Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №6

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Шаблони «Abstract Factory»,
«Factory Method», «Memento», «Observer»,
«Decorator»»

Варіант №18

Виконав:
студент групи ІА-24
Гуменюк К. Е.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

Contents

Тема.

Шаблони «Abstract Factory»,«Factory Method», «Memento», «Observer», «Decorator»

Мета.

Метою даної лабораторної роботи є ознайомлення з шаблонами проєктування, зокрема з шаблоном " Factory Method", та їх практичне застосування при розробці програмного забезпечення.

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

Обрана тема.

..18 Shell (total commander) (state, prototype, factory method, template method, interpreter, client-server)

Оболонка повинна вміти виконувати основні дії в системі - перегляд файлів папок в файлової системі, перемикання між дисками, копіювання, видалення, переміщення об'єктів, пошук.

Короткі теоретичні відомості.

Патерн "Фабричний метод" (Factory Method) є породжувальним патерном проектування, який визначає інтерфейс для створення об'єктів, але дозволяє підкласам вирішувати, який клас створювати. Таким чином, патерн "Фабричний метод" дозволяє відкласти інстанціювання до підкласів, забезпечуючи гнучкість і розширюваність системи.

Шаблон "фабричний метод" визначає інтерфейс для створення об'єктів певного базового типу. Це зручно, коли хочеться додати можливість створення об'єктів не базового типу, а деякого дочірнього. Фабричний метод у такому разі є зачіпкою для впровадження власного конструктора об'єктів. Основна ідея полягає саме в заміні об'єктів їх підтипами, що при цьому зберігає ту ж функціональність; інша частина поведінки об'єктів не є інтерфейсною (AnOperation) і дозволяє взаємодіяти із створеними об'єктами як з об'єктами базового типу. Тому шаблон "фабричний метод" носить ще назву "Віртуальний конструктор".

Хід роботи.

