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 №6

із дисципліни «*Технології розроблення програмного забезпечення*»
Тема: «Шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»**»**

Виконав:
Студент групи ІА-24
Котлярчук М. С.

Перевірив:
Мягкий М. Ю.

Київ-2024

ЗМІСТ

Мета: дізнатися, вивчити та навчитися використовувати шаблони проєктування abstract factory, factory method, memento, observer, decorator.

Тема: Powershell terminal (strategy, command, abstract factory, bridge, interpreter, client-server).
Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна).

Короткі теоретичні відомості

Abstract Factory: Шаблон, який забезпечує створення сімейств пов’язаних об’єктів без вказівки їхніх конкретних класів. Він інкапсулює логіку створення об’єктів, що належать до одного сімейства, і гарантує узгодженість між ними. Цей підхід спрощує заміну наборів об’єктів і дозволяє легко інтегрувати нові варіанти реалізацій, не змінюючи клієнтський код. Ключовими елементами є інтерфейс фабрики та конкретні фабрики для кожного сімейства об'єктів.

Factory Method: Шаблон, що надає спосіб делегування створення об’єктів підкласам. Базовий клас визначає метод, який повертає об’єкт певного типу, але саме підкласи вирішують, який конкретний об’єкт створювати. Це забезпечує гнучкість і спрощує підтримку коду, оскільки можна легко додавати нові класи без змін у базовому коді. Factory Method підходить для роботи з об'єктами, які потребують складної ініціалізації або належать до ієрархії класів.

Memento: Шаблон, що дозволяє зберігати стан об'єкта в певний момент часу і відновлювати його за потреби без порушення інкапсуляції. Memento складається з трьох компонентів: об'єкта-відправника, який створює та використовує збережений стан, об'єкта-хранителя (memento), що містить збережені дані, і об'єкта-керуючого, який управляє memento, забезпечуючи зберігання та відновлення стану. Цей шаблон широко застосовується в реалізації функцій "Скасувати" та "Повторити" у додатках.

Observer: Шаблон, який встановлює залежність типу "один до багатьох" між об'єктами, забезпечуючи автоматичне сповіщення та оновлення залежних об'єктів при зміні стану об'єкта-спостерігача. Основними компонентами є спостерігач (Observer), який реагує на оновлення, і суб'єкт (Subject), що управляє підписниками та повідомляє їх про зміни. Цей шаблон сприяє слабкому зв'язуванню між об'єктами і широко використовується в системах з динамічними даними.

Decorator: Шаблон, який дозволяє динамічно розширювати функціональність об'єкта шляхом "обгортання" його в інші об'єкти, що реалізують додаткові

можливості. Decorator працює через композицію, а не спадкування, що дозволяє створювати необмежену кількість комбінацій поведінок без модифікації базового класу. Основні елементи включають компонент (інтерфейс або базовий клас), конкретний компонент (базова реалізація) і декоратор, що додає нову поведінку. Цей підхід особливо корисний для додавання функціоналу в складних системах із багатьма варіантами поведінки.

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

Шаблон Abstract Factory

Для цієї лабораторної роботи я обрав реалізувати шаблон Abstract Factory. Реалізація цього шаблону дозволяє створювати сімейства взаємопов’язаних об’єктів без вказівки їхніх конкретних класів. Це забезпечує узгодженість між об'єктами, спрощує модифікацію та заміну компонентів і робить архітектуру програми більш гнучкою. У межах лабораторної роботи я реалізував фабрику для створення елементів теми, таких як колір тексту, колір фону та розмір шрифту. Нижче наведено ключові класи, що використовуються у реалізації шаблону Abstract Factory:

```
public interface ThemeFactory {
    TextColor createTextColor();
    BackgroundColor createBackgroundColor();
    TextSize createTextSize();
}
```

Рис. 1 – Інтерфейс ThemeFactory

Інтерфейс фабрики визначає набір методів для створення об’єктів, пов’язаних із темою. У цьому випадку він включає методи createTextColor(), createBackgroundColor() та createTextSize() для створення відповідних компонентів теми.

```
public class LightThemeFactory implements ThemeFactory {

    public TextColor createTextColor() {
        return new LightTextColor();
    }

    public BackgroundColor createBackgroundColor() {
        return new LightBackgroundColor();
    }

    public TextSize createTextSize() {
        return new LightTextSize();
    }
}
```

Рис. 2 – Реалізація світлої теми (LightThemeFactory)

Фабрика, яка реалізує методи інтерфейсу ThemeFactory для створення компонентів світлої теми. Вона генерує об’єкти, такі як чорний текст, білий фон і розмір шрифту 14. Ця реалізація забезпечує узгодженість компонентів світлої теми.

```
public class DarkThemeFactory implements ThemeFactory {

    public TextColor createTextColor() {
        return new DarkTextColor();
    }

    public BackgroundColor createBackgroundColor() {
        return new DarkBackgroundColor();
    }

    public TextSize createTextSize() {
        return new DarkTextSize();
    }
}
```

