Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №8

Технологія розроблення програмного забезпечення

«Shell (total commander)»

Варіант 18

Виконав
студент групи ІА-13
Окаянченко Давид Олександрович

Перевірив:
Мягкий Михайло Юрійович

Київ 2023

Мета: Дослідити шаблони «Composite», «Flyweigth», «Interpreter», «Visitor» та навчитися застосовувати один із них на практиці.

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.

Варіант:

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.

Хід роботи

Шаблони проєктування - це певні способи розв'язання типових проблем, які виникають під час розробки програмного забезпечення. Вони є своєрідними "рецептами" або наборами правил, які вже доведено було успішними в реальних проектах. Їх використання допомагає розробникам ефективно вирішувати спільні завдання та уникати типових помилок.

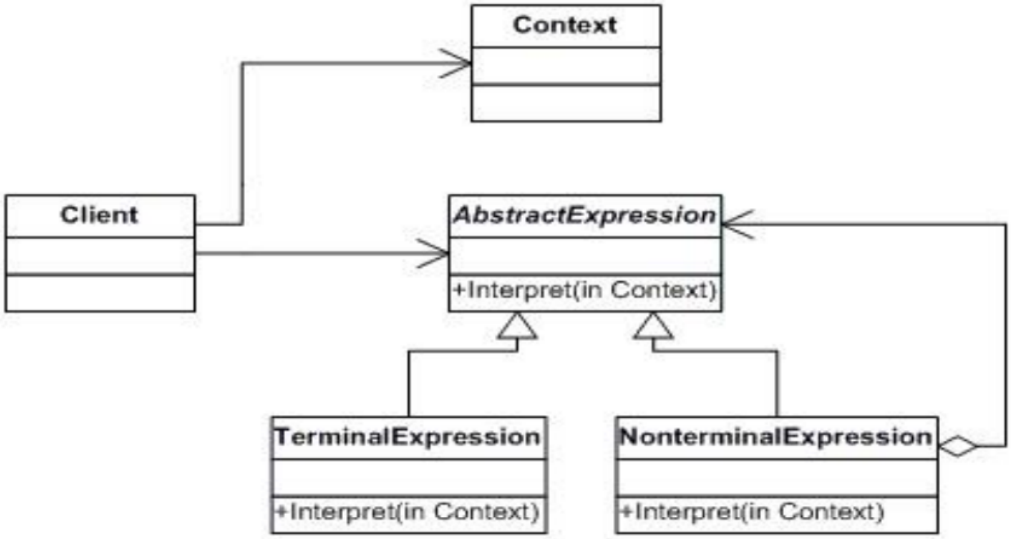
Важливі аспекти шаблонів проєктування:

- Полегшення розробки: Вони надають структурований підхід до розв'язання проблем, що допомагає розробникам швидше і ефективніше створювати програмне забезпечення.
- Підвищення якості: Шаблони допомагають уникати поширених помилок, що можуть призвести до поганої продуктивності або низької якості програми.
- Підвищення перевикористання: Вони сприяють створенню універсальних рішень, які можна використовувати в різних контекстах.
- Покращення зрозуміння: Використання шаблонів полегшує іншим розробникам розуміння коду та сприяє легшій підтримці.

- Спрощення спільної роботи: Шаблони допомагають командам розробників працювати спільно, оскільки вони знайомі із загальними концепціями та підходами.

Шаблон проєктування «Interpreter»

Структура:



Призначення:

Даний шаблон використовується для подання граматики і інтерпретатора для вибраної мови (наприклад, скриптової). Граматика мови представлена термінальними і нетермінальними символами, кожен з яких інтерпретується в контексті використання. Клієнт передає контекст і сформовану пропозицію в використовувану мову в термінах абстрактного синтаксичного дерева (деревоподібна структура, яка однозначно визначає ієрархію виклику підвиразів), кожен вираз інтерпретується окремо з використанням контексту. У разі наявності дочірніх виразів, батьківський вираз інтерпретує спочатку дочірні (рекурсивно), а потім обчислює результат власної операції. Шаблон зручно використовувати в разі невеликої граматики (інакше розростеться кількість використовуваних класів) і відносно простого контексту (без взаємозалежностей і т.п.). Даний шаблон визначає базовий каркас інтерпретатора, який за допомогою рекурсії повертає результат обчислення пропозиції на основі результатів окремих елементів. При використанні даного шаблону дуже легко реалізовується і розширюється граматика, а також додаються нові способи інтерпретації виразів

