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 №7

із дисципліни «*Технології розроблення програмного забезпечення*»
Тема: «Шаблони «Mediator», «Facade», «Bridge», «Template method»**»**

Виконав:
Студент групи ІА-33
Панащук Р. А.

Перевірив:
Мягкий М. Ю.

Київ-2025

ЗМІСТ

Мета: дізнатися, вивчити та навчитися використовувати шаблони проєктування mediator, facade, bridge, template method.

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

Mediator: Шаблон, який організовує взаємодію між об'єктами, централізуючи їхню комунікацію через окремий об'єкт-посередник. Це дозволяє зменшити кількість прямих залежностей між компонентами системи, сприяючи слабкому зв'язуванню. Основні елементи: посередник (Mediator), який координує взаємодію, і колеги (Colleagues), що взаємодіють лише через посередника. Використовується для побудови складних систем, де важливо спростити взаємозв'язки між модулями.

Facade: Шаблон, який забезпечує спрощений інтерфейс до складної підсистеми, приховуючи її внутрішню реалізацію. Facade дозволяє знизити складність використання підсистеми, концентруючи виклики в одному місці. Основні елементи включають фасад (Facade), який надає спрощений інтерфейс, і підсистеми, що виконують реальну роботу. Це покращує зручність використання і розділення відповідальностей.

Bridge: Шаблон, який розділяє абстракцію і реалізацію, дозволяючи їх незалежно змінювати. Абстракція визначає високорівневий інтерфейс, а реалізація містить деталі виконання, які підключаються через композицію. Основні елементи: абстракція (Abstraction), конкретна абстракція (Refined

Abstraction), реалізація (Implementation) і конкретна реалізація (Concrete Implementation). Це знижує зв'язність між класами та полегшує підтримку системи.

Template Method: Шаблон, який визначає скелет алгоритму в базовому класі, делегуючи реалізацію деяких кроків підкласам. Це забезпечує повторне використання загальної логіки і дозволяє варіювати поведінку в залежності від потреби. Основні елементи: шаблонний метод, що визначає алгоритм, і методи, які можуть бути перевизначені в підкласах. Використовується для забезпечення гнучкості при дотриманні загальної структури.

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 Bridge

Для цієї лабораторної роботи я обрав реалізувати шаблон Bridge. Реалізація цього шаблону дозволяє розділити абстракцію та реалізацію, що забезпечує можливість незалежно змінювати обидва аспекти. Це корисно для створення гнучкої архітектури, яка підтримує розширення функціональності без зміни основного коду. Шаблон Bridge допомагає усунути жорстку залежність між абстракціями (наприклад, налаштування терміналу) та конкретними реалізаціями (такими як PowerShell або CommandLine).У межах лабораторної роботи я реалізував Bridge, де абстракція - визначає логіку налаштування терміналу через класи BasicTerminalSettings і AdvancedTerminalSettings, реалізація - представлена класами, які реалізують інтерфейс Terminal, такими як CommandLine і Powershell.

```
public interface Terminal {
    void setBackgroundColor(String color);
    void setFont(String font, int size);
    void setSyntaxHighlighting(String color);
    void renderTab(String title);
}
```

Рис. 1 – Інтерфейс Terminal

Цей інтерфейс визначає базові методи, які можуть бути реалізовані будь-яким терміналом: setBackgroundColor(String color) - змінює колір фону, setFont(String font, int size) - встановлює шрифт і його розмір, setSyntaxHighlighting(String color) - встановлює колір для підсвічування синтаксису, renderTab(String title): відображає вкладку.

```
public class BasicTerminalSettings {
    private Terminal terminal;

    public BasicTerminalSettings(Terminal terminal) {
        this.terminal = terminal;
    }

    public void applyDefaultSettings() {
        terminal.setBackgroundColor("white");
        terminal.setFont( font: "Consolas",  size: 12);
        terminal.setSyntaxHighlighting("black");
    }
}
```

Рис. 2 – Клас BasicTerminalSettings

Цей клас представляє базову абстракцію для налаштувань терміналу: атрибути - зберігає посилання на об'єкт типу Terminal. Методи - applyDefaultSettings() - застосовує стандартні налаштування (білий фон, шрифт Consolas розміром 12, чорне підсвічування).

```
public class AdvancedTerminalSettings extends BasicTerminalSettings {
    private Terminal terminal;

    public AdvancedTerminalSettings(Terminal terminal) {
        super(terminal);
    }

    public void applyCustomSettings(String backgroundColor, String font, int fontSize,
                                    String syntaxColor) {
        terminal.setBackgroundColor(backgroundColor);
        terminal.setFont(font, fontSize);
        terminal.setSyntaxHighlighting(syntaxColor);
        terminal.renderTab( title: "Custom Tab");
    }
}
```

