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

Лабораторна робота №7

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Шаблони «Mediator»,
«Facade», «Bridge», «Template method»

Варіант №18

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

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

Тема.
«Шаблони «Mediator»,«Facade», «Bridge», «Template method»
Мета.
Метою даної лабораторної роботи є ознайомлення з шаблонами проєктування, зокрема з шаблоном "Template 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.

Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації подклассам. Можна привести в приклад формування веб-сторінки: необхідно додати заголовки, вміст сторінки, файли, що додаються, і нижню частину сторінки. Код для додавання вмісту сторінки може бути абстрактним і реалізовуватися в різних класах - AspNetCompiler, HtmlCompiler, PhpCompiler і т.п. Додавання всіх інших елементів виконується за допомогою вихідного абстрактного класу з алгоритмом.

Даний шаблон дещо нагадує шаблон «фабричний метод», однак область його використання абсолютно інша - для покрокового визначення конкретного алгоритму; більш того, даний шаблон не обов'язково створює нові об'єкти - лише визначає послідовність дій.

Хід роботи.

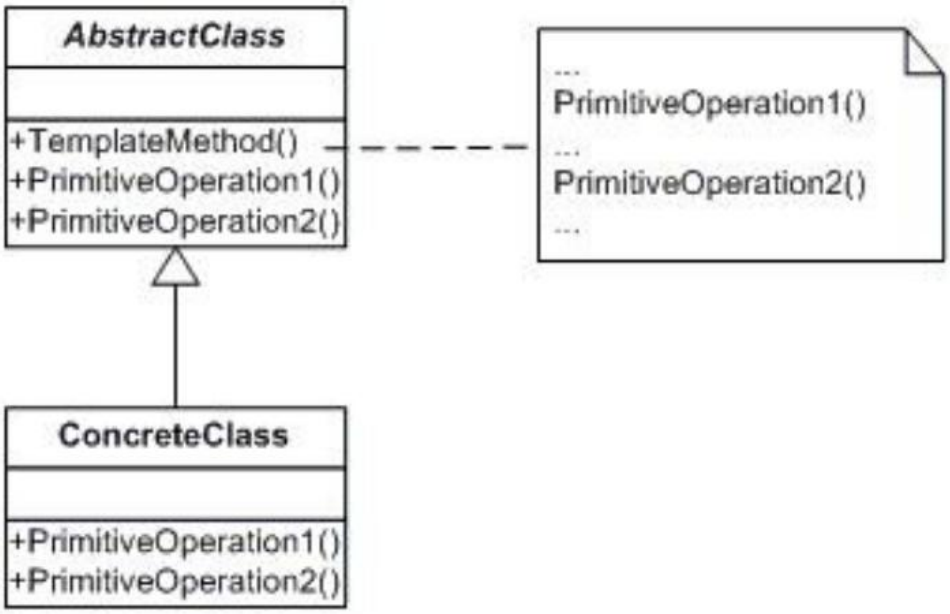

Рисунок 1.1 – UML діаграма шаблону Template method

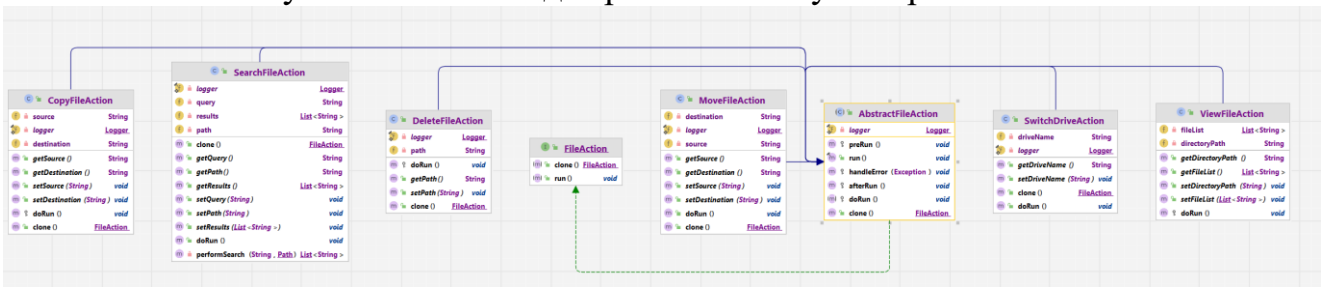

Рисунок 1.2 – Діаграма класів , згенерована IDE

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. Абстрактний клас AbstractFileAction:

- **Роль:** Забезпечує загальну структуру виконання дій, реалізуючи шаблонний метод.
- **Реалізує:** Інтерфейс FileAction.
- **Поля:**
  - logger: Logger - Для логування дій.
- **Методи:**
  - handleError(Exception) : void - Обробляє помилки, що виникають під час виконання дії.
  - preRun() : void - Виконує дії перед основною логікою дії (наприклад, валідація).
  - afterRun() : void - Виконує дії після основної логіки дії (наприклад, очищення ресурсів).
  - run() : void - Шаблонний метод, що послідовно викликає preRun(), doRun(), afterRun() та обробку помилок.
  - clone() : FileAction - Створює копію об'єкта.
  - doRun() : void - Абстрактний метод, який визначає специфічну логіку кожної конкретної дії.

3. Клас CopyFileAction:

