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 №8

із дисципліни «*Технології розроблення програмного забезпечення*»

Тема: «Шаблони «Composite», «Flyweight», «Interpreter», «Visitor»**»**

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

Перевірив:
Мягкий М. Ю.

Київ-2025

ЗМІСТ

Мета: дізнатися, вивчити та навчитися використовувати шаблони проєктування composite, flyweight, interpreter, visitor.

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

Composite: Шаблон, що дозволяє працювати з групою об'єктів так само, як і з окремими об'єктами. Він організовує об'єкти в деревоподібну структуру, де кожен вузол може бути як складеним об'єктом (компонентом, що містить інші компоненти), так і простим об'єктом. Composite спрощує роботу з ієрархіями та забезпечує гнучке додавання нових типів компонентів. Основні елементи включають базовий інтерфейс (Component), вузли (Composite), які можуть містити інші компоненти, і листки (Leaf), які є кінцевими об'єктами.

Flyweight: Шаблон, що оптимізує використання пам'яті шляхом зберігання спільних даних в одному місці. Flyweight розділяє стан об'єкта на внутрішній (унікальний для класу) і зовнішній (залежний від контексту). Завдяки цьому створюється велика кількість об'єктів із мінімальними витратами пам'яті. Основними компонентами є Flyweight (інтерфейс для об'єктів-легковаговиків), ConcreteFlyweight (конкретні реалізації), FlyweightFactory (фабрика для створення/збереження об'єктів) і клієнт, який передає зовнішній стан.

Interpreter: Шаблон, що визначає спосіб обробки та виконання граматики мови. Він забезпечує інтерпретацію простих конструкцій через ієрархію класів, які відповідають правилам граматики. Interpreter корисний для побудови парсерів

або виконання скриптів. Основні компоненти включають AbstractExpression (загальний інтерфейс), TerminalExpression (для кінцевих правил), NonTerminalExpression (для комбінованих правил) і Context (дані або змінні, необхідні для обчислень).

Visitor: Шаблон, що дозволяє додавати нову поведінку до класів без їхньої модифікації. Visitor відокремлює алгоритми від структури об'єктів, забезпечуючи зручність розширення. Основними компонентами є Visitor (інтерфейс або абстрактний клас для реалізації нових операцій), ConcreteVisitor (конкретні реалізації операцій), Element (інтерфейс для класів, що приймають відвідувачів), ConcreteElement (конкретні елементи, що приймають відвідувача), і метод accept для взаємодії між елементами та відвідувачами.

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 Interpreter

У цій лабораторній роботі я обрав реалізацію патерну "Інтерпретатор" (Interpreter), оскільки він дозволяє визначити граматику для мови команд та реалізувати обробку виразів за допомогою конкретних класів. Це дозволяє зручно інтерпретувати введені команди та реалізувати їхню логіку, не переписуючи код для кожного типу команди. У рамках лабораторної роботи потрібно обробляти різні типи команд, зокрема зміни директорії та застосування тем, що є гарним прикладом для застосування цього патерну.

```
public interface Expression {
    String interpret(String context, Tab tab);
}
```

Рис. 1 – Інтерфейс Expression

Це інтерфейс для всіх конкретних виразів у патерні. Він містить метод interpret(String context, Tab tab), який визначає, як обробляти команду в контексті

вкладки. У класах, що реалізують цей інтерфейс, буде реалізована специфічна логіка інтерпретації команд.

```
public class ApplyThemeExpression implements Expression {
    private final String theme;
    private TabService tabService;

    public ApplyThemeExpression(String theme, Long tabId, TabService tabService) {
        this.theme = theme.toLowerCase();
        this.tabService = tabService;
    }

    public String interpret(String context, Tab tab) {
        if (context.startsWith("applyTheme")) {
            String[] parts = context.split( regex: "\\s+");
            if (parts.length == 2) {
                String themeName = parts[1].replace( target: "'", replacement: "").toLowerCase();
                ThemeFactory themeFactory = createThemeFactory(themeName);
                if (themeFactory != null) {
                    tabService.applyTheme(tab.getId(), themeFactory);
                    return "Applying theme: " + themeName;
                }
            }
        }
        return "Unknown command";
    }

    private ThemeFactory createThemeFactory(String themeName) {
        switch (themeName) {
            case "dark":
                return new DarkThemeFactory();
            case "light":
                return new LightThemeFactory();
            default:
                System.out.println("Unknown theme: " + themeName);
                return null;
        }
    }
}
```

Рис. 2 – Клас ApplyThemeExpression

Цей клас реалізує інтерфейс Expression і відповідає за інтерпретацію команд, які стосуються застосування теми до вкладки. Він отримує команду типу "applyTheme" і викликає метод для застосування відповідної теми. У методі interpret перевіряється, чи є в команді ключове слово "applyTheme", потім

