

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»»

Варіант №18

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

Перевірив:  
Мягкий М. Ю.

Київ 2025

## **Зміст**

<b>Тема.....</b>	<b>3</b>
<b>Мета.....</b>	<b>3</b>
<b>Завдання .....</b>	<b>3</b>
<b>Обрана тема.....</b>	<b>3</b>
<b>Короткі теоретичні відомості .....</b>	<b>4</b>
<b>Хід роботи.....</b>	<b>5</b>
<b>Робота паттерну. ....</b>	<b>8</b>
<b>Висновки .....</b>	<b>8</b>
<b>Додаток А .....</b>	<b>9</b>

**Тема.**

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

**Мета.**

Метою даної лабораторної роботи є ознайомлення з шаблонами проектування, зокрема з шаблоном "Prototype", 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.**

**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.

Анти-патерни (anti-patterns), також відомі як пастки (pitfalls) - це класи найбільш часто

впроваджуваних поганих рішень проблем. Вони вивчаються, як категорія, в разі коли їх хочуть уникнути в майбутньому, і деякі їхні окремі випадки можуть бути розпізнані при вивченні непрацюючих систем.

Термін походить з інформатики, від авторів «Банди чотирьох» книги «Шаблони проектування», яка заклала приклади практики хорошого програмування. Автори назвали ці хороші методи «шаблонами проектування», і протилежними їм є «анти-патерни». Частиною хоршої практики програмування є уникнення анти-патернів.

### Патерн "Prototype"

Шаблон "prototype" (прототип) використовується для створення об'єктів за "шаблоном" (чи "кресленню", "ескізу") шляхом копіювання шаблонного об'єкту. Для цього визначається метод "клонувати" в об'єктах цього класу.

Цей шаблон зручно використати, коли заздалегідь відомо як виглядатиме кінцевий об'єкт (мінімізується кількість змін до об'єкту шляхом створення шаблону), а також для видалення необхідності створення об'єкту - створення відбувається за рахунок клонування, і зухвалій програмі абсолютно немає необхідності знати, як створювати об'єкт.

Також, це дозволяє маніпулювати об'єктами під час виконання програми шляхом настроювання відповідних шаблонів; значно зменшується ієрархія спадкоємства (оскільки в іншому випадку це були б не шаблони, а вкладені класи, що наслідують).

