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»,

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

Перевірив:
Мягкий М. Ю.

Київ-2025

ЗМІСТ

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

Singleton — це шаблон проектування, який гарантує, що у програмі існує тільки один екземпляр певного класу, і надає глобальну точку доступу до цього екземпляра. Він використовується для керування спільними ресурсами, такими як база даних, файлові системи або конфігураційні файли. Основні елементи Singleton включають приватний конструктор для обмеження створення об'єктів, статичний метод доступу до екземпляра і механізми синхронізації для забезпечення багатопотокової безпеки.

Iterator — це шаблон, який забезпечує послідовний доступ до елементів колекції без розкриття її внутрішньої реалізації. Він дозволяє обійти структури даних, такі як масиви, списки або дерева, використовуючи уніфікований інтерфейс. Цей шаблон корисний у випадках, коли потрібно працювати з колекціями різних типів, не змінюючи їхню реалізацію. Ключовими елементами є об'єкт-колекція, який надає ітератор, і сам ітератор, що виконує обхід.

Proxy — це структурний шаблон, який створює сурогат для іншого об’єкта, дозволяючи контролювати доступ до нього. Проксі-об'єкти часто використовуються для оптимізації ресурсів, кешування, управління правами доступу або роботи з віддаленими об'єктами. Proxy реалізує той самий інтерфейс, що й оригінальний об'єкт, але додає додаткову логіку, наприклад, для виконання перевірок перед викликом основного об'єкта.

State — це поведінковий шаблон, який дозволяє об'єктам змінювати свою поведінку залежно від їхнього поточного стану. Кожен стан реалізується як окремий клас, який визначає специфічну поведінку для цього стану. Контекст об’єкта делегує виконання завдань об’єктам стану, що дозволяє легко додавати нові стани або змінювати логіку без модифікації контексту. Шаблон використовується для автоматів із багатьма станами, наприклад, життєвого циклу документа (чернетка, рецензія, опубліковано).

Strategy — це поведінковий шаблон, який дозволяє визначити сімейство алгоритмів, інкапсулювати їх у окремі класи і робити їх взаємозамінними.

Об’єкт-контекст делегує виконання певної задачі об’єкту-стратегії, що дозволяє легко змінювати або розширювати алгоритми. Цей шаблон використовується в ситуаціях, коли потрібно мати різні підходи до вирішення однієї задачі, наприклад, різні способи сортування або форматування даних.

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 Strategy

Для цієї лабораторної роботи я обрав реалізувати шаблон Strategy. Реалізація цього шаблону полягатиме у створенні різних стратегій виконання команд у терміналі залежно від їх типу. Наприклад, якщо команда є PowerShellскриптом, вона вимагає спеціальної обробки для правильного виконання в середовищі PowerShell. Інший випадок — виконання зовнішніх виконуваних файлів. Цей сценарій потребує іншого підходу. Таким чином, моє завдання полягає у розробці двох стратегій: CommandExecutionStrategy для виконання PowerShell-команд і FileExecutionStrategy для запуску виконуваних файлів. Це дозволить забезпечити розширюваність і гнучкість системи, а також спростить додавання нових типів виконання в майбутньому.

ExecutionStrategy.java

```java
package com.example.terminal_powershell.strategy;

import com.example.terminal_powershell.model.Command;

2 implementations
public interface ExecutionStrategy {
    2 implementations
    String execute(Command command);
}
```

Рис.1 – Інтерфейс ExecutionStrategy

CommandExecutionStrategy.java

```java
public class CommandExecutionStrategy implements ExecutionStrategy {
    @Override
    public String execute(Command command) {
        System.out.println("Executing PowerShell command: " + command.getName());
        try {
            ProcessBuilder processBuilder = new ProcessBuilder("powershell.exe", "-Command", command.getName());
            processBuilder.redirectErrorStream(true);
            Process process = processBuilder.start();

            String output = new BufferedReader(new InputStreamReader(process.getInputStream()))
                    .lines()
                    .collect(Collectors.joining("\n"));

            int exitCode = process.waitFor();
            if (exitCode == 0) {
                return output;
            } else {
                return "Error: Command failed. Output: " + output;
            }
        } catch (Exception e) {
            e.printStackTrace();
            return "Error while executing PowerShell command: " + e.getMessage();
        }
    }
```

