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 №12

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 9](#_Toc180329259)

# **ПОСТАНОВКА ЗАДАЧИ**

Создать интерфейс Автомобиль. Создать класс ПростойАвтомобиль, содержащий основные свойства автомобиля. Создать декоратор СпортивныйАвтомобиль, дополняющий класс ПростойАвтомобиль свойствами спортивного. Создать декоратор ГрузовойАвтомобиль, дополняющий класс ПростойАвтомобиль свойствами грузовика. Продемонстрировать операции с различным набором декораторов.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В процессе выполнения лабораторной работы была разработана система для описания различных типов автомобилей с использованием паттерна "Декоратор" и технологии CDI (Contexts and Dependency Injection). Работа сосредоточена на создании гибкой архитектуры, позволяющей добавлять новые функции и свойства к объектам без изменения их базового кода.

Работа началась с создания интерфейса Car, который содержит метод getDescription(). Этот метод предназначен для предоставления текстового описания автомобиля. Интерфейс является основой для всех реализаций автомобиля, что обеспечивает единый контракт для получения информации о машине.

Затем был реализован класс SimpleCar, который реализует интерфейс Car. Этот класс представляет собой простую модель автомобиля с фиксированными параметрами: марка (BMW), модель (M3) и год выпуска (2018). Метод getDescription() формирует строку, описывающую автомобиль, и возвращает ее.

В классе CarBean используется аннотация @Inject для внедрения экземпляра Car. Метод getDescription() в этом классе вызывает метод getDescription() у инъектированного объекта Car, что позволяет получать описание автомобиля.

Основной частью работы являются декораторы, реализованные в классах SportsCar и TruckCar. Эти классы расширяют функциональность объектов, реализующих интерфейс Car, добавляя к ним дополнительные характеристики. Например, класс SportsCar добавляет информацию о максимальной скорости и мощности двигателя. Декоратор TruckCar добавляет информацию о грузоподъемности.

Оба декоратора используют аннотацию @Delegate, чтобы делегировать вызовы методов объекту Car, который они оборачивают. Таким образом, декораторы не заменяют, а дополняют функциональность оригинального объекта, обеспечивая возможность динамического добавления новых характеристик.

В классе Laboratory2 инициализируется CDI-контейнер с помощью библиотеки Weld. Затем создается экземпляр класса CarBean, который автоматически получает инъекцию всех необходимых зависимостей, включая декораторы. Вызов метода getDescription() у carBean выводит на экран полное описание автомобиля, включая все добавленные декораторами параметры.

Таким образом, лабораторная работа продемонстрировала, как использовать паттерн "Декоратор" вместе с CDI для создания модульной и расширяемой архитектуры. Это подход позволяет эффективно управлять сложностью системы и упрощает внесение изменений в функциональность приложения.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

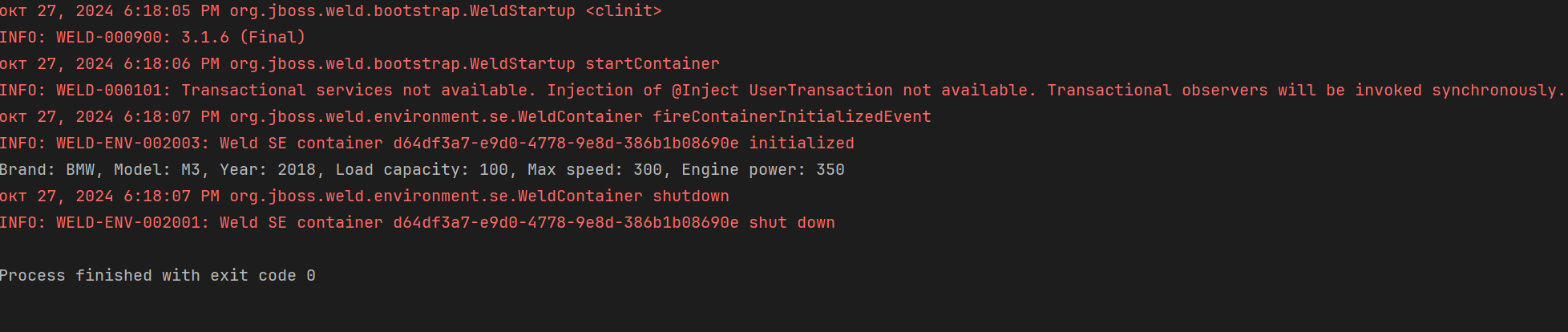


Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package Lab2\_Decorator.Car;  
  