## Хід роботи.

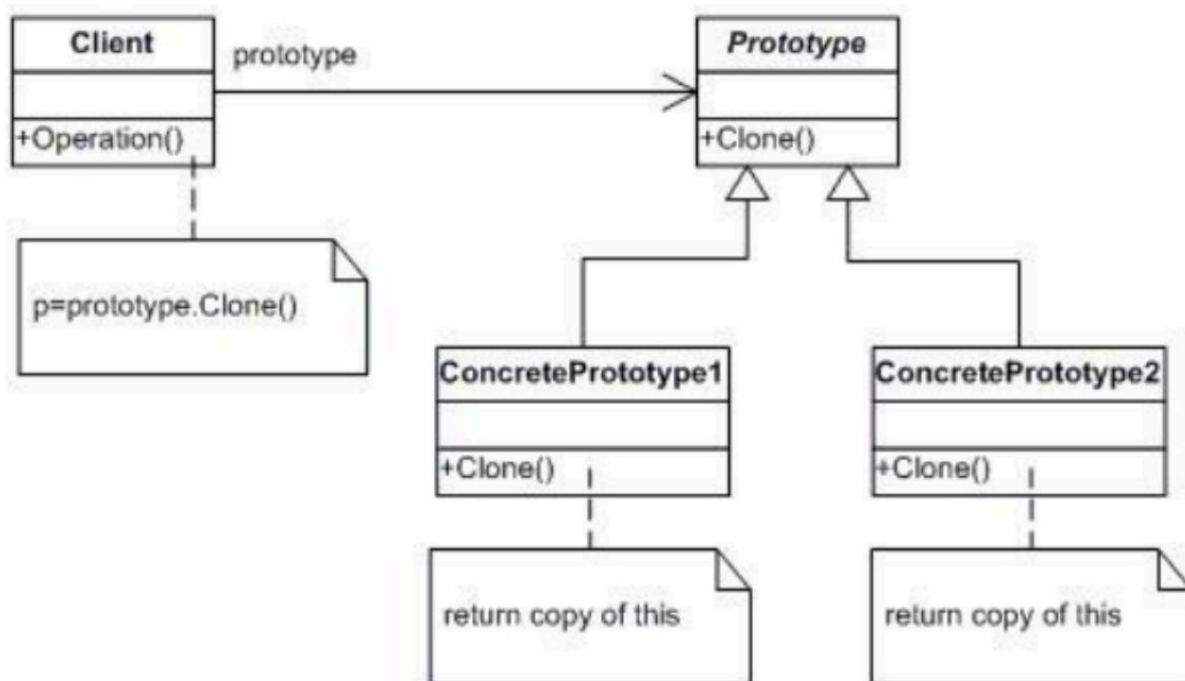


Рисунок 1.1 – UML діаграма шаблону Prototype

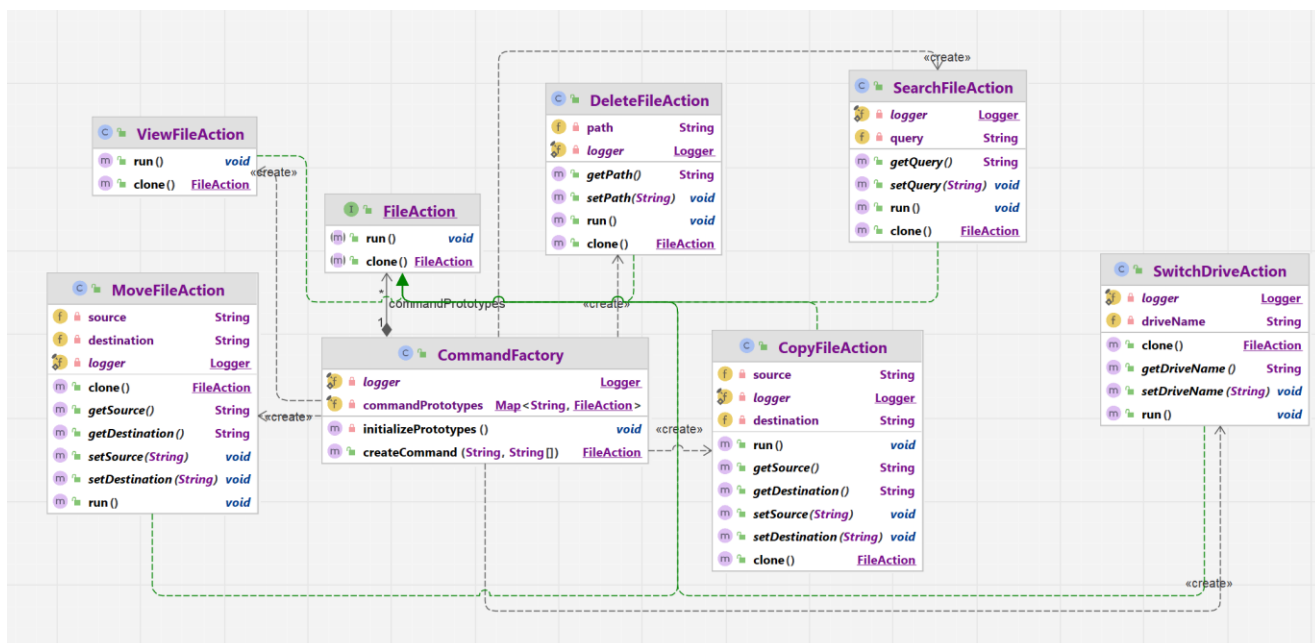


Рисунок 1.2 – Діаграма класів , згенерована IDE

### Інтерфейс FileAction:

- Роль: Базовий інтерфейс для всіх дій з файлами, встановлює загальний контракт.
- Методи:
  - clone() : FileAction - Створює копію екземпляра дії.
  - run() : void - Виконує дію.

