

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 №4

**Технологія розроблення програмного забезпечення**

«Shell (total commander)»

## Варіант 18

Виконав Перевірив:

студент групи ІА-13 Мягкий Михайло

Окаянченко Давид Олександрович Юрійович

Київ 2023

Мета: Дослідити шаблони «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY» 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 «State»

## Структура:

## 

## Призначення:

Шаблон «State» (Стан) дозволяє змінювати логіку роботи об'єктів у випадку зміни їх внутрішнього стану. Наприклад, відсоток нарахованих на картковий рахунок грошей залежить від стану картки: Visa Electron, Classic, Platinum і т.д. Або обсяг послуг, які надані хостинг компанією, змінюється в залежності від обраного тарифного плану (стану членства - бронзовий, срібний або золотий клієнт). Реалізація даного шаблону полягає в наступному: пов'язані зі станом поля, властивості, методи і дії виносяться в окремий загальний інтерфейс (State); кожен стан являє собою окремий клас (ConcreteStateA, ConcreteStateB), які реалізують загальний інтерфейс. Об’єкти, що мають стан (Context), при зміні стану просто записують новий об'єкт в поле state, що призводить до повної зміни поведінки об'єкта. Це дозволяє легко додавати в майбутньому і обробляти нові стани, відокремлювати залежні від стану елементи об'єкта в інших об'єктах, і відкрито проводити заміну стану (що має сенс у багатьох випадках).

## Переваги та недоліки:

+ Позбавляє від безлічі великих умовних операторів машини станів.

+ Концентрує в одному місці код, пов'язаний з певним станом.

+ Спрощує код контексту.

- Може невиправдано ускладнити код, якщо станів мало і вони рідко змінюються.

## Реалізація:

# Стуктура проєкта:

# 

# Клас Main:

package com.example.shellfx;  
  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
import org.kordamp.bootstrapfx.BootstrapFX;  
  
import java.io.IOException;  
  
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();  
 }  
}

# Клас Controller:

package com.example.shellfx;  
  
import com.example.shellfx.models.Disk;  
import com.example.shellfx.models.FileManager.FileManager;  
import com.example.shellfx.models.FileManager.State.Impl.CopyState;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.stage.DirectoryChooser;  
  
import java.io.File;  
  
public class Controller {  
 private final FileManager fileManager = new FileManager();  
  
 @FXML  
 private Button selectDestinationPathButton;  
 @FXML  
 private Button selectDriveButton;  
  
 @FXML  
 private Button copyButton;  
  
 @FXML  
 private void selectDriveButtonClicked() {  
 DirectoryChooser directoryChooser = new DirectoryChooser();  
 directoryChooser.setTitle("Оберіть диск");  
 Disk selectedDrive = new Disk(directoryChooser.showDialog(null).getPath());  
 fileManager.setCurrentDisk(selectedDrive);  
 }  
  
 @FXML  
 private void selectDestinationPathButtonClicked() {  
 DirectoryChooser directoryChooser = new DirectoryChooser();  
 directoryChooser.setTitle("Оберіть пункт призначення");  
 File selectedDestinationPath = directoryChooser.showDialog(null);  
 if (selectedDestinationPath != null) {  
 fileManager.setDestination(selectedDestinationPath);  
 }  
 }  
  
 @FXML  
 private void copyButtonClicked() {  
 fileManager.setState(new CopyState());  
 fileManager.perform();  
 }  
}

# Клас User:

package com.example.shellfx.models;  
  
public class User {  
 private final int id;  
 private final String name;  
 private final String email;  
 private final String password;  
  
 public User(int id, String name, String email, String password) {  
 this.id = id;  
 this.name = name;  
 this.email = email;  
 this.password = password;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
}

# Клас Disk:

package com.example.shellfx.models;  
  
import java.io.File;  
import java.util.Objects;  
  
public class Disk extends File {  
 public Disk(String pathname) {  
 super(pathname);  
 }  
  
 public String getName() {  
 return super.toString().substring(0, 1);  
 }  
  
 public double getFillPercentage() {  
 return 100 - (((double) super.getFreeSpace() / super.getTotalSpace()) \* 100);  
 }  
  
