МИНИСТЕРСТВО ТРАНСПОРТА РФ

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

Кафедра ЦТУТП

**Отчёт**

По лабораторной работе №3  
по дисциплине «Корпоративные информационные системы»

Тема: «События»

Вариант №4

Выполнил: Иванов Н. С.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc183976078)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc183976079)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 6](#_Toc183976080)

[КОД ПРОГРАММЫ 7](#_Toc183976081)

[ВЫВОД 14](#_Toc183976082)

# **ПОСТАНОВКА ЗАДАЧИ**

Разработать систему «Автобаза». Диспетчер распределяет заявки на Рейсы между Водителями и назначает для этого Автомобиль. Водитель может сделать заявку на ремонт. Диспетчер может отстранить Водителя от работы. Водитель делает отметку о выполнении Рейса и состоянии Автомобиля. Между классами могут передаваться разные сообщения.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В лабораторной работе реализована событийная система управления заявками на грузовые рейсы, ремонт автомобилей и статус водителей. Используется CDI (Contexts and Dependency Injection) для генерации и обработки событий с помощью пользовательских аннотаций.

Ключевые аннотации (@AddedFreightTripApplication, @SuspendedDriver, @AddedRepairApplication, @EditedFreightTripApplication, @EditedAutomobileStatus) определяют типы событий. Каждая аннотация связана с соответствующими обработчиками в классе EventListener, который обрабатывает события и выполняет действия, такие как вывод информации о рейсе, водителе или состоянии автомобиля.

Основная логика приложения разделена между двумя сервисами: DispatcherService и DriverService. DispatcherService отвечает за назначение рейсов и отстранение водителей, а DriverService управляет обновлением статусов рейсов, завершением рейсов и подачей заявок на ремонт. Все изменения сопровождаются генерацией событий, которые затем обрабатываются подписчиками.

Модель данных включает классы Driver, Automobile, FreightTripApplication и RepairApplication. Они представляют водителей, автомобили, заявки на рейсы и ремонты соответственно. Дополнительно используются перечисления AutomobileStatus и FreightTripStatus для определения состояния автомобилей и статусов рейсов.

Программа начинается с инициализации CDI-контейнера Weld, через который создаются экземпляры служб. В сценарии демонстрируются основные операции: создание заявки на рейс, изменение и завершение рейса, подача заявки на ремонт автомобиля и отстранение водителя. Каждый шаг генерирует соответствующее событие, которое обрабатывается подписчиками.

Обработчики событий в классе EventListener получают данные о событиях и выводят информацию в консоль. Например, при добавлении рейса отображается информация о водителе, автомобиле и статусе рейса. Аналогично при подаче заявки на ремонт выводятся данные о водителе и автомобиле.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

