МИНИСТЕРСТВО ТРАНСПОРТА РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА (МИИТ)»

Кафедра ЦТУТП

**Отчёт**

По лабораторной работе №2  
по дисциплине «Корпоративные информационные системы»

Тема: «Декоратор»

Вариант №14

Выполнил: Сафонов Г.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc180329255)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc180329256)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 6](#_Toc180329257)

[КОД ПРОГРАММЫ 7](#_Toc180329258)

[ВЫВОД 11](#_Toc180329259)

# **ПОСТАНОВКА ЗАДАЧИ**

Создать интерфейс ОкноПрограммы. Создать класс ПростоеОкноПрограммы, содержащий основные свойства программного окна. Создать декоратор ОкноПрограммыСМеню, дополняющий класс ПростоеОкноПрограммы меню. Создать декоратор ОкноПрограммыСоСтрокойСтатуса, дополняющий класс ПростоеОкноПрограммы строкой статуса. Создать декоратор ОкноПрограммыСБоковушкой, дополняющий класс ПростоеОкноПрограммы боковой панелью.  Продемонстрировать операции с различным набором декораторов.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

Интерфейс IFrameFactory служит основой для создания объектов типа JFrame. Он определяет метод createFrame(FrameParameters parameters), который будет реализован в других классах для создания и настройки фреймов. Этот интерфейс обеспечивает гибкость и возможность замены конкретной реализации фрейма без изменения кода, который его использует.

Класс FrameFactory реализует интерфейс IFrameFactory и отвечает за создание экземпляров JFrame. В методе createFrame он инициализирует фрейм, устанавливает его заголовок, размеры, видимость и поведение при закрытии. Это обеспечивает базовую функциональность для создания фрейма, который может быть дополнительно украшен декораторами.

Класс FrameFactoryBean представляет собой обертку для IFrameFactory, которая управляет инъекцией зависимости. Он инжектирует экземпляр IFrameFactory и предоставляет метод createFrame, который создает фрейм на основе переданных параметров. Это позволяет изолировать логику создания фрейма от остальной части приложения, делая код более чистым и поддерживаемым.

Класс FrameParameters используется для хранения всех необходимых параметров, которые требуются для создания фрейма, таких как заголовок, размеры, видимость и поведение при закрытии. Он использует аннотации @Getter, @Setter, @AllArgsConstructor и @NoArgsConstructor из библиотеки Lombok для автоматизации генерации методов доступа и конструкторов, упрощая работу с данными.

Декораторы FrameWithMenuDecorator, FrameWithSidebarDecorator и FrameWithStatusBarDecorator добавляют дополнительную функциональность к базовому фрейму. Каждый из них реализует интерфейс IFrameFactory и использует делегирование для вызова метода createFrame из обернутого экземпляра IFrameFactory. Внутри каждого декоратора создаются дополнительные компоненты, такие как меню, боковая панель и строка состояния, которые добавляются в фрейм после его создания.

В методе addMenu первого декоратора создается меню с несколькими пунктами, которые реагируют на действия пользователя с помощью ActionListener. Второй декоратор добавляет боковую панель с кнопками, каждая из которых также имеет свою обработку событий. Третий декоратор создает строку состояния в нижней части фрейма, информируя пользователя о текущем состоянии приложения.

Наконец, в классе Laboratory2 происходит инициализация CDI-контейнера Weld, который управляет созданием и инъекцией зависимостей. В методе main создаются параметры для окна, а затем вызывается метод createFrame через FrameFactoryBean для создания и отображения готового фрейма с добавленной функциональностью, определенной декораторами.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