 @Override  
 public String toString() {  
 return "Disk{" +  
 "name='" + getName() + '\'' +  
 '}';  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof Disk disk)) return false;  
 return Objects.*equals*(getName(), disk.getName());  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(getName());  
 }  
}

# Клас FileManager:

package com.example.shellfx.models.FileManager;  
  
import com.example.shellfx.models.Disk;  
import com.example.shellfx.models.FileManager.State.FileManagerState;  
  
import java.io.File;  
import java.util.Objects;  
  
public class FileManager {  
 private Disk currentDisk;  
 private File source;  
 private File destination;  
 private FileManagerState state;  
  
 public Disk getCurrentDisk() {  
 return currentDisk;  
 }  
  
 public File getSource() {  
 return source;  
 }  
  
 public File getDestination() {  
 return destination;  
 }  
  
 public FileManagerState getState() {  
 return state;  
 }  
  
 public void setCurrentDisk(Disk currentDisk) {  
 this.currentDisk = currentDisk;  
 }  
  
 public void setDestination(File destination) {  
 this.destination = destination;  
 }  
  
 public void setState(FileManagerState state) {  
 this.state = state;  
 }  
  
 public void getUpInFileSystem() {  
 String path = source.getPath();  
 this.source = new File(path.substring(0, path.lastIndexOf(File.*separator*)));  
 }  
  
 public void getDownInFileSystem(String folderPath) {  
 this.source = new File(source.getPath() + File.*separator* + folderPath);  
 }  
  
 public void perform() {  
 state.perform(this);  
 }  
  
 @Override  
 public String toString() {  
 return "FileManager{" +  
 "currentDisk=" + currentDisk +  
 ", source='" + source + '\'' +  
 ", state=" + state +  
 '}';  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof FileManager fileManager)) return false;  
 return Objects.*equals*(getCurrentDisk(), fileManager.getCurrentDisk())  
 && Objects.*equals*(getSource(),  
 fileManager.getSource());  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(getCurrentDisk(), getSource());  
 }  
}

# Інтерфейс FileManagerState:

package com.example.shellfx.models.FileManager.State;  
  
import com.example.shellfx.models.FileManager.FileManager;  
  
public interface FileManagerState {  
 void perform(FileManager ctx);  
}

# Клас CopyState:

package com.example.shellfx.models.FileManager.State.Impl;  
  
import com.example.shellfx.models.FileManager.FileManager;  
import com.example.shellfx.models.FileManager.State.FileManagerState;  
import org.apache.commons.io.FileUtils;  
  
import java.io.File;  
import java.io.IOException;  
  
public class CopyState implements FileManagerState {  
  
 @Override  
 public void perform(FileManager ctx) {  
 try {  
 copy(ctx.getSource(), ctx.getDestination());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void copy(File source, File destination) throws IOException {  
 if (!destination.exists() && destination.mkdirs()) {  
 if (source.isDirectory()) FileUtils.*copyDirectory*(source, destination);  
 if (source.isFile()) FileUtils.*copyFile*(source, destination);  
 }  
 }  
}

# Клас MoveState:

package com.example.shellfx.models.FileManager.State.Impl;  
  
import com.example.shellfx.models.FileManager.FileManager;  
import com.example.shellfx.models.FileManager.State.FileManagerState;  
  
import java.io.File;  
  
public class MoveState implements FileManagerState {  
  
 @Override  
 public void perform(FileManager ctx) {  
 move(ctx.getSource(), ctx.getDestination());  
 }  
  
 private void move(File source, File destination) {  
 if (source.exists() && !destination.exists()) {  
 File newFile = new File(destination, source.getName());  
 source.renameTo(newFile);  
 }  
 }  
}

# Клас DeleteState:

package com.example.shellfx.models.FileManager.State.Impl;  
  
import com.example.shellfx.models.FileManager.FileManager;  
import com.example.shellfx.models.FileManager.State.FileManagerState;  
import org.apache.commons.io.FileUtils;  
  
import java.io.File;  
import java.io.IOException;  
  
public class DeleteState implements FileManagerState {  
  
 @Override  
 public void perform(FileManager ctx) {  
 try {  
 delete(ctx.getSource());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void delete(File source) throws IOException {  
 if (source.isDirectory()) FileUtils.*deleteDirectory*(source);  
 if (source.isFile()) FileUtils.*delete*(source);  
 }  
}

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