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 розроблення програмного забезпечення

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

Виконала
студентка групи ІА–22:
Фоменко Альона

Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

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

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 SOLID

1. S: Принцип єдиного обов'язку (SRP)
    - Клас має виконувати лише одну задачу.
    - Зменшує зв'язаність компонентів і підвищує їхню цілісність.
2. O: Принцип відкритості/закритості (OCP)
    - Код має бути відкритим для розширення, але закритим для модифікації.
    - Використовуються інтерфейси або абстрактні класи для розширюваності.
3. L: Принцип підстановки Лісков (LSP)
    - Підкласи повинні замінювати батьківські класи без зміни поведінки програми.
    - Порушення принципу призводить до некоректної роботи з підкласами.
4. I: Принцип розділення інтерфейсу (ISP)
    - Інтерфейси повинні бути вузькоспеціалізованими.
    - Краще кілька дрібних інтерфейсів, ніж один універсальний.
5. D: Принцип інверсії залежностей (DIP)
    - Залежності мають базуватися на абстракціях, а не на конкретних реалізаціях.
    - Абстракції не залежать від деталей, а навпаки.

Шаблон "Abstract Factory"

- Призначення:
  Створює сімейства пов'язаних об'єктів без зазначення їхніх конкретних класів.
- Структура:
    1. AbstractFactory: спільний інтерфейс для фабрик.
    2. ConcreteFactory: реалізує створення конкретних об'єктів.

3. AbstractProduct: спільний інтерфейс для продуктів.
4. ConcreteProduct: конкретні реалізації продуктів.

- Приклад:
  У меблевому магазині "Крісло", "Диван", "Столик" мають спільні інтерфейси й стилі (Ар-деко, Вікторіанський, Модерн).
- Переваги:
  1. Забезпечує сумісність створюваних об'єктів.
  2. Усунення залежності клієнтського коду від конкретних класів.
  3. Реалізує принцип відкритості/закритості.
- Недоліки:
  1. Ускладнює розширення через велику кількість нових класів.

Шаблон "Factory Method"

- Призначення:
  Забезпечує створення об'єктів через інтерфейс, делегуючи реалізацію підкласам.
- Структура:
  1. Creator: оголошує фабричний метод, який повертає об'єкт типу Product.
  2. ConcreteCreator: реалізує фабричний метод, створюючи конкретні об'єкти.
  3. Product: спільний інтерфейс для створюваних об'єктів.
  4. ConcreteProduct: конкретна реалізація продукту.
- Приклад:
  У системі логістики класи "Вантажівка" і "Судно" реалізують інтерфейс "Транспорт". Фабричні методи створюють об'єкти відповідних типів залежно від потреби.
- Переваги:
  1. Усуває залежність коду від конкретних класів.
  2. Легке додавання нових типів продуктів.
  3. Код виробництва зосереджений в одному місці.
- Недоліки:
  1. Створення великих ієрархій класів.

Шаблон «Memento»

Призначення

Шаблон «Memento» дозволяє зберігати та відновлювати стан об'єкта без порушення інкапсуляції. Це корисно для функцій скасування або відкату до попереднього стану.