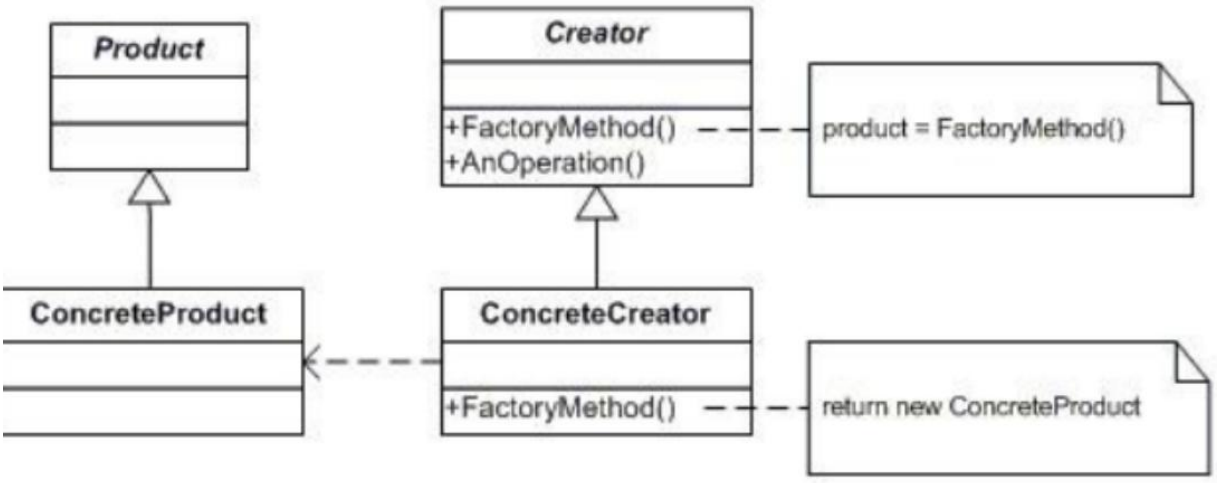

Рисунок 1.1 – UML діаграма шаблону Factory Method

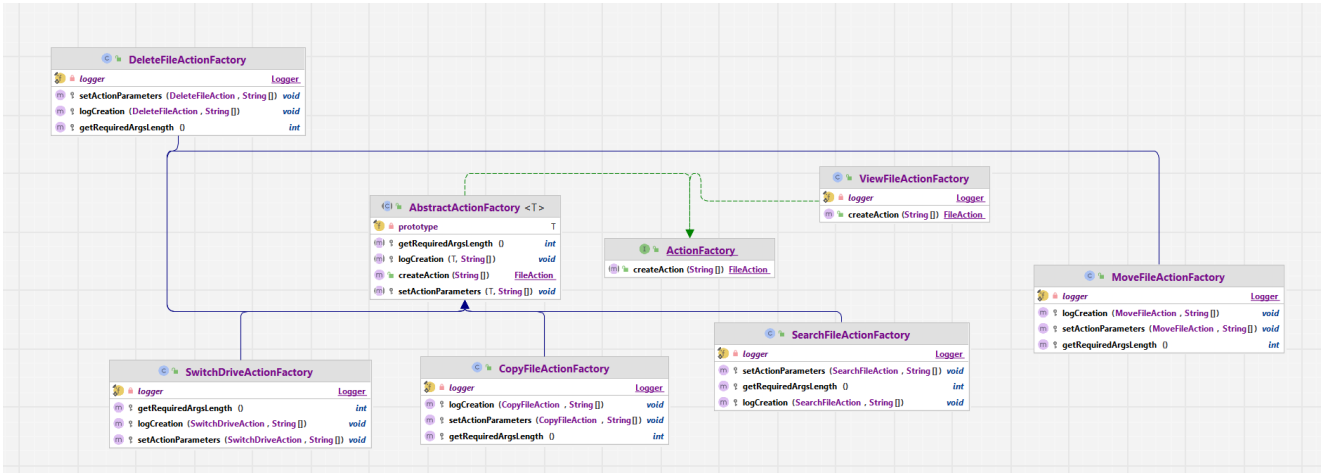

Рисунок 1.2 – Діаграма класів , згенерована IDE

. Інтерфейс ActionFactory:

- **Роль:** Базовий інтерфейс для всіх фабрик, які створюють об'єкти дій з файлами (FileAction).
- **Методи:**
  - createAction(String[] args) : FileAction - Створює об'єкт дії, використовуючи передані аргументи.

2. Клас SearchActionFactory:

- **Роль:** Фабрика для створення дій пошуку файлів (SearchFileAction).
- **Реалізує:** Інтерфейс ActionFactory.
- **Поля:**

    - logger: Logger - Для логування дій.
    - prototype: SearchFileAction - Прототип для клонування.
- **Методи:**
    - createAction(String[] args) : FileAction - Створює новий об'єкт дії пошуку шляхом клонування прототипу та налаштування його з аргументів.

3. Клас DeleteActionFactory:

- **Роль:** Фабрика для створення дій видалення файлів (DeleteFileAction).
- **Реалізує:** Інтерфейс ActionFactory.
- **Поля:**
    - prototype: DeleteFileAction - Прототип для клонування.
- **Методи:**
    - createAction(String[] args) : FileAction - Створює новий об'єкт дії видалення шляхом клонування прототипу та налаштування його з аргументів.

4. Клас SwitchDriveActionFactory:

- **Роль:** Фабрика для створення дій перемикання диска (SwitchDriveAction).
- **Реалізує:** Інтерфейс ActionFactory.
- **Поля:**
    - logger: Logger - Для логування дій.
    - prototype: SwitchDriveAction - Прототип для клонування.
- **Методи:**
    - createAction(String[] args) : FileAction - Створює новий об'єкт дії перемикання диска шляхом клонування прототипу та налаштування його з аргументів.

5. Клас CopyActionFactory:

- **Роль:** Фабрика для створення дій копіювання файлів (CopyFileAction).
- **Реалізує:** Інтерфейс ActionFactory.
- **Поля:**
    - prototype: CopyFileAction - Прототип для клонування.
- **Методи:**
    - createAction(String[] args) : FileAction - Створює новий об'єкт дії

копіювання шляхом клонування прототипу та налаштування його з аргументів.

