Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота № 5

З дисципліни «Технології розроблення програмного забезпечення»

Тема: «Шаблони «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Виконала: Лапа Руслана Ігорівна

студент групи ІА-11

Дата здачі

Захищено з балом

Перевірив:

ст. вик. кафедри ІСТ

Колеснік В. М.

Київ 2023

Тема: Шаблони «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE».

Мета: реалізувати один з розглянутих шаблонів.

Хід роботи:

..8 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).

1. Реалізувати не менше 3-х класів відповідно до обраної теми

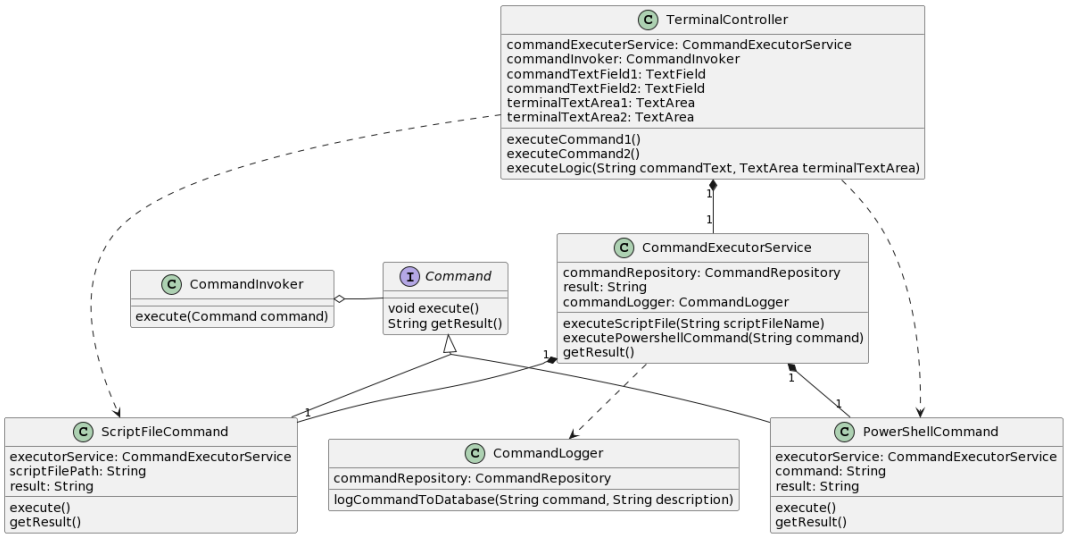

Рис. 1 – Діаграма класів для нижче описаного коду

Інтерфейс:

Створений інтерфейс Command, який містить метод, наприклад, execute(), який представляє дію, що має бути виконана.

Реалізовані конкретні класи команд:

PowershellCommand, ScriptFileCommand. Кожен з цих класів інкапсулює певну дію.

Отримувач:

Отримувач відповідає за виконання команди. У моєму випадку частина мого додатку, яка виконує команди PowerShell і виконує скриптові файли- CommandExecutorService. Також я винесла окремо частину коду в CommandLogger, де відбувалося логування команд в репозиторій, для подальшого завдання- отримання історії виконаних команд.

Інвокер:

Це CommandInvoker, який зберігає та викликає об'єкти команд. Метод execute служить єдиним точковим входом для виконання будь-якої команди. У майбутньому можливо застосовувати додаткову функціональність у єдиному стилі, таку як чергування або функцію скасування.

Клієнт:

Клієнт відповідає за налаштування та асоціацію інвокера з об'єктами команд. У додатку для термінала це місце, де я визначаю, яку команду слід виконати при натисканні кнопки, а саме TerminalController.

