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 №8

Технології розроблення програмного забезпечення

«ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR» »

Виконала
студентка групи ІА–22:
Фоменко Альона

Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

Тема: ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»

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

Шаблони роботи з БД при розробці корпоративних додатків

1. Active Record (Активний запис)

   Об'єкт управляє як даними, так і поведінкою. Кожен об'єкт є обгорткою для одного рядка з БД, включає логіку поводження з даними. Використовується у простих сценаріях, наприклад, у Ruby on Rails. Зі збільшенням складності запитів логіку виділяють в окремий об'єкт.

2. Table Data Gateway (Шлюз до даних)

   Окремий клас взаємодіє з БД для кожного класу даних. Зберігає логіку запитів, зокрема збереження і видалення записів. Підхід забезпечує гнучкість і тестованість, але потребує абстрагування загальних операцій у базовий клас.

3. Data Mapping (Відображення даних)

   Вирішує проблему невідповідності між об'єктами даних і реляційними джерелами. Створює об'єкти або методи для перетворення даних, виправляючи різницю в типах. Типовий приклад — перетворення властивостей об'єкта у колонки таблиці.

Інші шаблони проектування, пов'язані з ієрархіями і оптимізацією:

1. Composite (Компонувальник)

   Формує деревоподібну структуру для ієрархій "частина-ціле". Уніфікує обробку окремих і вкладених об'єктів. Приклад: дерево замовлень, де коробка містить продукти або інші коробки.

Структура:

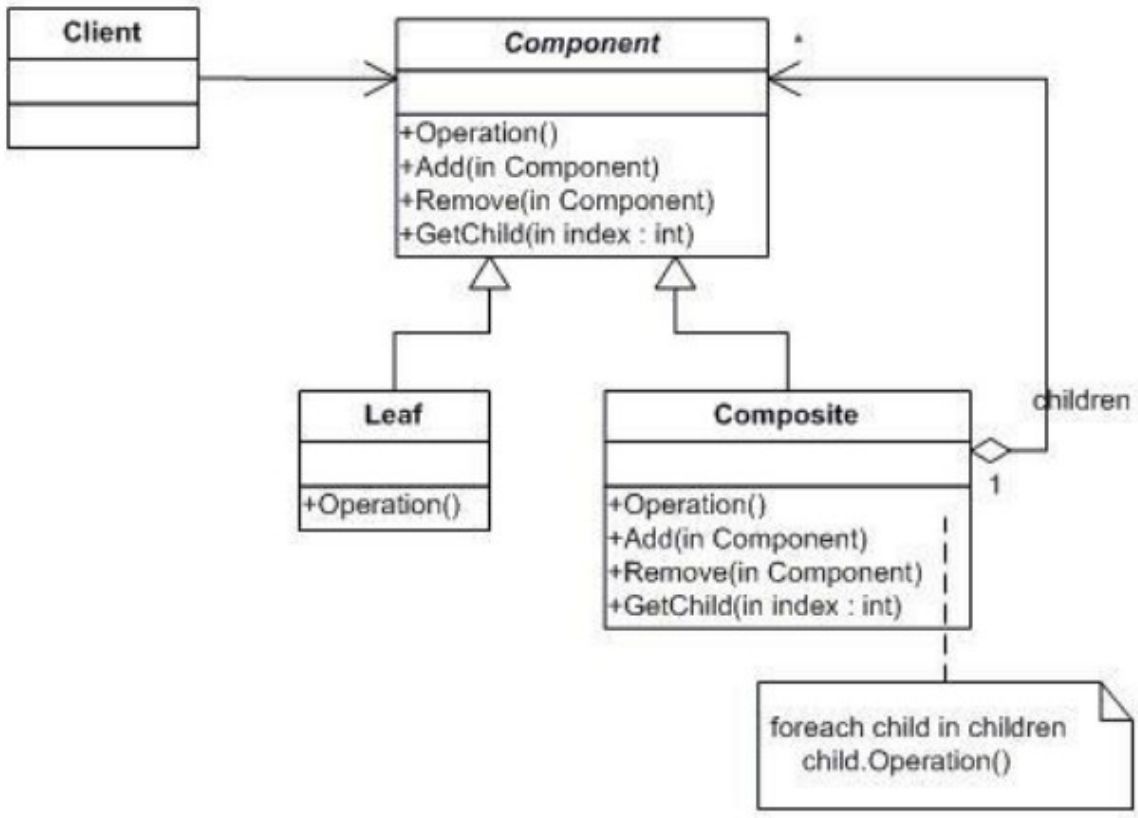

Рис. 1: Структура шаблону Composite

2. Flyweight (Легковаговик)

   Зменшує кількість об'єктів шляхом спільного використання їхніх внутрішніх станів. Ефективно використовується для оптимізації пам’яті у випадках множинних однакових об’єктів, наприклад, графічних елементів чи об’єктів БД.