- **Роль:** Представляє дію копіювання файлів.
- **Наслідує:** AbstractFileAction.
- **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.
- doRun() : void - Реалізує логіку копіювання файлів.

4. Клас MoveFileAction:

- **Роль:** Представляє дію переміщення файлів.
- **Наслідує:** AbstractFileAction.
- **Поля:**
  - logger: Logger - Для логування дій.
  - source: String - Шлях до файлу-джерела для переміщення.
  - destination: String - Шлях до місця призначення, куди переміщувати.
- **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.

  * doRun() : void - Реалізує логіку переміщення файлів.

5. Клас DeleteFileAction:

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

  * doRun() : void - Реалізує логіку видалення файлів.

6. Клас SearchFileAction:

- **Роль:** Представляє дію пошуку файлів.
- **Наслідує:** AbstractFileAction.
- **Поля:**
    - logger: Logger - Для логування дій.
    - results: List<String> - Результати пошуку.
    - query: String - Параметри для пошуку.
    - path : String - Шлях до директорії для пошуку.
- **Методи:**
- performSearch(String, String) : List<String> - Виконує пошук. * getQuery() : String - Повертає параметри пошуку. * getPath() : String - Повертає шлях до директорії для пошуку. * getResults() : List<String> - Повертає результати пошуку. * setQuery(String) : void - Встановлює параметри пошуку.
- setPath(String) : void - Встановлює шлях до директорії для пошуку. * setResults(List<String>) : void - Встановлює результати пошуку.
- clone() : FileAction - Створює копію об'єкта. * doRun() : void - Реалізує логіку пошуку файлів.

7. Клас ViewFileAction:

- **Роль:** Представляє дію перегляду файлів.
- **Наслідує:** AbstractFileAction.
- **Поля:**
    - fileList: List<String> - Список файлів для перегляду.
    - directoryPath : String - Шлях до директорії для перегляду.
- **Методи:**
    - getDirectoryPath() : String - Повертає шлях до директорії.
    - getFileList() : List<String> - Повертає список файлів.
    - setDirectoryPath(String) : void - Встановлює шлях до директорії.
    - setFileList(List<String>) : void - Встановлює список файлів.
- doRun() : void - Реалізує логіку перегляду файлів.

8. Клас SwitchDriveAction:

- **Роль:** Представляє дію перемикання диску.
- **Наслідує:** AbstractFileAction.