Рис. 3 – Клас AdvancedTerminalSettings

Цей клас розширює BasicTerminalSettings і додає можливість налаштування терміналу за допомогою параметрів. Метод applyCustomSettings(String

backgroundColor, String font, int fontSize, String syntaxColor) дозволяє встановити індивідуальні параметри терміналу. Ми можемо розширювати клас налаштувань не змінюючи код терміналів.

```
public class Powershell implements Terminal {
    private TabService tabService;
    public Powershell(TabService tabService) {
        this.tabService = tabService;
    }
    public void setBackgroundColor(String color) {
        tabService.setBackgroundColor(color);
        System.out.println("PowerShell: Setting background color to " + color);
    }
    public void setFont(String font, int size) {
        tabService.setFontAndSize(font, size);
        System.out.println("PowerShell: Setting font to " + font + " with size " + size);
    }
    public void setSyntaxHighlighting(String color) {
        tabService.setSyntaxColor(color);
        System.out.println("PowerShell: Setting syntax highlighting to " + color);
    }
    public void renderTab(String title) {
        tabService.render(title);
        System.out.println("PowerShell: Rendering tab with title " + title);
    }
}
```

Рис. 4 – Клас Powershell

```
public class CommandLine implements Terminal {
    private TabService tabService;
    public CommandLine(TabService tabService) {
        this.tabService = tabService;
    }
    public void setBackgroundColor(String color) {
        tabService.setBackgroundColor(color);
        System.out.println("CLI: Setting background color to " + color);
    }
    public void setFont(String font, int size) {
        tabService.setFontAndSize(font, size);
        System.out.println("CLI: Setting font to " + font + " with size " + size);
    }
    public void setSyntaxHighlighting(String color) {
        tabService.setSyntaxColor(color);
        System.out.println("CLI: Setting syntax highlighting to " + color);
    }
    public void renderTab(String title) {
        tabService.render(title);
        System.out.println("CLI: Rendering tab with title " + title);
    }
}
```

Рис. 5 – Клас CommandLine

Клас Powershell, код якого зображено на рисунку 4, та клас CommandLine, код якого зображено на рисунку 5, реалізують інтерфейс Terminal для роботи

терміналів. Вони мають спільний інтерфейс, тому вони можуть працювати з різними налаштуваннями.

```
@PostMapping("/apply-settings")
public ResponseEntity<String> applySettings(@RequestBody Map<String, Object> request) {
    Long tabId = Long.valueOf(request.get("tabId").toString());
    String mode = request.get("mode").toString();
    Tab tab = tabService.findTabById(tabId);
    Terminal terminal = new Powershell(tabService);
    if ("basic".equalsIgnoreCase(mode)) {
        BasicTerminalSettings basicSettings = new BasicTerminalSettings(terminal);
        basicSettings.applyDefaultSettings();
        return ResponseEntity.ok( body: "Basic settings applied to Tab ID: " + tabId);
    } else if ("advanced".equalsIgnoreCase(mode)) {
        String backgroundColor = request.getOrDefault( key: "backgroundColor",  defaultValue: "blue").toString();
        String font = request.getOrDefault( key: "font",  defaultValue: "Arial").toString();
        int fontSize = Integer.parseInt(request.getOrDefault( key: "fontSize",  defaultValue: "14").toString());
        String syntaxColor = request.getOrDefault( key: "syntaxColor",  defaultValue: "green").toString();
        AdvancedTerminalSettings advancedSettings = new AdvancedTerminalSettings(terminal);
        advancedSettings.applyCustomSettings(backgroundColor, font, fontSize, syntaxColor);
        return ResponseEntity.ok( body: "Advanced settings applied to Tab ID: " + tabId);
    } else {
        return ResponseEntity.badRequest().body("Invalid mode. Use 'basic' or 'advanced'.");
    }
}
```

Рис. 5 – Приклад використання патерну Bridge

У запиті клієнт вказує tabId і mode (basic або advanced). Для розширеного режиму можна додати додаткові параметри: backgroundColor, font, fontSize, syntaxColor.

Висновки: Під час виконання даної лабораторної роботи я дізнався, вивчив та навчився використовувати шаблони проєктування Mediator, Facade, Bridge та Template Method. Я реалізував шаблон Bridge для розділення абстракції та реалізації в системі налаштувань терміналу. Реалізація цього шаблону дозволила забезпечити незалежність між логікою налаштувань і конкретними реалізаціями терміналів, такими як PowerShell та CommandLine.

Застосування Bridge зробило архітектуру більш гнучкою, дозволивши легко додавати нові типи терміналів або розширювати функціонал налаштувань без змін у клієнтському коді. Абстрагування залежностей спростило тестування та зменшило ризик помилок при внесенні змін.