Структура

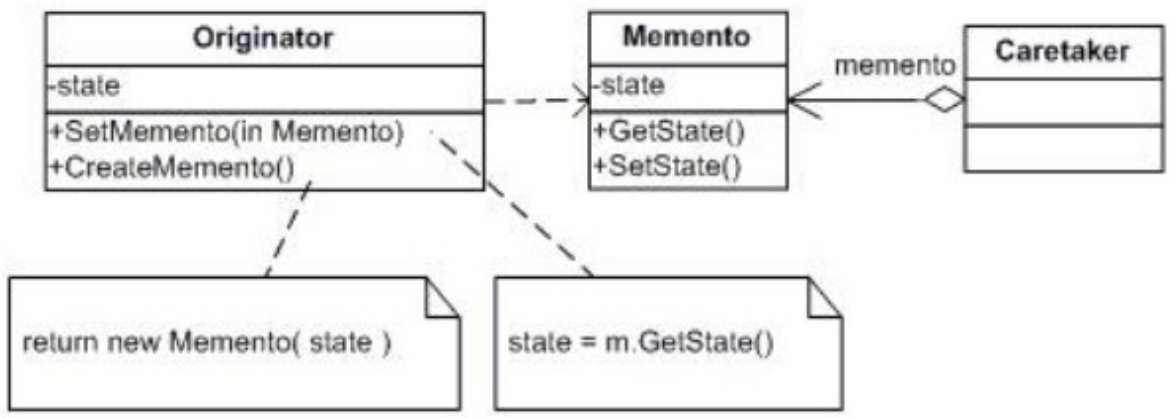

Рис. 1: Структура шаблону «Memento»

1. Originator: Створює та відновлює свій стан за допомогою об'єкта-знімка (Memento).
2. Memento: Зберігає стан Originator. Для інших об'єктів цей клас є закритим.
3. Caretaker: Зберігає знімки і управляє їх передачею назад Originator.

Переваги

- Інкапсуляція стану не порушується.
- Полегшує дизайн класу Originator.

Недоліки

- Великий обсяг пам'яті при частому створенні знімків.
- Можливі витрати на очищення застарілих знімків.

Приклад використання

- Реалізація функцій "Скасування" або "Відкат".
- Транзакції, коли потрібно повернутися до початкового стану.

Шаблон «Observer»

Призначення

Шаблон визначає залежність "один-до-багатьох": коли об'єкт змінює стан, його підписники отримують сповіщення.

Структура

1. Subject (Видавець): Містить стан, сповіщає підписників про зміни.
2. Observer (Підписник): Реагує на зміни стану видавця.

Переваги

- Дозволяє додавати та видаляти підписників під час роботи програми.
- Незалежність між видавцями та підписниками.
- Відповідає принципу відкритості/закритості.

Недоліки

- Підписники сповіщуються в непередбачуваній послідовності.

Приклад використання

- Система сповіщення в банківських системах.
- Реалізація зв’язків у графічних інтерфейсах (WPF, MVVM).

Шаблон «Decorator»

Призначення

Декоратор додає об'єкту нову функціональність без зміни його структури.

Структура

1. Component: Базовий інтерфейс або клас.
2. ConcreteComponent: Початковий клас, що реалізує Component.
3. Decorator: Базовий клас для декораторів.
4. ConcreteDecorator: Реалізує додаткову функціональність.

Переваги

- Дозволяє динамічно додавати функціональність.
- Гнучкість у порівнянні зі спадкуванням.
- Зменшує кількість класів у порівнянні з підкласами для кожної комбінації поведінки.

Недоліки

- Може створити багато дрібних класів.
- Складна конфігурація об'єктів з кількома декораторами.

Приклад використання

- Накладання різних форматувань на текст у текстових редакторах.
- Додавання нових типів сповіщень (SMS, email, Slack) до базового класу Notifier.

Використання шаблонів

- Memento: Для функцій скасування дій або збереження проміжних станів.
- Observer: Для відстеження змін в об'єкті (наприклад, оновлення інтерфейсу після змін даних).
- Decorator: Для динамічного розширення функціональності об'єкта без змін у його базовій реалізації.

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.