- **Поля:**
  - logger: Logger - Для логування дій.
  - driveName : String - Назва диску
- **Методи:**
  - getDriveName() : String - Повертає назву диску.
  - setDriveName(String) : void - Встановлює назву диску.
- clone() : FileAction - Створює копію об'єкта.

  * doRun() : void - Реалізує логіку перемикання диска.

Висновок.

Реалізація патерну "Шаблонний метод" через `AbstractFileAction` значно покращила структуру та надійність нашої системи:

- **Стандартизація виконання:** Загальна логіка виконання дій з файлами винесена в `AbstractFileAction`, що стандартизувало процес через чіткі етапи: `preRun()`, `doRun()`, `afterRun()` та обробку помилок `handleError()`.
- **Фокус на специфіці:** Конкретні дії, такі як `CopyFileAction`, `DeleteFileAction` та інші, тепер зосереджені лише на своїй специфічній логіці, реалізованій в методі `doRun()`, що робить код простішим і зрозумілішим.
- **Покращена надійність:** Метод `preRun()`, наприклад, може містити валідацію вхідних даних, що допомагає запобігти помилкам на ранніх етапах виконання дій.
- **Дотримання принципів ООП:** Застосування "Шаблонного методу" сприяє дотриманню принципу єдиного обов'язку, оскільки загальна логіка виконання дій зосереджена в одному місці, що спрощує розширення системи.
- **Зручне розширення:** Додавання нових дій з файлами спростилося: достатньо створити клас, що розширює `AbstractFileAction` та реалізувати метод `doRun()`, не турбуючись про загальну структуру виконання.
- **Структурований та гнучкий код:** "Шаблонний метод" допоміг уникнути дублювання коду та створив єдиний механізм виконання для всіх дій з файлами, зробивши код більш структурованим, гнучким та легким для підтримки.

Таким чином, використання "Шаблонного методу" покращило якість коду, зробило його більш надійним та спростило майбутній розвиток системи.
thumb_upthumb_down

ДОДАТОК

```
public abstract class AbstractFileAction implements FileAction {
    private static final Logger logger =
LogUtils.getLogger(CopyFileActionFactory.class);

    @Override
    public final void run() {
        try {
            preRun();
            doRun();
            afterRun();
        } catch (Exception e) {
            handleError(e);
        }
    }

    protected void preRun() {
        logger.info("Start of action {}",
this.getClass().getSimpleName());
    }

    protected abstract void doRun();

    protected void afterRun() {
        logger.info("End of action {}",
this.getClass().getSimpleName());
    }

    protected void handleError(Exception e) {
        if (e instanceof IllegalArgumentException) {
            logger.error("Incorect arguments: {}", e.getMessage());
        } else if (e instanceof FileNotFoundException) {
            logger.error("File not found: {}", e.getMessage());
        } else {
            logger.error("Unexpected failure during the execution of
{}", this.getClass().getSimpleName(), e);
        }
    }

    @Override
    public FileAction clone() {
        try {
            return (FileAction) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new AssertionError("Not supported", e);
        }
    }
}
@Getter
@Setter
public class ViewFileAction extends AbstractFileAction {
    private String directoryPath;
    private List<String> fileList;
```

```
    public ViewFileAction() {
        this.directoryPath =
Paths.get("").toAbsolutePath().toString();
    }

    public ViewFileAction(String directoryPath) {
        this.directoryPath = directoryPath;
    }

    @Override
    protected void doRun() {
        Path toDirPath = Paths.get(directoryPath);
        try (DirectoryStream<Path> stream =
Files.newDirectoryStream(toDirPath)) {
            fileList = new ArrayList<>();
            for (Path path : stream) {
                fileList.add(path.getFileName().toString());
            }
        } catch (IOException e) {
            try {
                throw new Exception("Issue encountered while listing
directory contents: " + e.getMessage(), e);
            } catch (Exception ex) {
                throw new RuntimeException(ex);
            }
        }
    }
}
```