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

Лабораторна робота №4

із дисципліни «Технології розроблення програмного забезпечення»
Тема: «Шаблони «Singleton», «Iterator», «Proxy», «State», «Strategy» »

Виконав:
Студент групи ІА-24
Боднар А. Д.

Перевірив:
Мягкий М.Ю.

Київ-2024

Завдання:
1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми

Хід роботи

Теоретичні відомості

1. Singleton (Одинак)

Призначення:

Шаблон **Singleton** гарантує, що клас матиме тільки один екземпляр, і надає глобальну точку доступу до цього екземпляра.

Сфера застосування:

- Коли потрібен єдиний об’єкт для координації дій у системі (наприклад, конфігурація програми або журнал подій).
- Коли обмеження кількості екземплярів класу є важливим.

Реалізація:

- Створення приватного конструктора, щоб запобігти створенню об'єктів за межами класу.
- Використання статичного методу для доступу до єдиного екземпляра.

Приклад використання:

- Менеджер підключення до бази даних.
- Клас, який зберігає глобальні налаштування програми.

2. Iterator (Ітератор)

Призначення:

Шаблон **Iterator** надає спосіб послідовного доступу до елементів колекції без розкриття її внутрішньої структури.

Сфера застосування:

- Коли потрібно надати уніфікований інтерфейс для доступу до елементів різних структур даних (наприклад, списки, дерева).
- Коли потрібно забезпечити можливість одночасної ітерації кількома клієнтами.

Реалізація:

- Створюється інтерфейс Iterator із методами для отримання наступного елемента та перевірки наявності елементів.
- Колекція надає екземпляр ітератора для доступу до своїх елементів.

Приклад використання:

- Обхід списків, масивів, дерев у стандартних бібліотеках.

3. Proxy (Проксі)

Призначення:

Шаблон **Proxy** надає об'єкт-замінник, який контролює доступ до іншого об'єкта.

Сфера застосування:

- Коли потрібен контроль доступу до ресурсу (наприклад, безпека, кешування, мережеві виклики).
- Коли необхідно додати додаткову логіку перед викликом методів основного об’єкта.

Типи Proxy:

- **Віртуальний проксі**: відкладає ініціалізацію дорогого об'єкта.
- **Захисний проксі**: контролює права доступу до об'єкта.
- **Мережевий проксі**: забезпечує доступ до об'єкта, який знаходиться в іншій системі.

Приклад використання:

- Управління доступом до бази даних або файлів.
- Заміна реальних об’єктів у тестуванні (мок-об’єкти).

4. State (Стан)

Призначення:

Шаблон **State** дозволяє об'єкту змінювати свою поведінку залежно від свого стану.

Сфера застосування:

- Коли поведінка об'єкта залежить від його стану, і цей стан змінюється під час виконання програми.
- Коли потрібно уникнути множини умов if-else для визначення поведінки.

Реалізація:

- Створюється базовий інтерфейс State із методами, які залежать від стану.
- Конкретні стани реалізують цей інтерфейс.
- Контекст містить посилання на поточний стан і делегує йому обробку.

Приклад використання:

- Автомат торгових апаратів (купівля товарів залежить від стану: готовий до продажу, очікує монети, видає решту).
- Зміна вигляду кнопок залежно від стану програми.

5. Strategy (Стратегія)

Призначення:

Шаблон **Strategy** визначає набір алгоритмів, інкапсулює кожен із них і робить їх взаємозамінними.

Сфера застосування:

- Коли потрібно динамічно змінювати алгоритм під час виконання програми.
- Коли потрібно уникнути дублювання коду через схожі алгоритми.

Реалізація:

- Створюється інтерфейс стратегії з методом для виконання.
- Конкретні стратегії реалізують різні алгоритми.
- Контекст використовує стратегії для виконання.

Приклад використання:

- Сортування (різні алгоритми сортування для різних даних).
- Розрахунок вартості доставки (різні методи залежно від типу доставки).

Обрано шаблон Strategy

```java
package navigation;

public interface NavigationProcessor {  3 usages  2 implemen
    Object navigate(int pageNumber, Integer lineNumber);
}
```

Рис.1 - Інтерфейс NavigationProcessor

Створюю інтерфейс, що передбачає реалізацію процесу обробки запиту на перехід.

```java
package navigation;

import models.Page;
import repository.PageRepository;

public class LineNavigation implements NavigationProcessor {  1 usa
    private PageRepository repository;  2 usages

    public LineNavigation(PageRepository repository) {  1 usage
        this.repository = repository;
    }

    @Override  1 usage
    public Object navigate(int pageNumber, Integer lineNumber) {
        Page page = repository.findByPageNumber(pageNumber);
        if (page != null) {
            String line = page.getLine(lineNumber);
            if (line != null) {
                return line; // Повертає текст рядка
            }
        }
        return null; // Якщо сторінку або рядок не знайдено
    }
}
```

Рис.2 - Клас LineNavigation що імплементує NavigationProcessor

Перша стратегія, в якій ми переходимо до рядка

```java
package navigation;

import models.Page;
import repository.PageRepository;

public class PageNavigation implements NavigationProcessor {  1 us
    private PageRepository repository;  2 usages

    public PageNavigation(PageRepository repository) {  1 usage
        this.repository = repository;
    }

    @Override  1 usage
    public Object navigate(int pageNumber, Integer lineNumber) {
        Page page = repository.findByPageNumber(pageNumber);
        if (page != null) {
            return page; // Повертає сторінку, якщо знайдено
        }
        return null; // Якщо сторінку не знайдено
    }
}
```

Рис.3 - Клас PageNavigation що імплементує NavigationProcessor

Друга стратегія, в якій ми переходимо до сторінки

```java
package navigation;

import repository.PageRepository;

public class NavigationService {  no usages
    private PageRepository repository;  3 usages

    public NavigationService(PageRepository repository) {  no usag
        this.repository = repository;
    }

    public Object navigate(int pageNumber, Integer lineNumber) {
        NavigationProcessor processor;

        if (lineNumber == null) {
            processor = new PageNavigation(repository);
        } else {
            processor = new LineNavigation(repository);
        }

        return processor.navigate(pageNumber, lineNumber);
    }
}
```

Рис.4 - Клас-сервіс NavigationService

Тут ми бачимо створений сервіс, де реалізований метод переходу і саме в цьому методі ми використовуємо шаблон Strategy, обираючи відповідний спосіб переходу залежно від того чи наданий рядок чи ні

Висновки

У ході виконання лабораторної роботи було вивчено та реалізовано шаблон проектування **"Стратегія"**, що є одним із ключових поведінкових шаблонів. Завдяки цьому шаблону вдалося розділити алгоритми для різних операцій та інкапсулювати їх у окремі класи, забезпечивши гнучкість і розширюваність системи.

1. **Теоретичне опрацювання:**
    - Було проаналізовано принцип роботи шаблону "Стратегія", його призначення та переваги у розробці програмного забезпечення.
    - Вивчено механізм динамічного вибору алгоритмів під час виконання програми, що дозволяє зменшити дублювання коду та спростити структуру програми.
2. **Практична реалізація:**
    - Реалізовано шаблон "Стратегія" для двох основних функцій текстового редактора:

        Перехід до сторінок і рядків:

        - Реалізовано стратегії для навігації за номером сторінки або за конкретним рядком.
        - Впроваджено механізм обробки помилок (якщо сторінка або рядок не знайдені).