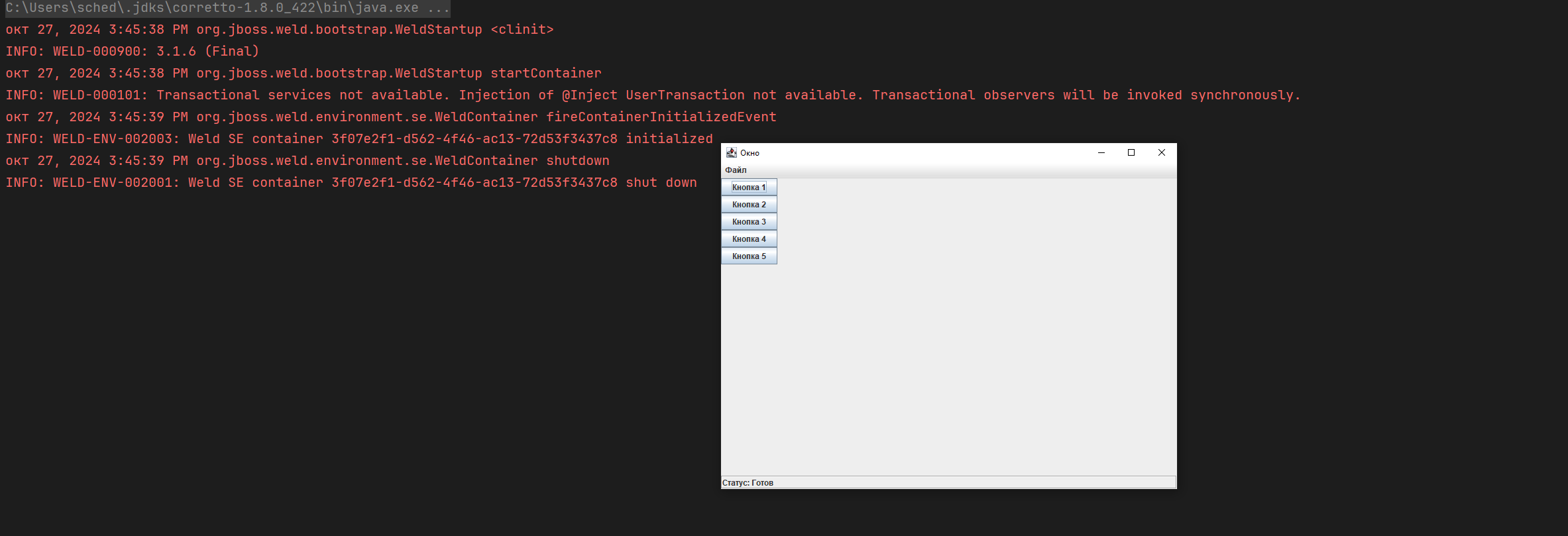


Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package Lab2\_Decorator.Decorator;  
  
import Lab2\_Decorator.FrameFactory.IFrameFactory;  
import Lab2\_Decorator.FrameFactory.Parameter.FrameParameters;  
import lombok.AllArgsConstructor;  
import lombok.NoArgsConstructor;  
  
import javax.decorator.Decorator;  
import javax.decorator.Delegate;  
import javax.inject.Inject;  
import javax.swing.\*;  
  
@Decorator  
@AllArgsConstructor  
@NoArgsConstructor  
public class FrameWithMenuDecorator implements IFrameFactory {  
  
 @Inject  
 @Delegate  
 private IFrameFactory frameFactory;  
  
 private void addMenu(JFrame frame) {  
 JMenuBar menuBar = new JMenuBar();  
  
 // Создание меню "Файл"  
 JMenu fileMenu = new JMenu("Файл");  
  
 // Добавление пунктов в меню  
 for (int i = 1; i <= 5; i++) {  
 // Можно добавить действия для каждого пункта, например, вывод сообщения  
 final int menuIndex = i;  
 JMenuItem menuItem = new JMenuItem("Пункт " + menuIndex);  
 menuItem.addActionListener(e -> JOptionPane.showMessageDialog(frame, "Вы выбрали Пункт " + menuIndex));  
 fileMenu.add(menuItem);  
 }  
  
 // Добавление пункта выхода  
 JMenuItem exitItem = new JMenuItem("Выход");  
 exitItem.addActionListener(e -> System.exit(0));  
 fileMenu.add(exitItem);  
  
 // Добавление меню в панель меню  
 menuBar.add(fileMenu);  
 frame.setJMenuBar(menuBar);  
 }  
  
 @Override  
 public JFrame createFrame(FrameParameters parameters) {  
 JFrame frame = frameFactory.createFrame(parameters);  
 addMenu(frame);  
 return frame;  
 }  
}

package Lab2\_Decorator.Decorator;  
  
import Lab2\_Decorator.FrameFactory.IFrameFactory;  
import Lab2\_Decorator.FrameFactory.Parameter.FrameParameters;  
import lombok.AllArgsConstructor;  
import lombok.NoArgsConstructor;  
  
import javax.decorator.Decorator;  
import javax.decorator.Delegate;  
import javax.inject.Inject;  
import javax.swing.\*;  
import java.awt.\*;  
  
@Decorator  
@AllArgsConstructor  
@NoArgsConstructor  
public class FrameWithSidebarDecorator implements IFrameFactory {  
  
 @Inject  
 @Delegate  
 private IFrameFactory frameFactory;  
  
 private void addSidebar(JFrame frame) {  
 // Создание боковой панели  
 JPanel sidebar = new JPanel();  
 sidebar.setLayout(new BoxLayout(sidebar, BoxLayout.Y\_AXIS)); // Вертикальное расположение  
  
 // Добавление кнопок в боковую панель  
 for (int i = 1; i <= 5; i++) {  
 final int menuIndex = i;  
 JButton button = new JButton("Кнопка " + menuIndex);  
 button.addActionListener(e -> JOptionPane.showMessageDialog(frame, "Вы нажали Кнопку " + menuIndex));  
 sidebar.add(button);  
 }  
  
 // Добавление боковой панели к фрейму  
 frame.getContentPane().add(sidebar, BorderLayout.WEST); // Боковая панель слева  
 }  
  