6. Клас MoveActionFactory:

- **Роль:** Фабрика для створення дій переміщення файлів (MoveFileAction).
- **Реалізує:** Інтерфейс ActionFactory.
- **Поля:**
  - logger: Logger - Для логування дій.
  - prototype: MoveFileAction - Прототип для клонування.
- **Методи:**
  - createAction(String[] args) : FileAction - Створює новий об'єкт дії переміщення шляхом клонування прототипу та налаштування його з аргументів.

7. Клас ViewActionFactory:

- **Роль:** Фабрика для створення дій перегляду файлів (ViewFileAction).
- **Реалізує:** Інтерфейс ActionFactory.
- **Поля:**
  - logger: Logger - Для логування дій.
  - prototype: ViewFileAction - Прототип для клонування.
- **Методи:**
  - createAction(String[] args) : FileAction - Створює новий об'єкт дії перегляду шляхом клонування прототипу та налаштування його з аргументів.

8. Клас ShellController:

- **Роль:** Основний клас для обробки команд користувача.
- **Поля:**
  - logger: Logger - Для логування дій.
  - commandFactories : Map<String, ActionFactory> - Карта фабрик дій, де ключ - назва команди, а значення - відповідна фабрика.
  - actionFlowController: ActionFlowController - Контролер для управління потоком дій.
  - scanner: Scanner - Для читання введених користувачем команд.
- **Методи:**

- start() : void - Запускає цикл обробки команд.
- processCommand(String command) : void - Обробляє введену команду, розпізнає назву дії та передає її відповідній фабриці для створення та виконання.

Робота паттерну.

Виконаємо команду перегляду

3.808+02:00 INFO 14656 --- [shell] [restartedMain] c.s.commands.factory.ViewActionFactory   : Викликано createCommand у ViewCommandFactory з аргументами: [view]
2025-01-24T12:37:03.808+02:00 INFO 14656 --- [shell] [restartedMain] c.s.commands.factory.ViewActionFactory   : Initializing ViewFileAction with directory path
2025-01-24T12:37:03.810+02:00 INFO 14656 --- [shell] [restartedMain] coursework.shell.ShellController   : Передача команди 'view' до ActionFlowController
2025-01-24T12:37:0

Висновок.

Використання патерну "Фабрика" забезпечує гнучке та зручне створення команд у проєкті. Це особливо важливо, оскільки кількість команд може збільшуватися:

- **Масштабованість:** Система легко масштабується, оскільки додавання нових команд не потребує змін в існуючому коді.
- **Розширеність:** Кожен тип команди створюється окремою фабрикою (наприклад, CopyCommandFactory), що спрощує додавання нових команд через реалізацію інтерфейсу CommandFactory.
- **Інкапсуляція:** Процес створення команд прихований у фабриках, тому клієнтський код (ShellConsole) не залежить від деталей їх ініціалізації, що робить код чистішим.
- **Принцип відкритості/закритості:** Система закрита для модифікацій, але відкрита для розширення. Додавання нової команди не вимагає змін у ShellConsole; достатньо створити нову фабрику.

Таким чином, "Фабрика" покращує організацію коду, спрощує підтримку та дозволяє легко додавати нові функції.