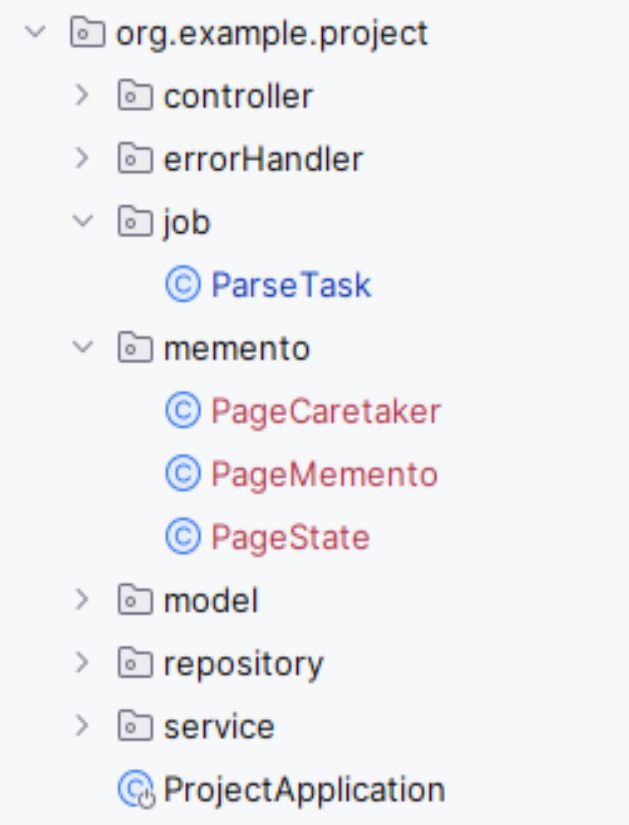

Рис. 2: Структура проекту

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

2. Реалізація шаблону Memento

Паттерн Memento забезпечує механізм збереження та відновлення стану об’єкта без розкриття деталей його реалізації. Він дозволяє створювати "знімки" поточного стану об'єкта, які можна зберегти для подальшого використання або відкату. У нашому проекті парсера цей паттерн можна застосувати для збереження оригінального HTML-контенту та інших важливих даних перед обробкою. Це дозволяє працювати з очищеними даними, а при потребі повертатися до оригінального стану для отримання додаткової інформації.

Таке рішення забезпечує гнучкість і захист від втрати даних, адже, навіть після видалення "зайвого" контенту, можна легко відновити вихідні дані зі збереженого знімка.

1. Клас PageMemento

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

6 usages
public class PageMemento {
    2 usages
    private  String originalHtml;
    2 usages
    private  String title;

    1 usage
    public PageMemento(String originalHtml, String title) {
        this.originalHtml = originalHtml;
        this.title = title;
    }

    1 usage
    public String getOriginalHtml() { return originalHtml; }

    public String getTitle() { return title; }
}
```

Фигура 1: Клас PageMemento

- Зберігає стан сторінки.
- Забезпечує методи доступу до збережених даних (getOriginalHtml, getTitle).

2. Клас PageState

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

3 usages
public class PageState {
    4 usages
    private String currentHtml;
    4 usages
    private String title;

    1 usage
    public void setState(String currentHtml, String title) {
        this.currentHtml = currentHtml;
        this.title = title;
    }

    1 usage
    public PageMemento saveStateToMemento() {
        return new PageMemento(currentHtml, title);
    }

    no usages
    public void restoreStateFromMemento(PageMemento memento) {
        this.currentHtml = memento.getOriginalHtml();
        this.title = memento.getTitle();
    }

    no usages
    public String getCurrentHtml() {
        return currentHtml;
    }

    public String getTitle() {
        return title;
    }
}
```

Фигура 2: Клас PageState

- Відповідає за управління станом сторінки.
- Метод setState(String currentHtml, String title)
    - Встановлює поточний стан сторінки, зберігаючи її HTML і заголовок.
- Метод saveStateToMemento()
    - Створює об'єкт PageMemento, що містить знімок поточного стану.
- Метод restoreStateFromMemento(PageMemento memento)
    - Відновлює стан сторінки з переданого знімка.

3. Клас PageCaretaker

```java
package org.example.project.memento;
import java.util.Stack;

3 usages
public class PageCaretaker {
    3 usages
    private final Stack<PageMemento> history = new Stack<>();

    1 usage
    public void save(PageMemento memento) { history.push(memento); }

    no usages
    public PageMemento undo() {
        if (!history.isEmpty()) {
            return history.pop();
        }
        return null;
    }
}
```

Фигура 3: Клас PageCaretaker

- Зберігає історію знімків станів сторінки у вигляді стеку Stack<PageMemento>.
- Метод save(PageMemento memento)
    - Додає знімок стану до стеку.
- Метод undo()
    - Відновлює попередній стан зі стеку. Якщо історія порожня, повертає null.