 @Override  
 public JFrame createFrame(FrameParameters parameters) {  
 JFrame frame = frameFactory.createFrame(parameters);  
 addSidebar(frame);  
 return frame;  
 }  
}

package Lab2\_Decorator.Decorator;  
  
import Lab2\_Decorator.FrameFactory.IFrameFactory;  
import Lab2\_Decorator.FrameFactory.Parameter.FrameParameters;  
import lombok.AllArgsConstructor;  
import lombok.NoArgsConstructor;  
  
import javax.decorator.Decorator;  
import javax.decorator.Delegate;  
import javax.inject.Inject;  
import javax.swing.\*;  
import java.awt.\*;  
  
@Decorator  
@AllArgsConstructor  
@NoArgsConstructor  
public class FrameWithStatusBarDecorator implements IFrameFactory {  
  
 @Inject  
 @Delegate  
 private IFrameFactory frameFactory;  
  
 private void addStatusBar(JFrame frame) {  
 // Создание строки статуса  
 JLabel statusBar = new JLabel("Статус: Готов");  
 statusBar.setBorder(BorderFactory.createEtchedBorder());  
  
 // Добавление строки статуса в нижнюю часть окна  
 frame.getContentPane().add(statusBar, BorderLayout.SOUTH);  
 }  
  
 @Override  
 public JFrame createFrame(FrameParameters parameters) {  
 JFrame frame = frameFactory.createFrame(parameters);  
 addStatusBar(frame);  
 return frame;  
 }  
}

package Lab2\_Decorator.FrameFactory.Parameter;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
public class FrameParameters {  
 private String title;  
 private int height;  
 private int width;  
 private boolean isVisible;  
 private int closeOperation;  
  
}

package Lab2\_Decorator.FrameFactory;  
  
import Lab2\_Decorator.FrameFactory.Parameter.FrameParameters;  
import lombok.NoArgsConstructor;  
  
import javax.enterprise.inject.Default;  
import javax.swing.\*;  
  
@Default  
@NoArgsConstructor  
public class FrameFactory implements IFrameFactory {  
  
 @Override  
 public JFrame createFrame(FrameParameters params) {  
 JFrame frame = new JFrame(params.getTitle());  
  
 frame.setSize(params.getWidth(), params.getHeight());  
 frame.setDefaultCloseOperation(params.getCloseOperation());  
 frame.setVisible(params.isVisible());  
  
 return frame;  
 }  
}

package Lab2\_Decorator.FrameFactory;  
  
import Lab2\_Decorator.FrameFactory.Parameter.FrameParameters;  
  
import javax.inject.Inject;  
import javax.swing.\*;  
  
public class FrameFactoryBean {  
  
 @Inject  
 private IFrameFactory frameFactory;  
  
 public JFrame createFrame(FrameParameters frameParameters) {  
 return frameFactory.createFrame(frameParameters);  
 }  
}

package Lab2\_Decorator.FrameFactory;  
  
import Lab2\_Decorator.FrameFactory.Parameter.FrameParameters;  
  
import javax.swing.\*;  
  
public interface IFrameFactory {  
 JFrame createFrame(FrameParameters parameters);  
}

package Lab2\_Decorator;  
  
import Lab2\_Decorator.FrameFactory.FrameFactoryBean;  
import Lab2\_Decorator.FrameFactory.Parameter.FrameParameters;  
import org.jboss.weld.environment.se.Weld;  
import org.jboss.weld.environment.se.WeldContainer;  
  
import javax.swing.\*;  
  
public class Laboratory2 {  
 protected static Weld weld;  
 protected static WeldContainer container;  
  
 public static void main(String[] args) {  
 weld = new Weld();  
 container = weld.initialize();  
  
 try {  
 FrameFactoryBean frameFactoryBean = container.select(FrameFactoryBean.class).get();  
  
 // Создание параметров для окна  
 FrameParameters params = new FrameParameters("Окно", 400, 300, true, JFrame.EXIT\_ON\_CLOSE);  
  
 // Создание окна  
 JFrame frame = frameFactoryBean.createFrame(params);  
  
 } finally {  
 weld.shutdown();  
 }  
 }  
  
}

# **ВЫВОД**

В ходе выполнения лабораторной работы было разработано приложение с использованием паттерна проектирования **Декоратор** для создания графического интерфейса на Java с помощью библиотеки Swing. Реализованные классы и декораторы продемонстрировали возможность динамического добавления функциональности к базовым элементам интерфейса без изменения их исходного кода.

В результате работы были созданы несколько классов, отвечающих за разные аспекты интерфейса, включая создание фреймов, добавление меню, боковых панелей и строк состояния. Использование интерфейса IFrameFactory обеспечило гибкость и расширяемость приложения, что позволяет легко добавлять новые декораторы или изменять существующие.

Также была использована библиотека Lombok для упрощения кода, что сделало его более читаемым и поддерживаемым. Инъекция зависимостей через CDI (Contexts and Dependency Injection) позволила изолировать логику создания фреймов и улучшить тестируемость приложения.

Таким образом, лабораторная работа показала, как можно эффективно применять паттерны проектирования и принципы объектно-ориентированного программирования для создания гибких и масштабируемых приложений, что является важным навыком в современном программировании.