ДОДАТОК

```
package coursework.shell;

import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.util.*;

@Component
public class ShellController {

    private final RestTemplate restTemplate = new RestTemplate();
    private Set<String> availableActions;

    public void startShell() {
        getAvailableActions();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.print("Input the action to perform or 'exit'
to stop the program: ");
            String commandLine = scanner.nextLine();
            processAction(commandLine);
        }
    }

    private void getAvailableActions() {
        try {
            ResponseEntity<Set> response =
restTemplate.getForEntity("http://localhost:8080/actions",
Set.class);
            this.availableActions = new
HashSet<>((Collection<String>) response.getBody());
        } catch (Exception e) {
            System.out.println("Unexpected error: " +
e.getMessage());
            this.availableActions = new
HashSet<>(Arrays.asList("copy", "move", "delete", "search",
"switch", "view"));
        }
        this.availableActions.add("exit");
    }

    private void processAction(String commandLine) {
        if (!commandLine.trim().isEmpty()) {
            String[] parts = commandLine.trim().split("\\s+");
            String userInput = parts[0].toLowerCase();

            if ("exit".equals(userInput)) {
```

```
                System.out.println("Program stopped");
                System.exit(0);
            }

            else if (!availableActions.contains(userInput)) {
                System.out.println("Unknown action: " + userInput);
            } else {
                try {
                    if (!"search".equals(userInput) &&
!"view".equals(userInput)) {
                        String response =
restTemplate.postForObject("http://localhost:8080/command-
line/execute", commandLine, String.class);
                        System.out.println(response);
                    } else {
                        handleDataReturning(userInput, parts);
                    }
                } catch (Exception e) {
                    System.out.println("Unexpected error: " +
e.getMessage());
                }
            }
        }
    }

    private void handleDataReturning(String userInput, String[]
parts) {
        try {
            String url;
            ResponseEntity<List> response;
            List<String> fileList;
            if ("search".equals(userInput)) {
                if (parts.length < 2) {
                    System.out.println("not enough arguments
'search'.");
                    return;
                }
                String query = parts[1];
                url = "http://localhost:8080/actions/search?query="
+ URLEncoder.encode(query, StandardCharsets.UTF_8);
                response = restTemplate.postForEntity(url, null,
List.class);
                fileList = response.getBody();
                if (fileList != null && !fileList.isEmpty()) {
                    System.out.println("Searching results:");
                    for (String fileName : fileList) {
                        System.out.println(" - " + fileName);
                    }
                } else {
                    System.out.println("No mathes.");
                }
            } else if ("view".equals(userInput)) {
                url = "http://localhost:8080/actions/view";
                if (parts.length >= 2) {
                    String path = parts[1];
                    url = url + "?path=" + URLEncoder.encode(path,
StandardCharsets.UTF_8);
```

```
                }

                response = restTemplate.postForEntity(url, null,
List.class);
                fileList = response.getBody();
                if (fileList != null && !fileList.isEmpty()) {
                    System.out.println("Current directory:");
                    for (String fileName : fileList) {
                        System.out.println(" - " + fileName);
                    }
                } else {
                    System.out.println("Current directory is
empty.");
                }
            }
        } catch (Exception e) {
            System.out.println("Error '" + userInput + "': " +
e.getMessage());
        }
    }
}
```

```
package coursework.shell.actions.factory;

import coursework.shell.actions.FileAction;

public interface ActionFactory {
    FileAction createAction(String[] args);
}
```

```
package coursework.shell.actions.factory;

import coursework.shell.actions.FileAction;
import coursework.shell.utils.ArgumentValidator;

public abstract class AbstractActionFactory<T extends FileAction>
implements ActionFactory {

    private final T prototype;

    public AbstractActionFactory(T prototype) {
        this.prototype = prototype;
    }

    @Override
    public FileAction createAction(String[] args) {
        if (!ArgumentValidator.validateArgsLength(args,
getRequiredArgsLength())) {
            return null;
        }
        T action = (T) prototype.clone();
        setActionParameters(action, args);
        logCreation(action, args);
        return action;
    }

    protected abstract void setActionParameters(T action, String[]
args);
    protected abstract int getRequiredArgsLength();
```

```
    protected abstract void logCreation(T action, String[] args);
}
package coursework.shell.actions.factory;

import coursework.shell.actions.CopyFileAction;
import coursework.shell.utils.LogUtils;
import org.slf4j.Logger;
import org.springframework.stereotype.Component;

@Component("copy")
public class CopyFileActionFactory extends
AbstractActionFactory<CopyFileAction> {

    private static final Logger logger =
LogUtils.getLogger(CopyFileActionFactory.class);

    public CopyFileActionFactory() {
        super(new CopyFileAction("", ""));
    }

    @Override
    protected void setActionParameters(CopyFileAction action,
String[] args) {
        action.setSource(args[1]);
        action.setDestination(args[2]);
    }

    @Override
    protected int getRequiredArgsLength() {
        return 3;
    }

    @Override
    protected void logCreation(CopyFileAction action, String[]
args){
        logger.info("Copy action created with source '{}' and
destination '{}'", args[1], args[2]);
    }
}
```