2. Реалізація шаблону «Command»

```
package com.example.terminal.Model.Execution;

2 implementations
public interface Command {
    1 usage  2 implementations
    void execute();
    1 usage  2 implementations
    String getResult();
}
```

```java
package com.example.terminal.Model.Execution;

public class ScriptFileCommand implements Command {
    3 usages
    private final CommandExecutorService executorService;
    2 usages
    private final String scriptFilePath;
    2 usages
    private String result;

    1 usage
    public ScriptFileCommand(String scriptFilePath, CommandExecutorService executorService) {
        this.executorService = executorService;
        this.scriptFilePath = scriptFilePath;
    }

    1 usage
    @Override
    public void execute() {
        executorService.executeScriptFile(scriptFilePath);
        result = executorService.getResult();
    }
    1 usage
    @Override
    public String getResult() {
        return result;
    }
}
```

```java
package com.example.terminal.Model.Execution;

public class PowerShellCommand implements Command {
    3 usages
    private final CommandExecutorService executorService;
    2 usages
    private final String command;
    2 usages
    private String result;

    1 usage
    public PowerShellCommand(String command, CommandExecutorService executorService) {
        this.executorService = executorService;
        this.command = command;
    }

    1 usage
    @Override
    public void execute() {
        executorService.executePowershellCommand(command);
        result = executorService.getResult();
    }

    1 usage
    @Override
    public String getResult() { return result; }
}
```

```java
package com.example.terminal.Model.Execution;

import ...

public class CommandExecutorService {
    3 usages
    private final CommandRepository commandRepository;
    7 usages
    private String result;

    1 usage
    public CommandExecutorService(CommandRepository commandRepository) { this.commandRepository = commandRepository; }

    1 usage
    public void executeScriptFile(String scriptFileName) {
        try {
            ProcessBuilder processBuilder = new ProcessBuilder( ...command: "powershell.exe", "-NoProfile",
                    "-ExecutionPolicy", "Bypass", "-Command", scriptFileName);
            Process process = processBuilder.start();
            // Capture the command output
            StringBuilder output = new StringBuilder();
            BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                output.append(line).append("\n");
            }
            // Wait for the process to complete
            process.waitFor();

            // After executing the command, log it to the database
            CommandLogger commandLogger = new CommandLogger(commandRepository);
            commandLogger.logCommandToDatabase(scriptFileName, output.toString());


            // Store the command result
            result = output.toString();
            System.out.println("Script file executed successfully.");
        } catch (IOException | InterruptedException e) {
            result = "Error executing Script file: " + e.getMessage();
            System.err.println(result);
        }
    }

    1 usage
    public void executePowershellCommand(String command) {

        try {
            ProcessBuilder processBuilder = new ProcessBuilder( ...command: "powershell.exe", "-NoProfile",
                    "-ExecutionPolicy", "Bypass", "-Command", command);
            Process process = processBuilder.start();
```

```
            // Collect the output of the command
            BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
            String line;
            StringBuilder output = new StringBuilder();
            while ((line = reader.readLine()) != null) {
                output.append(line).append("\n");
            }

            process.waitFor();

            CommandLogger commandLogger = new CommandLogger(commandRepository);
            commandLogger.logCommandToDatabase(command, output.toString());

            result = output.toString();
            System.out.println("PowerShell command executed successfully.");
        } catch (IOException | InterruptedException e) {
            result = "Error executing PowerShell command: " + e.getMessage();
            System.err.println(result);
        }
    }

    2 usages
    public String getResult() { return result; }
}
```

```
package com.example.terminal.Model.Execution;

import ...

public class CommandLogger {
    2 usages
    private final CommandRepository commandRepository;

    2 usages
    public CommandLogger(CommandRepository commandRepository) {
        this.commandRepository = commandRepository;
    }

    2 usages
    public void logCommandToDatabase(String command, String description) {
        CommandEntry commandEntry = new CommandEntry();
        commandEntry.setCommandText(command);
        commandEntry.setDescription(description);
        commandEntry.setExecutionTime(new Timestamp(System.currentTimeMillis()));
        try {
            commandRepository.insertCommand(commandEntry);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```java
package com.example.terminal.Model.Execution;

public class CommandInvoker {

    1 usage
    public void execute(Command command) {
        command.execute();
    }
}
```

```java
@FXML
private void executeCommand1() {
    String commandText = commandTextField1.getText();

    if (commandText.trim().isEmpty()) {
        return;
    }
    executeLogic(commandText, terminalTextArea1);
}

@FXML
private void executeCommand2() {
    String commandText = commandTextField2.getText();

    if (commandText.trim().isEmpty()) {
        return;
    }
    executeLogic(commandText, terminalTextArea2);
}
```

```java
private void executeLogic(String commandText, TextArea terminalTextArea) {
    Command commandObject;
    String output;
    if (commandText.toLowerCase().endsWith(".ps1")) {
        // This is a PowerShell script file
        commandObject = new ScriptFileCommand(commandText, executorService);
    } else {
        // This is a command
        commandObject = new PowerShellCommand(commandText, executorService);
    }
    commandInvoker.execute(commandObject);
    output = commandObject.getResult();
    terminalTextArea.appendText( s: commandText + "\n" + output);
}
```

sample.fxml

```
<TextField fx:id="commandTextField1" promptText="Enter your command..."/>
<TextArea fx:id="terminalTextArea1" editable="false" wrapText="true"/>
<Button onAction="#executeCommand1" text="Execute"/>
```

Висновок: у цій лабораторній роботі я ознайомилась з різними шаблонами проєктування і реалізувала шаблон “Command” .