3. Interpreter (Інтерпретатор)

   Описує граматику мови та її інтерпретатор через термінальні і нетермінальні символи. Використовується у невеликих мовах або при вирішенні часто змінюваних задач, таких як пошук рядків за зразком.

4. Visitor (Відвідувач)

   Дозволяє додавати нові операції над елементами об’єктної структури без зміни їхньої структури. Зручно при роботі з різнотипними об’єктами (наприклад, у компіляторах для обходу синтаксичних дерев).

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.

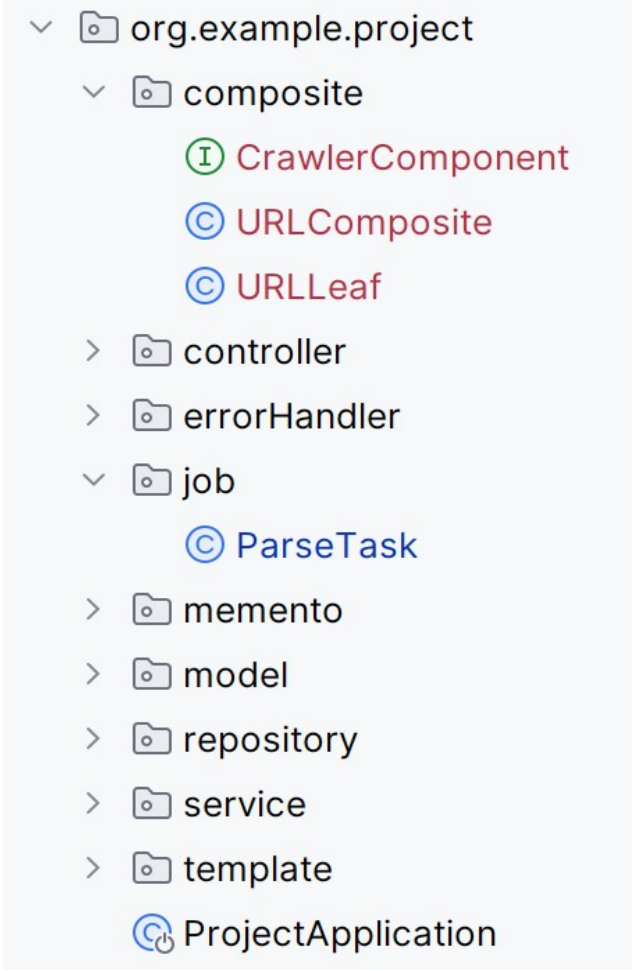

Рис. 2: Структура проекту

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

2. Реалізація шаблону template method

Паттерн **Composite** дозволяє обробляти окремі об'єкти та їх композиції в однаковий спосіб, що забезпечує гнучкість у побудові деревоподібних структур. Цей патерн застосовується для створення складних об'єктів, де кожен елемент (як окремий об'єкт, так і складена структура) має однакову інтерфейсну реалізацію. У проекті парсера цей паттерн можна застосувати для організації структури обробки елементів веб-сторінки, де кожен елемент (наприклад, окремий текст або посилання) може бути частиною більш складної структури, яка містить в собі інші елементи (наприклад, блоки тексту або списки посилань). Завдяки використанню шаблону **Composite**, система може ефективно управляти як окремими об'єктами, так і їх композиціями, забезпечуючи однакове поводження з ними через спільний інтерфейс. Це дозволяє легко масштабувати та модифікувати систему, оскільки додавання нових компонентів або складних структур не потребує значних змін у вже існуючому коді.

Такий підхід дозволяє спростити підтримку та розширення системи, адже обробка елементів і їх композицій може бути адаптована без зміни основної логіки роботи. Водночас, це забезпечує більшу гнучкість і ефективність у роботі з різноманітними типами елементів веб-сторінки.

Опис класів

1. Інтерфейс CrawlerComponent

```
package org.example.project.composite;

6 usages  2 implementations
public interface CrawlerComponent {
    2 usages  2 implementations
    void crawl() throws Exception;
}
```

Рис. 3: Клас CrawlerComponent

- Призначення:
  Визначає загальний контракт для об'єктів, які беруть участь у краулінгу.
- Особливості:
  - Декларує метод crawl, який реалізують класи-компоненти (URLComposite, URLLeaf).

- Дозволяє реалізувати патерн Композит, забезпечуючи єдиний інтерфейс для складних (груп) і простих (листків) компонентів.

