

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 №6

**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 «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» 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 «Factory Method»

## Структура:

## 

## Призначення:

Шаблон "фабричний метод" визначає інтерфейс для створення об'єктів певного базового типу. Це зручно, коли хочеться додати можливість створення об'єктів не базового типу, а деякого дочірнього. Фабричний метод у такому разі є зачіпкою для впровадження власного конструктора об'єктів. Основна ідея полягає саме в заміні об'єктів їх підтипами, що при цьому зберігає ту ж функціональність; інша частина поведінки об'єктів не є інтерфейсною (AnOperation) і дозволяє взаємодіяти із створеними об'єктами як з об'єктами базового типу. Тому шаблон "фабричний метод" носить ще назву "Віртуальний конструктор".

Розглянемо простий приклад. Нехай наше застосування працює з мережевими драйверамі і використовує клас Packet для зберігання даних, що передаються в мережу. Залежно від використовуваного протоколу, існує два перевантаження - TcpPacket, UdpPacket. І відповідно два створюючі об'єкти (TcpCreator, UdpCreator) з фабричним методом (який створює відповідні реалізації).

Проте базова функціональність (передача пакету, прийом пакету, заповнення пакету даними) нічим не відрізняється один від одного, відповідно поміщається у базовий клас PacketCreator. Таким чином поведінка системи залишається тим же, проте з'являється можливість підстановки власних об'єктів в процес створення і роботи з пакетами.

## Переваги та недоліки:

+ Позбавляє клас від прив’язки до конкретних класів продуктів.

+ Виділяє код виробництва продуктів в одне місце, спрощуючи підтримку коду.

+ Спрощує додавання нових продуктів до програми.

- Може призвести до створення великих паралельних ієрархій класів.

## Реалізація:

Шаблон «Factory Method» в моєму проекті надає гнучкість та можливість легко розширювати функціональність створення файлового перегляду. Базовий клас FilesViewFactory визначає метод createFilesView, який є фабричним методом для створення екземпляру класу View. Класи-спадкоємці, такі як ListFilesViewFactory і TableFilesViewFactory, реалізують цей метод, надаючи конкретні екземпляри View, такі як ListFilesView і TableFilesView.

Такий підхід дозволяє нам динамічно вибирати, який тип файлового перегляду використовувати в залежності від потреб додатка або налаштувань користувача. Зміна типу перегляду може бути здійснена легкою заміною фабричного методу, не впливаючи на інші частини системи.

# Стуктура проєкта:

# 

# Клас Main:

Цей клас є частиною додатку та використовується для створення графічного інтерфейсу користувацького файлового менеджера за допомогою JavaFX. Крім того цей клас використовує шаблон «Factory Method» для створення та відображення файлів списком і таблицею, створюючи окреме вікно з результатом.

package com.example.shell;  
  
import com.example.shell.fx.layouts.files.view.FilesViewFactory;  
import com.example.shell.fx.layouts.files.view.views.Impl.View;  
import com.example.shell.fx.layouts.files.view.views.ListFilesViewFactory;  
import com.example.shell.fx.layouts.files.view.views.TableFilesViewFactory;  
import com.example.shell.models.User;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Scene;  
import javafx.scene.control.Tab;  
import javafx.scene.control.TabPane;  
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;  
  
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 factory method  
 FilesViewFactory filesViewFactory;  
  
 List<File> testFiles = Arrays.*asList*(  
 new File("Folder1"),  
 new File("Folder2"),  
 new File("File1.txt"),  
 new File("File2.txt"),  
 new File("File3.txt")  
 );  
  
 filesViewFactory = new ListFilesViewFactory();  
 View listView = filesViewFactory.create();  
 Tab listViewTab = createFilesViewTab("List View", listView.getNode());  
 listView.display(testFiles);  
  
 filesViewFactory = new TableFilesViewFactory();  
 View tableView = filesViewFactory.create();  
 Tab tableViewTab = createFilesViewTab("Table View", tableView.getNode());  
 tableView.display(testFiles);  
  
 TabPane tabPane = new TabPane(listViewTab, tableViewTab);  
 Scene testScene = new Scene(tabPane, 640, 480);  
 stage.setScene(testScene);  
 stage.show();  
 }  
  
 private Tab createFilesViewTab(String title, Node node) {  
 Tab tab = new Tab(title);  
 BorderPane pane = new BorderPane();  
 pane.setCenter(node);  
 tab.setContent(pane);  
 return tab;  
 }  
}

# Абстрактний клас FilesViewFactory:

Цей клас є абстрактною фабрикою для створення різних видів файлових переглядів у додатку. Метод create() викликає абстрактний «фабричний» метод createFilesView(), що надає можливість конкретним фабрикам створювати різні види файлових переглядів.

package com.example.shell.fx.layouts.files.view;  
  
import com.example.shell.fx.layouts.files.view.views.Impl.View;  
  
public abstract class FilesViewFactory {  
 public View create() {  
 return createFilesView();  
 }  
  
 protected abstract View createFilesView();  
}

# Клас ListFilesViewFactory:

Цей клас представляє конкретну фабрику для створення файлового перегляду у вигляді списку. Він наслідує абстрактну фабрику FilesViewFactory і реалізує метод createFilesView(), який повертає новий екземпляр спискового файлового перегляду.