### 2. Клас CopyFileAction:

- Роль: Представляє дію копіювання файлів.
- Реалізує: Інтерфейс FileAction.
- Поля:
  - logger: Logger - Для логування дій.
  - source: String - Шлях до файлу-джерела для копіювання.
  - destination: String - Шлях до місця призначення, куди копіювати.
- Методи:
  - getSource() : String - Повертає шлях до файлу-джерела.
  - getDestination() : String - Повертає шлях до місця призначення.
  - setSource(String) : void - Встановлює шлях до файлу-джерела.
  - setDestination(String) : void - Встановлює шлях до місця призначення.
  - clone() : FileAction - Створює копію об'єкта.
  - run() : void - Виконує дію копіювання.

### 3. Клас MoveFileAction:

- Роль: Представляє дію переміщення файлів.
- Реалізує: Інтерфейс FileAction.
- Поля:
  - destination: String - Шлях до місця призначення, куди переміщувати.
  - logger: Logger - Для логування дій.
  - source: String - Шлях до файлу-джерела для переміщення.
- Методи:
  - getSource() : String - Повертає шлях до файлу-джерела.
  - getDestination() : String - Повертає шлях до місця призначення.
  - setSource(String) : void - Встановлює шлях до файлу-джерела.
  - setDestination(String) : void - Встановлює шлях до місця призначення.
  - clone() : FileAction - Створює копію об'єкта.
  - run() : void - Виконує дію переміщення.

### 4. Клас ViewFileAction:

- Роль: Представляє дію перегляду файлів.
- Реалізує: Інтерфейс FileAction.
- Методи:
  - clone() : FileAction - Створює копію об'єкта.

- `run() : void` - Виконує дію перегляду.

#### 5. Клас `DeleteFileAction`:

- Роль: Представляє дію видалення файлів.
- Реалізує: Інтерфейс `FileAction`.
- Поля:
  - `path: String` - Шлях до файлу для видалення.
  - `logger: Logger` - Для логування дій.
- Методи:
  - `getPath() : String` - Повертає шлях до файлу.
  - `setPath(String) : void` - Встановлює шлях до файлу.
  - `clone() : FileAction` - Створює копію об'єкта.
  - `run() : void` - Виконує дію видалення.

#### 6. Клас `SearchFileAction`:

- Роль: Представляє дію пошуку файлів.
- Реалізує: Інтерфейс `FileAction`.
- Поля:
  - `logger: Logger` - Для логування дій.
  - `query: String` - Параметри для пошуку.
- Методи:
  - `getQuery() : String` - Повертає параметри пошуку.
  - `setQuery(String) : void` - Встановлює параметри пошуку.
  - `clone() : FileAction` - Створює копію об'єкта.
  - `run() : void` - Виконує дію пошуку.

#### 7. Клас `SwitchDriveAction`:

- Роль: Представляє дію перемикання диску.
- Реалізує: Інтерфейс `FileAction`.
- Поля:
  - `driveName : String` - Назва диску
  - `logger: Logger` - Для логування дій.
- Методи:
  - `getDriveName() : String` - Повертає назву диску.
  - `setDriveName(String) : void` - Встановлює назву диску.
  - `clone() : FileAction` - Створює копію об'єкта.
  - `run() : void` - Виконує дію перемикання диску.

#### 8. Клас `CommandFactory`:

- Роль: Фабрика для створення дій `FileAction`.
- Поля:
  - `logger: Logger` - Для логування дій.

- `commandPrototypes: Map<String, FileAction>` - Карта прототипів дій з файлами.
- **Методи:**
  - `initializePrototypes() : void` - Заповнює `commandPrototypes` об'єктами, які слугують прототипами для створення нових дій.
  - `createCommand(String, String[]) : FileAction` - Створює новий об'єкт дії шляхом клонування відповідного прототипу з `commandPrototypes` за ключем. Додаткові аргументи можуть бути передані для налаштування дій.



