

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 №7

**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 «Mediator», «Facade», «Bridge», «Template method» 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 «Template Method»

## Структура:

## 

## Призначення:

Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підклассам. 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.

## Переваги та недоліки:

+ Полегшує повторне використання коду.

- Ви жорстко обмежені скелетом існуючого алгоритму.

- Ви можете порушити принцип підстановки Барбари Лісков, змінюючи базову поведінку одного з кроків алгоритму через підклас.

- З ростом кількості кроків шаблонний метод стає занадто складно підтримувати

## Реалізація:

У проєкті шаблон Template Method використовується у класах, які відповідають за створення та налаштування комбінованих меню (ComboBox) для вибору диска і вида відображення файлів у файловому менеджері. Абстрактний клас ComboBoxTemplate визначає шаблонний метод createMenu, який включає базовий алгоритм створення меню, і визначає абстрактні методи createBaseMenu та customizeMenu, які залишаються для реалізації в конкретних підкласах.

Такий підхід дозволяє розширювати та модифікувати логіку створення меню в окремих класах (DiskComboBoxTemplate і ViewComboBoxTemplate), при цьому забезпечуючи сталу структуру базового алгоритму. Це дозволяє зберігати загальний інтерфейс та зменшує залежність конкретних класів від деталей реалізації.

# Стуктура проєкта:

# 

# Клас Main:

Клас Main управляє точкою входу в JavaFX-застосування, встановлює вікно та завантажує макет. Він також включає в себе демонстрацію шаблонів Template Method, Prototype та Factory Method для створення комбо-боксів вибору диска та виду виводу файлів, а також тестові дані для відображення вмісту файлового менеджера.

package com.example.shell;  
  
import com.example.shell.fx.layouts.files.view.views.FilesViewTab;  
import com.example.shell.fx.layouts.files.view.views.FilesViewFactory;  
import com.example.shell.fx.layouts.files.view.comboBoxes.Impl.ViewComboBoxTemplate;  
import com.example.shell.fx.layouts.files.view.views.Impl.FilesView;  
import com.example.shell.fx.layouts.files.view.views.ListFilesViewFactory;  
import com.example.shell.fx.layouts.files.view.views.TableFilesViewFactory;  
import com.example.shell.fx.layouts.files.view.comboBoxes.ComboBoxTemplate;  
import com.example.shell.fx.layouts.files.view.comboBoxes.Impl.DiskComboBoxTemplate;  
import com.example.shell.fx.layouts.files.view.views.Impl.FilesViewType;  
import com.example.shell.models.Disk;  
import com.example.shell.models.User;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.layout.BorderPane;  
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  
 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.*setFiles*(testFiles);  
 FilesViewTab.*setFilesView*(listView);  
  
 pane.setCenter(FilesViewTab.*getFilesView*().getNode());  
 viewTab.setContent(pane);  
  
 TabPane tabPane = new TabPane(diskTab, viewTab);  
 Scene testScene = new Scene(tabPane, 640, 480);  
 stage.setScene(testScene);  
 stage.show();  
 }  
}

# Клас FilesViewTab:

Цей клас представляє собою вкладку для файлового менеджера. Він включає в себе засоби для отримання та встановлення об'єкта FilesView, який відображає вміст вкладки, а також можливість установки списку файлів для відображення.

package com.example.shell.fx.layouts.files.view.views;  
  
import com.example.shell.fx.layouts.files.view.views.Impl.FilesView;  
import javafx.scene.control.Tab;  
  
import java.io.File;  
import java.util.List;  
  
public class FilesViewTab extends Tab {  
 private static FilesView *filesView*;  
 private static List<File> *files*;  
  
 public static FilesView getFilesView() {  
 return *filesView*;  
 }  
  
 public static void setFilesView(FilesView filesView) {  
 FilesViewTab.*filesView* = filesView;  
 FilesViewTab.*filesView*.addFiles(*files*);  
 }  
  
 public static void setFiles(List<File> files) {  
 FilesViewTab.*files* = files;  
 }  
}