package com.example.shell.fx.layouts.files.view.views;  
  
import com.example.shell.fx.layouts.files.view.FilesViewFactory;  
import com.example.shell.fx.layouts.files.view.views.Impl.ListFilesView;  
import com.example.shell.fx.layouts.files.view.views.Impl.View;  
  
public class ListFilesViewFactory extends FilesViewFactory {  
 @Override  
 public View createFilesView() {  
 return new ListFilesView();  
 }  
}

# Клас TableFilesViewFactory:

Цей клас є конкретною фабрикою для створення файлового перегляду у вигляді таблиці. Він розширює абстрактну фабрику FilesViewFactory і реалізує метод createFilesView(), який створює та повертає новий екземпляр файлового перегляду у вигляді таблиці.

package com.example.shell.fx.layouts.files.view.views;  
  
import com.example.shell.fx.layouts.files.view.FilesViewFactory;  
import com.example.shell.fx.layouts.files.view.views.Impl.TableFilesView;  
import com.example.shell.fx.layouts.files.view.views.Impl.View;  
  
public class TableFilesViewFactory extends FilesViewFactory {  
 @Override  
 public View createFilesView() {  
 return new TableFilesView();  
 }  
}

# Інтерфейс FilesView:

Цей інтерфейс визначає основні методи для усіх видів файлових переглядів у додатку. Метод getNode() повертає вузол, який представляє графічний інтерфейс перегляду, а метод display(List<File> files) відображає переданий список файлів у цьому перегляді.

package com.example.shell.fx.layouts.files.view.views.Impl;  
  
import javafx.scene.Node;  
  
import java.io.File;  
import java.util.List;  
  
public interface FilesView {  
 Node getNode();  
  
 void display(List<File> files);  
}

# Клас ListFilesView:

Цей клас реалізує файловий перегляд у вигляді списку для графічного інтерфейсу додатка. Він використовує компонент ListView для відображення імен файлів та папок у переданому списку.

package com.example.shell.fx.layouts.files.view.views.Impl;  
  
import javafx.scene.Node;  
import javafx.scene.control.ListView;  
  
import java.io.File;  
import java.util.List;  
  
public class ListFilesView implements FilesView {  
 private final ListView<String> node = new ListView<>();  
  
 @Override  
 public Node getNode() {  
 return node;  
 }  
  
 @Override  
 public void display(List<File> files) {  
 node.getItems().clear();  
 for (File file : files) {  
 node.getItems().add(file.getName());  
 }  
 }  
}

# Клас TableFilesView:

Цей клас відповідає за відображення файлового перегляду у вигляді таблиці для графічного інтерфейсу додатка. Кожен файл чи папка представлений блоком з ім'ям, яке виводиться в таблиці, розміщеною по 4 елемента в кожному рядку. Блоки відображаються у вертикальному та горизонтальному форматі, розміщені на рівномірному відстані один від одного, займаючи всю доступну область.

package com.example.shell.fx.layouts.files.view.views.Impl;  
  
import javafx.geometry.Insets;  
import javafx.scene.Node;  
import javafx.scene.control.Label;  
import javafx.scene.layout.\*;  
  
import java.io.File;  
import java.util.List;  
import java.util.stream.IntStream;  
  
public class TableFilesView implements FilesView {  
 private final VBox node = new VBox();  
 private final int filesPerRow = 4;  
  
 @Override  
 public Node getNode() {  
 return node;  
 }  
  
 @Override  
 public void display(List<File> files) {  
 // Очищуємо дочірні елементи поточного вузла (VBox)  
 node.getChildren().clear();  
  
 // Створюємо вертикальний контейнер для групування рядків  
 final HBox[] rowBox = {new HBox()};  
 rowBox[0].setSpacing(10); // Відступ між рядками  
  
 // Створюємо горизонтальний контейнер для групування блоків у кожному рядку  
 VBox columnBox = new VBox();  
 columnBox.setSpacing(10); // Відступ між блоками в рядку  
  
 // Проходимо по списку файлів  
 IntStream.*range*(0, files.size()).forEach(i -> {  
 File file = files.get(i);  
  
 // Створюємо блок для файлу і додаємо його в поточний рядок  
 rowBox[0].getChildren().add(createFileBlockFromFile(file));  
  
 // Якщо досягли кінця рядку або останнього елемента,  
 // додаємо рядок у вертикальний контейнер  
 if ((i + 1) % filesPerRow == 0 || i == files.size() - 1) {  
 columnBox.getChildren().add(rowBox[0]); // Додаємо рядок у контейнер рядків  
 rowBox[0] = new HBox(); // Створюємо новий рядок  
 rowBox[0].setSpacing(10); // Встановлюємо відступ для нового рядка  
 }  
 });  
  
 // Додаємо вертикальний контейнер у основний контейнер  
 node.getChildren().add(columnBox);  
 }  
  
 private Region createFileBlockFromFile(File file) {  
 // Створюємо горизонтальний контейнер для блоку файлу  
 HBox fileBlock = new HBox();  
 fileBlock.setMinSize(0, 0); // Встановлюємо мінімальний розмір блоку  
 HBox.*setHgrow*(fileBlock, Priority.*ALWAYS*); // Встановлюємо горизонтальне розширення блоку  
  
 // Створюємо напис з ім'ям файлу і додаємо його в блок  
 Label label = new Label(file.getName());  
 label.setPadding(new Insets(5)); // Встановлюємо внутрішній відступ для напису  
  
 fileBlock.getChildren().add(label); // Додаємо напис в блок  
 return fileBlock; // Повертаємо створений блок  
 }  
}

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