викликається фабрика тем, яка створює об'єкти для теми, зокрема DarkThemeFactory або LightThemeFactory.

```
public class CommandTypeExpression implements Expression {

    private ContextStrategy contextStrategy;
    private DirectoryCommandHandler directoryCommandHandler;

    public CommandTypeExpression(ContextStrategy contextStrategy, TabService tabService) {
        this.contextStrategy = contextStrategy;
        this.directoryCommandHandler = new DirectoryCommandHandler(tabService);
    }

    public String interpret(String context, Tab tab) {
        String cleanedCommand = context.trim();

        if (cleanedCommand.toLowerCase().startsWith("cd") ||
                cleanedCommand.toLowerCase().startsWith("set-location") ||
                cleanedCommand.matches( regex: "^[a-zA-Z]:$") ||
                new File(cleanedCommand).exists()) {
            return directoryCommandHandler.handleDirectoryCommand(cleanedCommand, tab);
        }

        if (cleanedCommand.endsWith(".exe") || cleanedCommand.endsWith(".bat") || cleanedCommand.endsWith(".ps1")) {
            System.out.println("Command is an executable file: " + context);
            contextStrategy.setStrategy(new FileExecutionStrategy());
        } else {
            System.out.println("Command is assumed to be a PowerShell command: " + context);
            contextStrategy.setStrategy(new CommandExecutionStrategy());
        }

        return contextStrategy.executeStrategy(cleanedCommand, tab.getCurrentDirectory());
    }
}
```

Рис. 3 – Клас CommandTypeExpression Цей клас також реалізує інтерфейс Expression і відповідає за інтерпретацію команд, що стосуються зміни директорії або виконання файлів. В залежності від типу команди (директорія чи файл), викликається відповідний обробник або стратегія виконання. Клас взаємодіє з обробником команд директорії (DirectoryCommandHandler) і стратегіями виконання команд або файлів.

```java
public class DirectoryCommandHandler {

    private final TabService tabService;

    public DirectoryCommandHandler(TabService tabService) {
        this.tabService = tabService;
    }

    public String handleDirectoryCommand(String command, Tab tab) {
        String currentDirectory = tab.getCurrentDirectory();

        if (command.matches( regex: "^[a-zA-Z]:$")) {
            File newDrive = new File( pathname: command + "\\");
            if (newDrive.exists() && newDrive.isDirectory()) {
                tabService.changeDirectory(tab, newDrive.getAbsolutePath());
                return "Changed to drive: " + newDrive.getAbsolutePath();
            } else {
                return "Error: Drive " + command + " does not exist.";
            }
        }

        if (command.toLowerCase().startsWith("cd ") || command.toLowerCase().startsWith("set-location ")) {
            String targetPath = command.substring( beginIndex: command.indexOf(" ") + 1).trim();
            File newDir = resolvePath(targetPath, currentDirectory);

            if (newDir.exists() && newDir.isDirectory()) {
                tabService.changeDirectory(tab, newDir.getAbsolutePath());
                return "Changed directory to: " + newDir.getAbsolutePath();
            } else {
                return "Error: Directory does not exist: " + targetPath;
            }
        }

        File directPath = new File(command);
        if (directPath.exists() && directPath.isDirectory()) {
            tabService.changeDirectory(tab, directPath.getAbsolutePath());
            return "Changed directory to: " + directPath.getAbsolutePath();
        }
        return "Error: Unrecognized or invalid directory command: " + command;
    }

    private static File resolvePath(String path, String currentDirectory) {
        File resolvedPath = new File(path);
        if (!resolvedPath.isAbsolute()) {
            resolvedPath = new File(currentDirectory, path);
        }
        return resolvedPath;
    }
}
```

Рис. 4 – Клас DirectoryCommandHandler

Цей клас обробляє команди для зміни директорії, такі як "cd" або "setlocation". Він перевіряє правильність введеного шляху та змінює поточну директорію вкладки через TabService. Метод handleDirectoryCommand відповідає за обробку таких команд і викликає відповідні методи для зміни директорії.

Висновки: Під час виконання даної лабораторної роботи я вивчив та реалізував шаблони Interpreter, Composite, Flyweight та Visitor. Я реалізував шаблон Interpreter для інтерпретації команд PowerShell терміналу, таких як зміна директорії або застосування теми. Цей патерн дозволив створити структуру, де кожен тип команди має свій клас з чітко визначеною логікою обробки. Застосування шаблону зробило процес додавання нових команд простим і зручним, підвищивши модульність та гнучкість системи.