# Абстрактний клас ComboBoxTemplate:

Цей клас є абстрактним шаблоном, який визначає структуру створення та налаштування комбінованого меню (ComboBox) в JavaFX. Метод createMenu представляє шаблонний метод, який включає в себе виклики абстрактних методів для створення базового меню та його налаштування. Цей шаблон може бути використаний для створення різних видів комбінованих меню в рамках файлового менеджера.

package com.example.shell.fx.layouts.files.view.comboBoxes;  
  
import javafx.scene.control.ComboBox;  
  
import java.util.List;  
  
public abstract class ComboBoxTemplate {  
 // Шаблонний метод  
 public ComboBox<String> createMenu(List<String> menuItems) {  
 ComboBox<String> menu = createBaseMenu(menuItems);  
 customizeMenu(menu);  
 return menu;  
 }  
  
 // Абстрактний метод, який потрібно реалізувати в підкласах  
 protected abstract ComboBox<String> createBaseMenu(List<String> menuItems);  
  
 // Абстрактний метод для налаштування меню  
 protected abstract void customizeMenu(ComboBox<String> menu);  
}

# Клас DiskComboBoxTemplate:

Цей клас є конкретною реалізацією абстрактного класу ComboBoxTemplate і відповідає за створення та налаштування комбінованого меню (ComboBox) для вибору диска в контексті файлового менеджера. Метод createBaseMenu визначає базовий вигляд меню з переданим списком дисків, а customizeMenu додає додаткові налаштування, такі як текст-підказка ("Select Disk") та логіка обробки подій для визначення обраного диска у FileManager.Начало формы

package com.example.shell.fx.layouts.files.view.comboBoxes.Impl;  
  
import com.example.shell.fx.layouts.files.view.comboBoxes.ComboBoxTemplate;  
import com.example.shell.models.FileManager.FileManager;  
import javafx.scene.control.ComboBox;  
  
import java.util.List;  
  
public class DiskComboBoxTemplate extends ComboBoxTemplate {  
 @Override  
 protected ComboBox<String> createBaseMenu(List<String> disks) {  
 ComboBox<String> diskMenu = new ComboBox<>();  
 diskMenu.getItems().addAll(disks);  
 diskMenu.setValue(disks.get(0));  
 return diskMenu;  
 }  
  
 // Надаємо додаткові налаштування для меню вибору диска  
 @Override  
 protected void customizeMenu(ComboBox<String> menu) {  
 menu.setPromptText("Select Disk");  
  
 menu.setOnAction(event -> {  
 String selectedDisk = menu.getValue();  
 FileManager.*setCurrentDisk*(FileManager.*findDiskByName*(selectedDisk));  
 });  
 }  
}

# Інтерфейс ViewComboBoxTemplate:

Цей клас є конкретною реалізацією абстрактного класу ComboBoxTemplate і відповідає за створення та налаштування комбінованого меню (ComboBox) для вибору виду відображення файлів у файловому менеджері. Метод createBaseMenu визначає базовий вигляд меню з переданим списком видів, а customizeMenu додає додаткові налаштування, такі як текст-підказка ("Select View").Начало формы

package com.example.shell.fx.layouts.files.view.comboBoxes.Impl;  
  
import com.example.shell.fx.layouts.files.view.comboBoxes.ComboBoxTemplate;  
import javafx.scene.control.\*;  
  
import java.util.List;  
  
public class ViewComboBoxTemplate extends ComboBoxTemplate {  
 @Override  
 protected ComboBox<String> createBaseMenu(List<String> views) {  
 ComboBox<String> viewMenu = new ComboBox<>();  
 viewMenu.getItems().addAll(views);  
 viewMenu.setValue(views.get(0));  
 return viewMenu;  
 }  
  
 // Надаємо додаткові налаштування для меню вида  
 @Override  
 protected void customizeMenu(ComboBox<String> menu) {  
 menu.setPromptText("Select View");  
 }  
}

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