2. Клас URLComposite

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

import java.util.ArrayList;
import java.util.List;

3 usages
public class URLComposite implements CrawlerComponent {
    3 usages
    private final List<CrawlerComponent> children = new ArrayList<>();

    public void add(CrawlerComponent component) { children.add(component); }

    no usages
    public void remove(CrawlerComponent component) { children.remove(component); }

    2 usages
    @Override
    public void crawl() throws Exception {
        for (CrawlerComponent component : children) {
            component.crawl();
        }
    }
}
```

Рис. 4: Клас URLComposite

- Призначення:
  Представляє складений компонент у структурі Композиту, що містить кілька інших компонентів (CrawlerComponent).
- Особливості:
  - Зберігає список дочірніх компонентів (List<CrawlerComponent>).
  - Методи:
    1. add — додає компонент до списку.
    2. remove — видаляє компонент зі списку.
    3. crawl — ітерується через всі дочірні компоненти та викликає їх метод crawl.
- Приклад використання:
  Об'єднує групу веб-сторінок (URLLeaf) або підгрупу сторінок (інший URLComposite) для краулінгу.

3. Клас URLLeaf

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

import org.example.project.template.BasicCrawler;


2 usages
public class URLLeaf implements CrawlerComponent {
    2 usages
    private final String url;
    2 usages
    private final int siteId;
    2 usages
    private final int level;
    2 usages
    private final int documentId;
    2 usages
    private final BasicCrawler crawler;

    1 usage
    public URLLeaf(String url, int siteId, int level, int documentId, BasicCrawler crawler) {
        this.url = url;
        this.siteId = siteId;
        this.level = level;
        this.documentId = documentId;
        this.crawler = crawler;
    }

    2 usages
    @Override
    public void crawl() throws Exception {
        crawler.crawl(url, siteId, level, documentId);
    }
}
```

Рис. 5: Код URLLeaf

- Призначення:
  Представляє листовий компонент у структурі Композиту, що відповідає за краулінг окремої сторінки.
- Особливості:
  - Містить атрибути, необхідні для краулінгу сторінки:
    - url — адреса сторінки.
    - siteId — ідентифікатор сайту.
    - level — рівень вкладеності сторінки.
    - documentId — ідентифікатор документа у базі даних.
    - crawler — об'єкт класу BasicCrawler, який виконує основний краулінг.
  - Метод crawl викликає BasicCrawler.crawl, передаючи параметри сторінки.

- Призначення:
  Забезпечує краулінг окремої веб-сторінки, реалізуючи функціональність із класу BasicCrawler.

4. Клас ParseTask

```java
@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();


        URLComposite rootComposite = new URLComposite();


        for (org.example.project.model.Document document : documentList) {
            int level = document.getLevel();
            if (level < 5) {
                rootComposite.add(new URLLeaf(document.getUrl(), document.getSiteId(),
                        level: level + 1, document.getId(), crawler));
            }
        }

        rootComposite.crawl();
    }
}
```

Рис. 6: Клас ParseTask

- Призначення:
  Організовує процес краулінгу сторінок, використовуючи структуру Композиту та функціонал BasicCrawler.
- Особливості:
  - Отримує список сторінок через метод getAllDocs із DocumentService, викликаний із BasicCrawler.
  - Створює кореневий об'єкт Композиту (URLComposite) і додає нього дочірні компоненти (URLLeaf) для краулінгу сторінок.
  - Обмежує краулінг сторінок до 5-го рівня вкладеності.
  - Використовує планувальник (@Scheduled), щоб виконувати метод parseSite кожні 10 секунд.
  - Викликає crawl для кореневого компонента URLComposite, який організовує краулінг для всіх дочірніх елементів.

Проблема, яку вирішує патерн Composite

Уніфікація роботи з об'єктами, які мають ієрархічну структуру
Патерн Composite дозволяє об'єднати прості та складені об'єкти в єдину ієрархічну структуру, щоб працювати з ними однаково. Це особливо корисно, коли потрібно організувати роботу з деревоподібними структурами, які складаються з окремих елементів (листків) і груп (композитів).

Розглянемо приклад системи краулінгу. У цій системі можуть існувати:

- Індивідуальні сторінки (URLLeaf), які потрібно обробляти окремо.
- Групи сторінок (URLComposite), які об'єднують кілька окремих сторінок або навіть підгруп для спільної обробки.

Без патерну Composite нам довелося б створювати окремі алгоритми для роботи з кожним типом об'єктів. Це ускладнює код і робить його менш гнучким.

Переваги патерну Composite

1. Простота використання:
   Забезпечує єдиний інтерфейс для роботи як із окремими елементами, так і з їх групами.
2. Гнучкість:
   Легко розширювати структуру, додаючи нові елементи або групи.
3. Масштабованість:
   Можна обробляти складні ієрархічні структури без збільшення складності коду.

Висновок:
У даній лабораторній роботі я реалізувала патерн проєктування "Composite" для уніфікації роботи з об'єктами, що мають ієрархічну структуру. Це дозволило об'єднати прості (листки) та складені (композити) об'єкти в єдину ієрархію, яка може оброблятися однаковим чином. Завдяки цьому патерну було спрощено управління складними структурами даних, що підвищило читабельність і зручність роботи з кодом. Таким чином, використання патерну Composite дозволило побудувати ефективну архітектуру для організації процесу краулінгу. Ця архітектура сприяє спрощенню розширення функціоналу та підтримки системи, роблячи її більш структурованою та гнучкою у використанні.