Реалізація шаблону забезпечила можливість зберігати поточний стан процесу сканування. Завдяки цьому користувач може призупинити процес сканування в будь-який момент та відновити процес з того моменту, де воно було зупинене, без повторного обходу сторінок.

Збережений стан дозволяє асоціювати прогрес сканування із конкретною сесією. Це допомагає відслідковувати процес у базі даних і забезпечує більш чисту архітектуру.

```java
public class CrawlerState implements Serializable {
    2 usages
    private final Set<String> visitedUrls;
    2 usages
    private final Queue<String> urlQueue;
    2 usages
    private final String sessionId;

    3 usages
    public CrawlerState(Set<String> visitedUrls, Queue<String> urlQueue, String sessionId) {
        // Create defensive copies of collections
        this.visitedUrls = new HashSet<>(visitedUrls);
        this.urlQueue = new LinkedList<>(urlQueue);
        this.sessionId = sessionId;
    }

    1 usage
    public Set<String> getVisitedUrls() {
        return new HashSet<>(visitedUrls);
    }

    1 usage
    public Queue<String> getUrlQueue() {
        return new LinkedList<>(urlQueue);
    }
```

Рис. 1 – Код класу CrawlerState

Цей клас зберігає стан сканування: список відвіданих URL, поточну чергу URL та інші необхідні дані.

```java
public class CrawlerCaretaker {
    5 usages
    private final Stack<CrawlerState> stateHistory = new Stack<>();

    3 usages
    public void saveState(CrawlerState state) {
        stateHistory.push(state);
    }

    1 usage
    public CrawlerState restoreState() {
        if (!stateHistory.isEmpty()) {
            return stateHistory.pop();
        }
        return null;
    }

    1 usage
    public boolean hasSavedState() {
        return !stateHistory.isEmpty();
    }
    1 usage
    public void clearHistory() {
        stateHistory.clear();
    }
}
```

Рис. 2 – Код класу CrawlerCaretaker

Цей клас буде зберігає знімки стану (CrawlerState) та забезпечувати їх відновлення.

```java
@Override
public void pauseCrawling() {
    System.out.println("Attempting to pause crawling...");
    if (currentSession != null) {
        System.out.println("Current session #" + currentSession.getId() + " status: " + currentSession.getStatus());
        if ("IN_PROGRESS".equals(currentSession.getStatus())) {
            running = false;
            // Зберігаємо поточний стан
            caretaker.saveState(createMemento());
            System.out.println("Saved crawling state with " + visitedUrls.size() +
                " visited URLs and " + urlQueue.size() + " URLs in queue");

            currentSession.setStatus("PAUSED");
            sessionRepository.save(currentSession);
            System.out.println("Session #" + currentSession.getId() + " paused successfully");
        } else {
            System.out.println("Cannot pause: session is not in progress (status: " + currentSession.getStatus() + ")");
        }
    } else {
        System.out.println("Cannot pause: no active session");
    }
}
```

```java
@Override
public void resumeCrawling() {
    System.out.println("Attempting to resume crawling...");
    if (currentSession != null) {
        System.out.println("Current session #" + currentSession.getId() + " status: " + currentSession.getStatus());
        if ("PAUSED".equals(currentSession.getStatus())) {
            // Відновлюємо стан
            if (caretaker.hasSavedState()) {
                CrawlerState savedState = caretaker.restoreState();
                if (savedState.getSessionId().equals(currentSession.getId().toString())) {
                    restoreFromMemento(savedState);
                    System.out.println("Restored previous crawling state with " +
                        visitedUrls.size() + " visited URLs and " +
                        urlQueue.size() + " URLs in queue");

                    running = true;
                    currentSession.setStatus("IN_PROGRESS");
                    sessionRepository.save(currentSession);
                    System.out.println("Session #" + currentSession.getId() + " resumed successfully");
                    new Thread(() -> crawl()).start();
```

Рис. 3 – Приклад використання Memento

Як це працює:

1) Під час сканування:

- Краулер відвідує сторінки і додає їх у visitedUrls
- Нові знайдені посилання додаються в urlQueue

2) Коли користувач натискає "Пауза" (/api/crawler/pause):

- Створюється об'єкт *CrawlerState* з поточними *visitedUrls* та *urlQueue*
- Цей об'єкт зберігається в *CrawlerCaretaker*
- Сканування призупиняється *(running = false)*
- Статус сесії змінюється на "PAUSED"

3) **Коли користувач натискає "Відновити" (/api/crawler/resume)**:

- *CrawlerCaretaker* повертає останній збережений стан
- Перевіряється чи цей стан належить поточній сесії
- Відновлюються *visitedUrls* та *urlQueue*
- Сканування відновлюється з того місця, де зупинилось

4) При повній зупинці (/api/crawler/stop):

- Вся історія станів очищається через *clearHistory()*
- Сесія позначається як "COMPLETED"

Таким чином, шаблон Memento дозволяє:

- Зберегти повний стан краулера в будь-який момент
- Відновити стан точно таким, яким він був при паузі
- Захистити дані стану від модифікацій (через immutable поля та defensive copies)
- Прив'язати стан до конкретної сесії сканування

Зображення структури шаблону

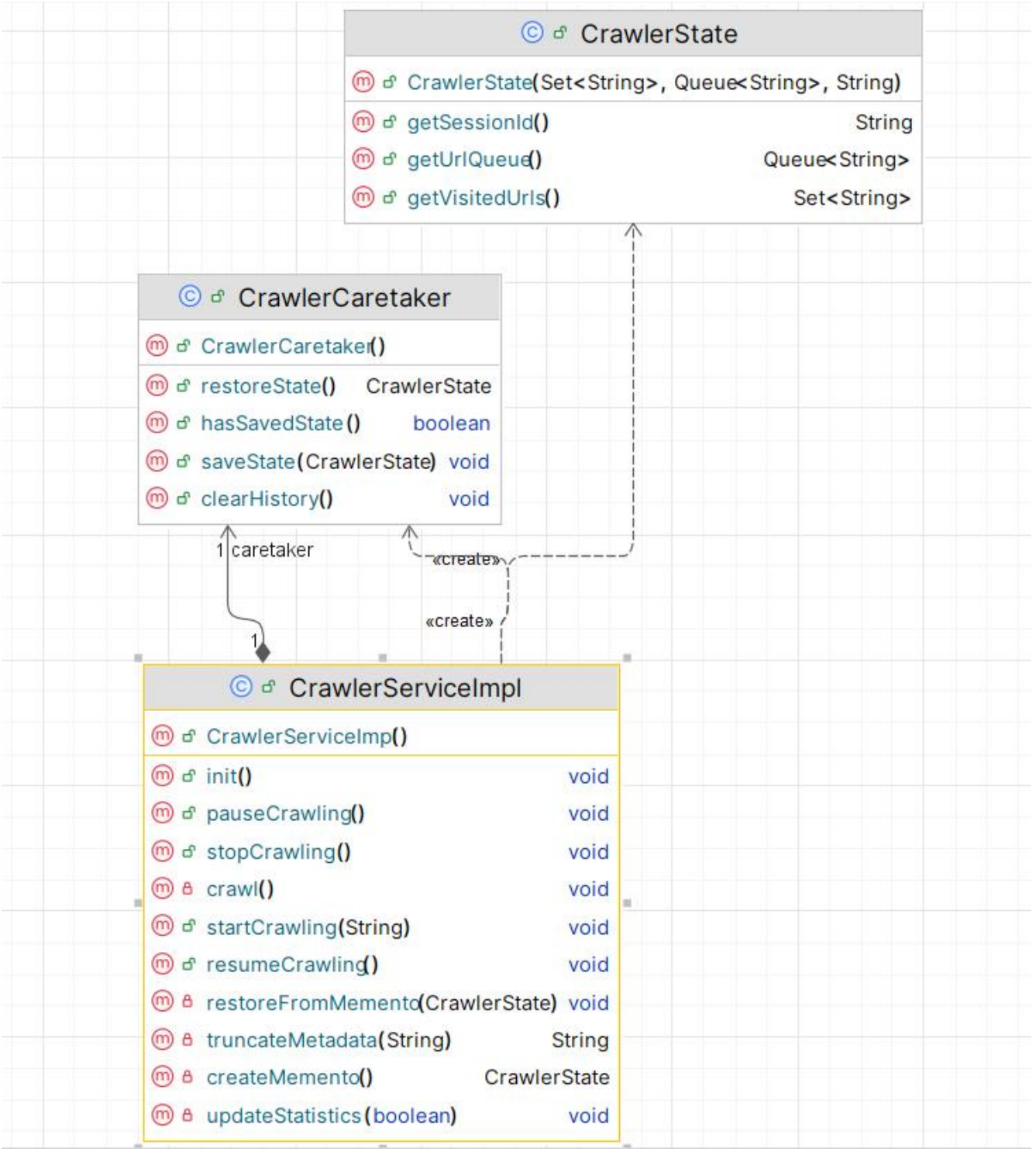

Рис. 4 – Структура шаблону

Посилання на репозиторій:

https://github.com/mashunchik/Webcrawler

Висновок: В процесі виконання лабораторної роботи я ознайомилась з такими патернами проектування як «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator». Особливу увагу приділила шаблону Memento, який дозволив реалізувати функціональність збереження та відновлення стану сканування без повторного обходу сторінок, зберігаючи поточну сесію. Використання шаблону Memento у моєму проекті веб-сканера надає можливість ефективно управляти процесом сканування, зберігати та відновлювати прогрес без втрати даних, підвищити надійність системи у разі збоїв або втручання користувача. Це робить систему більш гнучкою та стійкою, що є критично важливим для роботи з великими обсягами даних та тривалими процесами.