public interface Car {  
 String getDescription();  
}

package Lab2\_Decorator.Car;  
  
import javax.inject.Inject;  
  
public class CarBean {  
  
 @Inject  
 private Car car;  
  
 public String getDescription() {  
 return car.getDescription();  
 }  
}

package Lab2\_Decorator.Car;  
  
import lombok.NoArgsConstructor;  
  
import javax.enterprise.inject.Default;  
  
@Default  
@NoArgsConstructor  
public class SimpleCar implements Car {  
 @Override  
 public String getDescription() {  
 int year = 2018;  
 String model = "M3";  
 String brand = "BMW";  
 return "Brand: " + brand + ", Model: " + model + ", Year: " + year;  
 }  
  
}

package Lab2\_Decorator.Decorator;  
import Lab2\_Decorator.Car.Car;  
import lombok.AllArgsConstructor;  
import lombok.NoArgsConstructor;  
import javax.decorator.Decorator;  
import javax.decorator.Delegate;  
import javax.inject.Inject;  
@Decorator  
@AllArgsConstructor  
@NoArgsConstructor  
public class SportsCar implements Car {  
 private int maxSpeed = 300;  
 private int enginePower = 350;  
  
 @Inject  
 @Delegate  
 private Car car;  
  
 @Override  
 public String getDescription() {  
 return car.getDescription() + ", Max speed: " + maxSpeed + ", Engine power: " + enginePower;  
 }  
}

package Lab2\_Decorator.Decorator;  
  
import Lab2\_Decorator.Car.Car;  
import lombok.AllArgsConstructor;  
import lombok.NoArgsConstructor;  
  
import javax.decorator.Decorator;  
import javax.decorator.Delegate;  
import javax.inject.Inject;  
  
@Decorator  
@AllArgsConstructor  
@NoArgsConstructor  
public class TruckCar implements Car {  
  
 private int loadCapacity = 100;  
 @Delegate  
 @Inject  
 private Car car;  
  
 @Override  
 public String getDescription() {  
 return car.getDescription() + ", Load capacity: " + loadCapacity;  
 }  
}

package Lab2\_Decorator;  
  
import Lab2\_Decorator.Car.CarBean;  
import org.jboss.weld.environment.se.Weld;  
import org.jboss.weld.environment.se.WeldContainer;  
  
public class Laboratory2 {  
 protected static Weld weld;  
 protected static WeldContainer container;  
  
 public static void main(String[] args) {  
 weld = new Weld();  
 container = weld.initialize();  
  
 try {  
 CarBean carBean = container.select(CarBean.class).get();  
  
 System.out.println(carBean.getDescription());  
  
 } finally {  
 weld.shutdown();  
 }  
 }  
  
}

# **ВЫВОД**

В ходе выполнения лабораторной работы была успешно разработана система для описания различных автомобилей с использованием паттерна "Декоратор" и технологий CDI (Contexts and Dependency Injection). Реализация продемонстрировала важные принципы объектно-ориентированного проектирования, такие как инкапсуляция, модульность и гибкость. Использование паттерна "Декоратор" позволило динамически добавлять новые свойства к базовым объектам автомобиля без изменения их исходного кода, что делает систему более устойчивой к изменениям и легкой для расширения.

Технология CDI упростила управление зависимостями в приложении, позволяя автоматически внедрять необходимые компоненты и снижая количество "кода связывания". Это, в свою очередь, упрощает тестирование и повышает читаемость кода. Классы декораторов аккуратно разделяют различные аспекты описания автомобилей, что делает код более читаемым и поддерживаемым. Каждое изменение или добавление нового функционала возможно без вмешательства в существующие классы.

В работе был успешно реализован паттерн "Декоратор", который продемонстрировал, как эффективно применять данный подход для обогащения поведения объектов. Классы SportsCar и TruckCar добавили специфические характеристики к базовому классу SimpleCar, что показало возможности декорирования. В результате, лабораторная работа не только продемонстрировала значимость использования паттернов проектирования и технологий инъекции зависимостей, но и подготовила к созданию гибких и расширяемых систем в