Рис. 3 – Реалізація темної теми (DarkThemeFactory)

Фабрика, яка створює компоненти темної теми. Вона включає об'єкти білого тексту, чорного фону і розміру шрифту 16.

```java
public interface TextColor {
    String getTextColor();
}

public interface BackgroundColor {
    String getColor();
}

public interface TextSize {
    int getTextSize();
}
```

Рис. 4 – Інтерфейси продуктів (TextColor, BackgroundColor, TextSize)

Кожен інтерфейс визначає поведінку компонентів. TextColor – отримує колір тексту через метод getTextColor(). BackgroundColor – отримує колір фону через метод getColor(). TextSize – визначає розмір шрифту через метод getTextSize().

```java
public class LightTextColor implements TextColor {

    public String getTextColor() {
        return "black";
    }
}

public class LightBackgroundColor implements BackgroundColor {

    public String getColor() {
        return "white";
    }
}

public class LightTextSize implements TextSize {

    public int getTextSize() {
        return 14;
    }
}
```

Рис. 5 – Реалізація конкретних продуктів для світлої теми

Компоненти LightTextColor, LightBackgroundColor та LightTextSize реалізують інтерфейси продуктів для світлої теми. Наприклад, LightTextColor задає чорний текст, а LightBackgroundColor – білий фон.

```
public class DarkTextColor implements TextColor {

    public String getTextColor() {
        return "white";
    }
}

public class DarkBackgroundColor implements BackgroundColor {

    public String getColor() {
        return "black";
    }
}

public class DarkTextSize implements TextSize {

    public int getTextSize() {
        return 16;
    }
}
```

Рис. 6 – Реалізація конкретних продуктів для темної теми

Компоненти DarkTextColor, DarkBackgroundColor та DarkTextSize відповідають за темну тему. Вони реалізують білий текст, чорний фон і більший розмір шрифту, відповідно.

```java
@Service
public class TabService {

    private final TabRepository tabRepository;

    public TabService(TabRepository tabRepository) {
        this.tabRepository = tabRepository;
    }

    public void applyTheme(Long tabId, ThemeFactory themeFactory) {
        TextColor textColor = themeFactory.createTextColor();
        BackgroundColor backgroundColor = themeFactory.createBackgroundColor();
        TextSize textSize = themeFactory.createTextSize();

        changeTextColor(tabId, textColor.getTextColor());
        changeBackgroundColor(tabId, backgroundColor.getColor());
        changeTextSize(tabId, textSize.getTextSize());

        System.out.println("Applied theme to tab " + tabId + ": " +
                "Text color = " + textColor.getTextColor() +
                ", Background color = " + backgroundColor.getColor() +
                ", Font size = " + textSize.getTextSize());
    }
```

Рис. 7 – Реалізація сервісу для застосування тем (TabService)

TabService є основним отримувачем у шаблоні. Він використовує фабрику ThemeFactory для створення об'єктів теми та застосовує їх до вкладок через метод applyTheme. Сервіс інкапсулює бізнес-логіку зміни кольору тексту, фону і розміру шрифту, роблячи процес зміни теми простим і масштабованим.

```java
public class TabController {

    private final TabService tabService;
    private final CommandInvoker commandInvoker;


    public TabController(TabService tabService) {
        this.tabService = tabService;
        this.commandInvoker = new CommandInvoker();
    }

    @PostMapping("/change-theme")
    public ResponseEntity<String> changeTheme(@RequestBody Map<String, Object> request) {
        Long tabId = Long.valueOf(request.get("tabId").toString());
        String theme = request.get("theme").toString();

        ThemeFactory themeFactory;
        if ("light".equalsIgnoreCase(theme)) {
            themeFactory = new LightThemeFactory();
        } else if ("dark".equalsIgnoreCase(theme)) {
            themeFactory = new DarkThemeFactory();
        } else {
            return ResponseEntity.badRequest().body("Invalid theme: " + theme);
        }
        tabService.applyTheme(tabId, themeFactory);
        return ResponseEntity.ok( body: "Theme changed to: " + theme + " for tab " + tabId);
    }
```

Рис. 8 – Реалізація контролера для управління темами (TabController)

Контролер виступає посередником між користувачем і бізнес-логікою. Він приймає запити, визначає тип теми (світла чи темна), ініціалізує відповідну фабрику та викликає метод applyTheme сервісу.

Висновки Під час виконання даної лабораторної роботи я дізнався та вивчив шаблони abstract factory, factory method, memento, observer, decorator. Я реалізував шаблон Abstract Factory для створення сімейств взаємопов’язаних об’єктів, таких як колір тексту, колір фону та розмір шрифту. Реалізація цього шаблону дозволила забезпечити узгодженість між компонентами теми та спростила процес зміни тем в системі.

Застосування Abstract Factory зробило код більш гнучким, оскільки тепер додавання нової теми можливо без внесення змін у клієнтський код. Всі залежності інкапсульовано в окремих фабриках, що забезпечує простоту розширення функціональності та зменшує ризик виникнення помилок при внесенні змін.