Рис. 2 – Клас CommandExecutionStrategy, що виконує Powershell-команди

FileExecutionStrategy.java

```java

public class FileExecutionStrategy implements ExecutionStrategy {
    @Override
    public String execute(Command command) {
        System.out.println("Executing file: " + command.getName());
        try {
            ProcessBuilder processBuilder = new ProcessBuilder(command.getName());
            processBuilder.redirectErrorStream(true);
            Process process = processBuilder.start();

            String output = new BufferedReader(new InputStreamReader(process.getInputStream()))
                    .lines()
                    .collect(Collectors.joining("\n"));

            int exitCode = process.waitFor();
            if (exitCode == 0) {
                return output;
            } else {
                return "Error: Command failed. Output: " + output;
            }
        } catch (Exception e) {
            e.printStackTrace();
            return "Error while executing PowerShell command: " + e.getMessage();
        }
    }
}
```

Рис. 3 – Клас FileExecutionStrategy, що запускає виконувані файли

CommandController.java

```java
import com.example.terminal_powershell.model.Command;
import com.example.terminal_powershell.services.CommandService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/commands")
public class CommandController {
    private final CommandService commandService;

    public CommandController(CommandService commandService) { this.commandService = commandService; }

    public ResponseEntity<String> executeCommand(@PathVariable Long id, @RequestBody Command command) {
        try {
            String result = commandService.executeCommand(command);
            if (result.contains("Error:")) {
                return ResponseEntity.badRequest().body(result);
            }
            return ResponseEntity.ok(result);
        } catch (IllegalArgumentException e) {
            return ResponseEntity.badRequest().body(e.getMessage());
        }
    }
}
```

Рис. 4 – Клас CommandController, що обробляє запити на виконання команд

CommandService.java

```java
package com.example.terminal_powershell.services;

import com.example.terminal_powershell.model.Command;
import com.example.terminal_powershell.strategy.CommandExecutionStrategy;
import com.example.terminal_powershell.strategy.FileExecutionStrategy;
import org.springframework.stereotype.Service;

@Service
public class CommandService {

    public String executeCommand(Command command) {
        if (command.getIsPowerShellCommand()) {
            return new CommandExecutionStrategy().execute(command);
        } else {
            return new FileExecutionStrategy().execute(command);
        }
    }
}
```

Рис. 5 – Клас CommandService, що обирає стратегію для виконання команди

CommandControllerTest.java

```java
class CommandControllerTest {
    private static final Logger logger = LoggerFactory.getLogger(CommandControllerTest.class);
    private final CommandService commandService = new CommandService();
    private final CommandController commandController = new CommandController(commandService);

    @Test
    void testExecutePowerShellCommand() {
        Command command = new Command("echo 'Hello, PowerShell!'", true);
        ResponseEntity<String> response = commandController.executeCommand(1L, command);
        logger.info("Response: {}", response.getBody());
        assertEquals(200, response.getStatusCode().value());
        assertNotNull(response.getBody());
        assertTrue(response.getBody().contains("Hello, PowerShell!"));
    }

    @Test
    void testExecuteInvalidCommand() {
        Command command = new Command("invalid_command", true);
        ResponseEntity<String> response = commandController.executeCommand(2L, command);
        logger.info("Response: {}", response.getBody());
        assertEquals(400, response.getStatusCode().value());
        assertNotNull(response.getBody());
        assertTrue(response.getBody().contains("Error"));
    }


    @Test
    void testExecuteFileCommand() {
        Command command = new Command("notepad.exe", false);
        ResponseEntity<String> response = commandController.executeCommand(1L, command);
        logger.info("Response: {}", response.getBody());
        assertEquals(200, response.getStatusCode().value());
        assertNotNull(response.getBody());
        assertTrue(response.getBody().isEmpty());
    }
}
```

Рис. 6 – Клас CommandControllerTest, що тестує виконання команд та файлів

Висновки: Під час виконання даної лабораторної роботи я дізнався та вивчив шаблони «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY».

Я реалізував шаблон Strategy у своєму проєкті, що дозволило значно покращити структуру коду та зробити його більш гнучким і підтримуваним. Використання цього шаблону допомогло створити різні стратегії для обробки команд та виконуваних файлів, що спрощує процес модифікації та розширення функціоналу.