Переваги та недоліки:

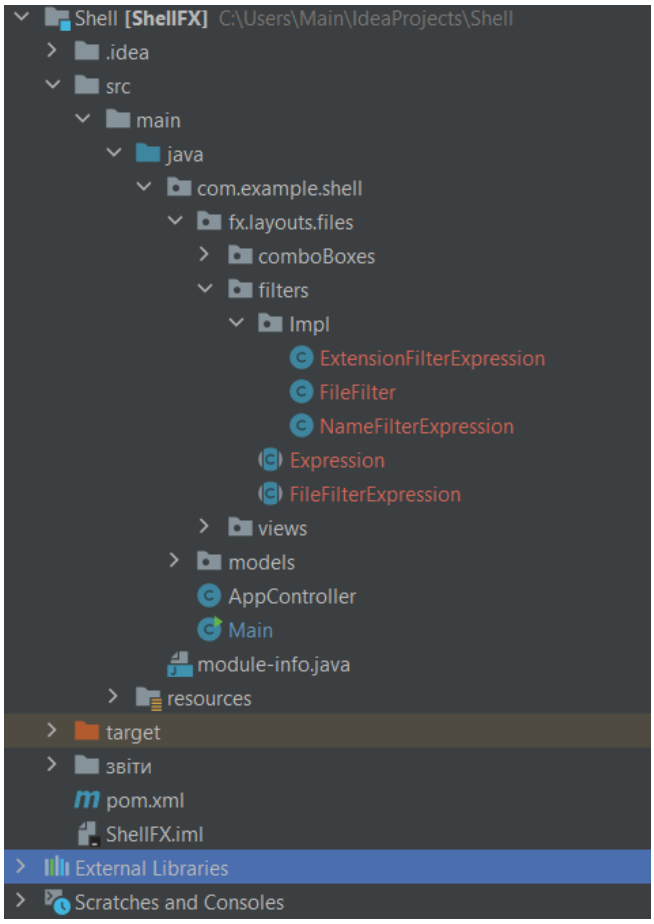
+ Граматику стає легко розширювати та змінювати, реалізації класів, що описують вузли абстрактного синтаксичного дерева схожі (легко кодуються).
+ Можна легко змінювати спосіб обчислення виразів.

- Супровід граматики з великим числом правил стає важким.

Реалізація:

Шаблон Interpreter в моєму проекті надає можливість створювати різноманітні фільтри для файлів, що використовуються в графічному інтерфейсі програми на JavaFX. Цей підхід дозволяє динамічно змінювати та комбінувати умови фільтрації файлів за різними критеріями, такими як розширення файлів чи їхні імена. Кожен фільтр представлений власним класом, який реалізує логіку фільтрації у методі interpret. Це дозволяє гнучко налаштовувати та розширювати можливості фільтрації у файловому менеджері.

Стуктура проєкта:



Клас Main:

Клас Main є центральним елементом додатку, який використовує широкий спектр шаблонів проектування, таких як Prototype, Template Method, Factory Method та Interpreter. Він забезпечує взаємодію з інтерфейсом користувача JavaFX, включає в себе логіку обробки користувацького вводу, створення об'єктів та фільтрацію файлів за допомогою інтерпретатора виразів фільтрації.

```
package com.example.shell;

import com.example.shell.fx.layouts.files.filters.FileFilterExpression;
import com.example.shell.fx.layouts.files.filters.Impl.ExtensionFilterExpression;
import com.example.shell.fx.layouts.files.filters.Impl.FileFilter;
import com.example.shell.fx.layouts.files.filters.Impl.NameFilterExpression;
import com.example.shell.fx.layouts.files.views.FilesViewTab;
import com.example.shell.fx.layouts.files.views.FilesViewFactory;
import com.example.shell.fx.layouts.files.comboBoxes.Impl.ViewComboBoxTemplate;
import com.example.shell.fx.layouts.files.views.Impl.FilesView;
import com.example.shell.fx.layouts.files.views.ListFilesViewFactory;
import com.example.shell.fx.layouts.files.views.TableFilesViewFactory;
import com.example.shell.fx.layouts.files.comboBoxes.ComboBoxTemplate;
import com.example.shell.fx.layouts.files.comboBoxes.Impl.DiskComboBoxTemplate;
import com.example.shell.fx.layouts.files.views.Impl.FilesViewType;
import com.example.shell.models.Disk;
import com.example.shell.models.User;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import org.kordamp.bootstrapfx.BootstrapFX;

import java.io.File;
import java.io.IOException;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class Main extends javafx.application.Application {
    public static void main(String[] args) {
        launch();
    }

    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource("shell.fxml"));
        Scene scene = new Scene(fxmlLoader.load());
        scene.getStylesheets().add(BootstrapFX.bootstrapFXStylesheet());
        stage.setTitle("Shell");
        stage.setScene(scene);
        stage.show();

        // Test prototype pattern
        User user1 = new User("Davyd", "david@gmail.com", "123123");
        User user2 = user1.clone();
        user2.setName("Vitaliy");
        user2.setEmail("vitaliy@gmail.com");

        // Test Template and Factory method patterns
        List<File> testFiles = Arrays.asList(
                new File("Folder1"),
                new File("Folder2"),
                new File("File1.txt"),
                new File("File2.txt"),
```

```
                new File("File3.txt")
        );

        List<Disk> testDisks = Arrays.asList(
                new Disk("C"),
                new Disk("D"),
                new Disk("E")
        );

        ComboBoxTemplate diskComboBoxTemplate = new DiskComboBoxTemplate();

        ComboBox<String> diskComboBox = diskComboBoxTemplate.createMenu(
                testDisks.stream().map(Disk::getName).toList());

        Tab diskTab = new Tab("Disk Menu");
        diskTab.setContent(diskComboBox);


        ComboBoxTemplate viewComboBoxTemplate = new ViewComboBoxTemplate();

        ComboBox<String> viewComboBox = viewComboBoxTemplate.createMenu(
                Arrays.stream(FilesViewType.values())
                        .map(Enum::name)
                        .collect(Collectors.toList()));

        Tab viewTab = new Tab("View Menu");
        BorderPane pane = new BorderPane();
        pane.setTop(viewComboBox);

        viewComboBox.setOnAction(event -> {
            String selectedView = viewComboBox.getValue();
            FilesViewType viewType =
FilesViewType.valueOf(selectedView.toUpperCase());

            FilesViewFactory filesViewFactory;

            if (viewType == FilesViewType.LIST) {
                filesViewFactory = new ListFilesViewFactory();
            } else {
                filesViewFactory = new TableFilesViewFactory();
            }

            FilesView filesView = filesViewFactory.create();
            FilesViewTab.setFilesView(filesView);
            pane.setCenter(FilesViewTab.getFilesView().getNode());
        });

        final FilesViewFactory filesViewFactory;

        filesViewFactory = new ListFilesViewFactory();
        FilesView listView = filesViewFactory.create();

        FilesViewTab.setFilesView(listView);
        FilesViewTab.setFiles(testFiles);

        pane.setCenter(FilesViewTab.getFilesView().getNode());
        viewTab.setContent(pane);

        // Test Interpreter pattern
        TextField extensionTextField = new TextField();
        extensionTextField.setPromptText("Enter Extension");

        // Кнопка для застосування фільтрації
        Button applyFilterButton = new Button("Apply Extension Filter");
        applyFilterButton.setOnAction(event -> {
            String extension = extensionTextField.getText();
            if (!extension.isEmpty()) {
                FileFilterExpression extensionFilter = new
```

```
ExtensionFilterExpression(extension);
                FileFilter.applyFilter((ListView<String>)
FilesViewTab.getFilesView().getNode(), testFiles, extensionFilter);
            } else {
                FilesViewTab.setFiles(testFiles);
            }
        });

        TextField nameTextField = new TextField();
        nameTextField.setPromptText("Enter File Name");

        // Кнопка для застосування фільтрації за іменем файлу
        Button applyNameFilterButton = new Button("Apply Name Filter");
        applyNameFilterButton.setOnAction(event -> {
            String targetName = nameTextField.getText();
            if (!targetName.isEmpty()) {
                FileFilterExpression nameFilter = new
NameFilterExpression(targetName);
                FileFilter.applyFilter((ListView<String>)
FilesViewTab.getFilesView().getNode(), testFiles, nameFilter);
            } else {
                FilesViewTab.setFiles(testFiles);
            }
        });

        VBox root = new VBox(extensionTextField, applyFilterButton, nameTextField,
applyNameFilterButton, FilesViewTab.getFilesView().getNode());
        viewTab.setContent(root);

        TabPane tabPane = new TabPane(diskTab, viewTab);
        Scene testScene = new Scene(tabPane, 640, 480);
        stage.setScene(testScene);
        stage.show();
    }
}
```