4. Клас ParseTask

```java
@Component
public class ParseTask extends ProjectApplication {

    @Autowired
    PictureService pictureService;
    @Autowired
    DocumentService documentService;

    alionafomenko *
    @Scheduled(fixedDelay = 10000)
    public void parseSite() {

        List<org.example.project.model.Document> documentList = documentService.getAllDocs();
        PageCaretaker caretaker = new PageCaretaker();

        for (org.example.project.model.Document document : documentList) {
            // System.out.println("Level------- " + document.getLevel());
            int level = document.getLevel();
```

Фигура 4: Впровадження у ParseTask частина 1

```
        ((HttpStatusErrorHandler) handler1).setNextHandler(handler
        ((TimeoutErrorHandler) handler2).setNextHandler(handler3)

        PageState pageState = new PageState();

        try {
            Document doc = Jsoup.connect(url)
                    .userAgent( s: "AlionaCrawler")
```

Фигура 5: Впровадження у ParseTask частина 2

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


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

Фигура 6: Впровадження у ParseTask частина 3

```
    }   catch (Exception e) {
        handler1.handleError(e);

        PageMemento savedState = caretaker.undo();
        if (savedState != null) {
            pageState.restoreStateFromMemento(savedState);
            documentService.saveContent(document.getId(), pageState.getTitle(), pa
        }
    }
```

Фигура 7: Впровадження у ParseTask частина 4

- Здійснює краулінг веб-сторінок.
- Обробляє елементи сторінок (посилання, зображення, текст).
- Використовує Memento для збереження стану сторінки перед обробкою.
- У разі помилки стан сторінки відновлюється через caretaker.undo() і зберігається з міткою про помилку.

Проблема, яку вирішує патерн "Memento"

Збереження і відновлення стану

Патерн **Memento** дозволяє зберігати поточний стан об'єкта у вигляді знімка, щоб при необхідності повернутися до цього стану. Це особливо корисно, коли потрібно здійснити складну обробку даних, яка включає видалення або зміну початкового стану. Наприклад, у системі краулінгу (як у ParseTask), збереження оригінального HTML-контенту та заголовка сторінки перед їхньою обробкою забезпечує можливість:

- повторно використовувати вихідний стан для додаткових операцій;
- відновлювати дані у разі виникнення помилки.

Це вирішує проблему втрати важливих даних під час роботи над очищенням або трансформацією контенту, дозволяючи системі залишатися гнучкою та безпечною.

Ізоляція збереженого стану від зовнішніх змін

Memento забезпечує ізоляцію збереженого стану від змін, що відбуваються з об'єктом після створення знімка. Наприклад, після збереження вихідного HTML-контенту будь-яка подальша обробка (видалення непотрібних тегів, додавання структурованих даних) не впливає на збережений стан. Завдяки цьому знімки можуть бути використані незалежно від поточного стану об'єкта.

Переваги використання патерну "Memento"

Гнучкість і контроль даних:

- Дозволяє зберігати початковий стан об'єкта перед будь-якими змінами.
- Забезпечує можливість відновлення вихідних даних навіть після складної обробки.

Безпека і локалізація:

- Оригінальний стан зберігається у вигляді окремого об'єкта (Memento), що ізолює його від зовнішніх змін.
- Доступ до стану відбувається лише через визначений API (PageState), що забезпечує інкапсуляцію даних.

Простота інтеграції:

- Патерн легко інтегрувати в існуючу систему.
- Він не вимагає змін у логіці обробки даних і працює поверх основного алгоритму.

Висновок: У даній лабораторній роботі я реалізувала патерн проєктування "Memento" для збереження і відновлення стану об'єктів. Це дозволило створити механізм для збереження стану системи на певний момент часу, що дає змогу здійснювати операції "скасування" змін. Завдяки цьому патерну було збережено інкапсуляцію об'єктів, а також спрощено додавання можливості відновлення попередніх станів без необхідності втручатися в код основних класів. Система стала більш гнучкою, оскільки додавання нових типів відновлення стану не потребує змін в існуючих об'єктах.