## Робота паттерну.

```
2025-01-24T12:10:48.219+02:00 INFO 7324 --- [shell] [ restartedMain]
s.coursework.factory.CommandFactory : Created delete action
2025-01-24T12:10:48.220+02:00 INFO 7324 --- [shell] [ restartedMain]
s.coursework.commands.DeleteFileAction : deleting 'lab55.pdf'
2025-01-24T12:10:48.221+02:00 INFO 7324 --- [shell] [ restartedMain]
s.coursework.commands.DeleteFileAction : file 'lab55.pdf' deleted
```

## Висновок.

Використання паттерну "Прототип" значно покращило гнучкість та продуктивність нашої системи:

1. **Швидке створення копій:** Клонування прототипів замість конструювання з нуля прискорює створення об'єктів, особливо складних.
2. **Гнучке налаштування:** Можна створювати копії команд з базовою конфігурацією і потім налаштовувати їх перед виконанням, не змінюючи оригінал.
3. **Легке розширення:** Додавання та зміна команд стали простішими та безпечнішими завдяки роботі з чіткими прототипами.

## Додаток

```

package shell.coursework.factory;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;
import shell.coursework.commands.*;

import java.util.HashMap;
import java.util.Map;

@Component
public class CommandFactory {

    private static final Logger logger =
LoggerFactory.getLogger(CommandFactory.class);
    private final Map<String, FileAction> commandPrototypes = new
HashMap<>();

    public CommandFactory() {
        initializePrototypes();
    }

    private void initializePrototypes() {
        commandPrototypes.put("view", new ViewFileAction());
        commandPrototypes.put("copy", new CopyFileAction("", ""));
        commandPrototypes.put("delete", new DeleteFileAction(""));
        commandPrototypes.put("move", new MoveFileAction("", ""));
        commandPrototypes.put("search", new SearchFileAction(""));
        commandPrototypes.put("switch", new SwitchDriveAction(""));
    }

    public FileAction createCommand(String action, String[] args) {
        FileAction prototype = commandPrototypes.get(action);
        if (prototype != null) {
            FileAction fileAction = prototype.clone();
            logger.info("Created {} action", action);

            switch (action) {
                case "copy":
                    if (args.length > 2) {
                        ((CopyFileAction)
fileAction).setSource(args[1]);
                        ((CopyFileAction)
fileAction).setDestination(args[2]);
                    } else {
                        System.out.println("Type the destination");
                        return null;
                    }
                    break;
                case "delete":
                    if (args.length > 1) {
                        ((DeleteFileAction)
fileAction).setPath(args[1]);
                    } else {

```

```

        System.out.println("Type the destination");
        return null;
    }
    break;
    case "move":
        if (args.length > 2) {
            ((MoveFileAction)
fileAction).setSource(args[1]);
            ((MoveFileAction)
fileAction).setDestination(args[2]);
        } else {
            System.out.println("Type the destination.");
            return null;
        }
        break;
    case "search":
        if (args.length > 1) {
            ((SearchFileAction)
fileAction).setQuery(args[1]);
        } else {
            System.out.println("Type serch request");
            return null;
        }
        break;
    case "switch":
        if (args.length > 1) {
            ((SwitchDriveAction)
fileAction).setDriveName(args[1]);
        } else {
            System.out.println("Drive name pls");
            return null;
        }
        break;
    }
    return fileAction;
}
return null;
}
}

package shell.coursework.commands;

import lombok.Getter;
import lombok.Setter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.nio.file.*;

@Getter
@Setter
public class DeleteFileAction implements FileAction {
    private static final Logger logger =
LoggerFactory.getLogger(DeleteFileAction.class);

    private String path;

    public DeleteFileAction(String path) {

```

```

        this.path = path;
    }

    public DeleteFileAction(DeleteFileAction deleteCommand) {
        this.path = deleteCommand.path;
    }

    @Override
    public void run() {
        logger.info("deleting '{}'", path);
        try {
            Files.deleteIfExists(Paths.get(path));
            logger.info("file '{}' deleted", path);
        } catch (IOException e) {
            logger.error("Error", e);
        }
    }

    @Override
    public FileAction clone() {
        return new DeleteFileAction(this);
    }
}

package shell.coursework.commands;

public interface FileAction extends Cloneable {
    void run();
    FileAction clone();
}

```