Абстрактний клас Expression:

Клас Expression є абстрактним класом, представляючим абстрактний клас для виразів у патерні Interpreter. Він визначає абстрактний метод interpret, який вимагає від конкретних підкласів надавати власну реалізацію для виразу, який може бути інтерпретований.

```
package com.example.shell.fx.layouts.files.filters;

public abstract class Expression<T> {
    public abstract boolean interpret(T context);
}
```

Абстрактний клас FileFilterExpression:

Клас FileFilterExpression є абстрактним підкласом класу Expression<File>, який визначає загальний абстрактний клас виразів фільтрації файлів у контексті патерну Interpreter. Він вимагає від конкретних підкласів надавати власну реалізацію методу interpret(File file), який визначає, чи відповідає файл заданому критерію фільтрації.

```
package com.example.shell.fx.layouts.files.filters;

import java.io.File;

public abstract class FileFilterExpression extends Expression<File> {
    public abstract boolean interpret(File file);
}
```

Клас ExtensionFilterExpression:

Клас ExtensionFilterExpression є конкретною реалізацією абстрактного класу FileFilterExpression, представляючи вираз фільтрації за розширенням файлів у контексті патерну Interpreter. Цей клас перевіряє, чи розширення заданого файлу відповідає зазначеному критерію фільтрації.

```
package com.example.shell.fx.layouts.files.filters.Impl;

import com.example.shell.fx.layouts.files.filters.FileFilterExpression;

import java.io.File;

public class ExtensionFilterExpression extends FileFilterExpression {
    private final String extension;

    public ExtensionFilterExpression(String extension) {
        this.extension = extension;
    }

    @Override
    public boolean interpret(File file) {
        return file.getName().endsWith("." + extension);
    }
}
```

Клас NameFilterExpression:

Клас NameFilterExpression є конкретною реалізацією абстрактного класу FileFilterExpression у контексті патерну Interpreter. Він представляє вираз фільтрації за іменем файлу і перевіряє, чи містить ім'я файлу зазначену підстроку, що відповідає критеріям фільтрації.

```
package com.example.shell.fx.layouts.files.filters.Impl;

import com.example.shell.fx.layouts.files.filters.FileFilterExpression;

import java.io.File;

public class NameFilterExpression extends FileFilterExpression {
    private final String targetName;

    public NameFilterExpression(String targetName) {
        this.targetName = targetName;
    }

    @Override
    public boolean interpret(File file) {
        return file.getName().contains(targetName);
    }
}
```

Клас FileFilter:

Клас FileFilter є частиною реалізації патерну Interpreter в контексті фільтрації файлів у програмі на JavaFX. Він містить метод applyFilter, який використовує заданий вираз FileFilterExpression для фільтрації списку файлів, і виводить результат у відповідний ListView у графічному інтерфейсі.

```
package com.example.shell.fx.layouts.files.filters.Impl;

import com.example.shell.fx.layouts.files.filters.FileFilterExpression;
import javafx.scene.control.ListView;

import java.io.File;
import java.util.List;

public class FileFilter {
    public static void applyFilter(ListView<String> fileListView, List<File> files,
FileFilterExpression filterExpression) {
        fileListView.getItems().clear();

        for (File file : files) {
            if (filterExpression.interpret(file)) {
                fileListView.getItems().add(file.getName());
            }
        }
    }
}
```

Висновок: У ході виконання лабораторної роботи було проведено ознайомлення з теоретичними відомостями та реалізовано шаблон проєктування «Interpreter». Окрім того, підготовлений звіт включає всі необхідні компоненти, що відображають структуру розробленої системи.