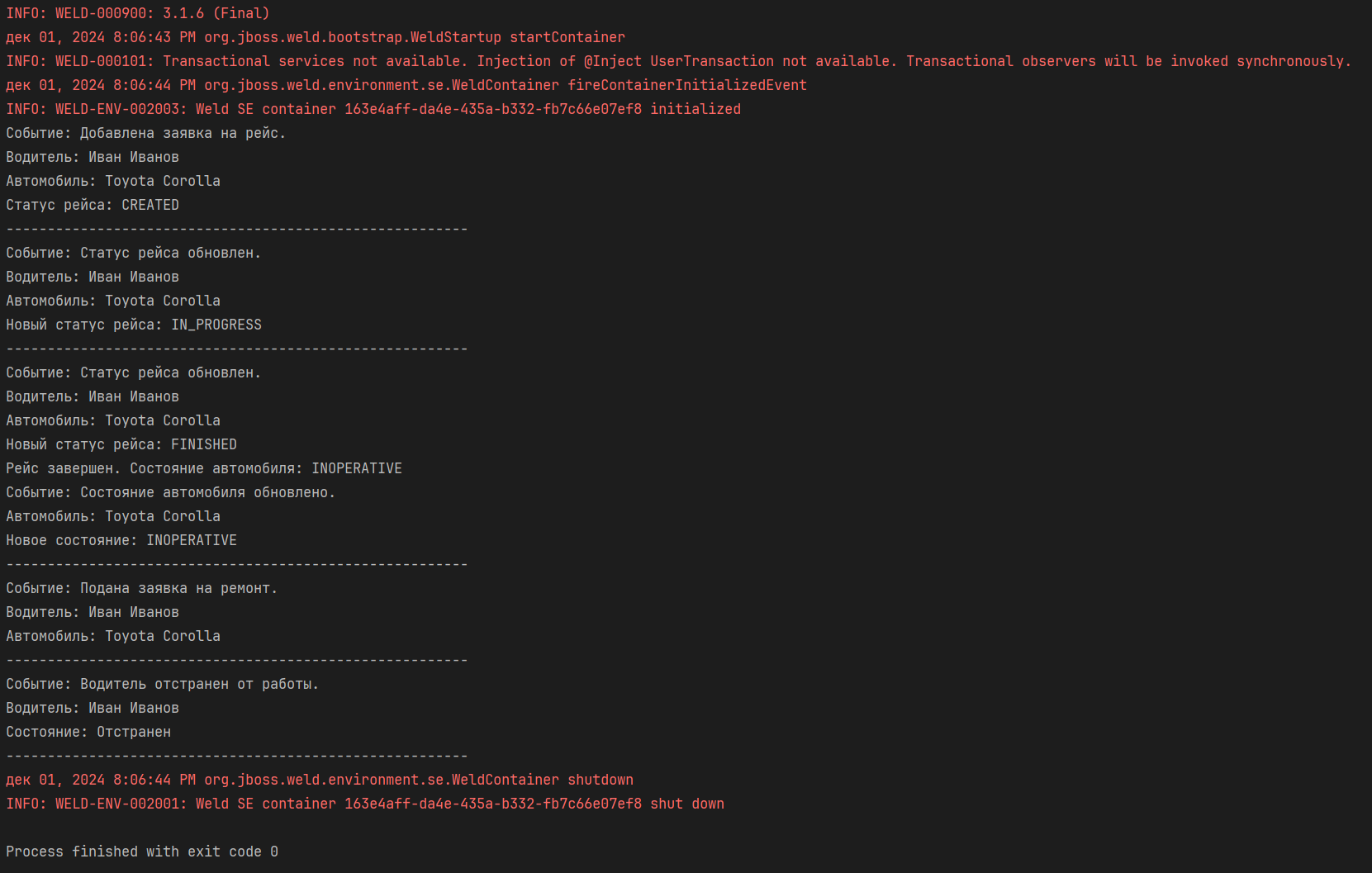


Рисунок 2 – Результат работы программы

# **КОД ПРОГРАММЫ**

package Lab3\_Event.Annotation;  
  
import javax.inject.Qualifier;  
import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Qualifier  
@Retention(RetentionPolicy.RUNTIME)  
@Target({ElementType.FIELD, ElementType.TYPE, ElementType.METHOD, ElementType.PARAMETER})  
public @interface AddedFreightTripApplication {  
}

package Lab3\_Event.Annotation;  
  
import javax.inject.Qualifier;  
import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Qualifier  
@Retention(RetentionPolicy.RUNTIME)  
@Target({ElementType.FIELD, ElementType.TYPE, ElementType.METHOD, ElementType.PARAMETER})  
public @interface AddedRepairApplication {  
}

package Lab3\_Event.Annotation;  
  
import javax.inject.Qualifier;  
import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Qualifier  
@Retention(RetentionPolicy.RUNTIME)  
@Target({ElementType.FIELD, ElementType.TYPE, ElementType.METHOD, ElementType.PARAMETER})  
public @interface EditedAutomobileStatus {  
}

package Lab3\_Event.Annotation;  
  
import javax.inject.Qualifier;  
import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Qualifier  
@Retention(RetentionPolicy.RUNTIME)  
@Target({ElementType.FIELD, ElementType.TYPE, ElementType.METHOD, ElementType.PARAMETER})  
public @interface EditedFreightTripApplication {  
}

package Lab3\_Event.Annotation;  
  
import javax.inject.Qualifier;  
import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Qualifier  
@Retention(RetentionPolicy.RUNTIME)  
@Target({ElementType.FIELD, ElementType.TYPE, ElementType.METHOD, ElementType.PARAMETER})  
public @interface SuspendedDriver {  
}

package Lab3\_Event.Listener;  
  
import Lab3\_Event.Annotation.\*;  
import Lab3\_Event.Model.\*;  
  
import javax.enterprise.event.Observes;  
  
public class EventListener {  
  
 /\*\*  
 \* Обработчик события добавления заявки на рейс.  
 \*  
 \* @param trip объект FreightTripApplication, связанный с добавленным рейсом.  
 \*/  
 public void onFreightTripAdded(@Observes @AddedFreightTripApplication FreightTripApplication trip) {  
 System.out.println("Событие: Добавлена заявка на рейс.");  
 System.out.println("Водитель: " + trip.getDriver().getName() + " " + trip.getDriver().getSurname());  
 System.out.println("Автомобиль: " + trip.getAutomobile().getMark() + " " + trip.getAutomobile().getModel());  
 System.out.println("Статус рейса: " + trip.getStatus());  
 }  
  
 /\*\*  
 \* Обработчик события отстранения водителя.  
 \*  
 \* @param driver объект Driver, связанный с отстраненным водителем.  
 \*/  
 public void onDriverSuspended(@Observes @SuspendedDriver Driver driver) {  
 System.out.println("Событие: Водитель отстранен от работы.");  
 System.out.println("Водитель: " + driver.getName() + " " + driver.getSurname());  
 System.out.println("Состояние: " + (driver.isSuspendedFromWork() ? "Отстранен" : "Активен"));  
 }  
  
 /\*\*  
 \* Обработчик события добавления заявки на ремонт.  
 \*  
 \* @param repairApplication объект RepairApplication, связанный с заявкой на ремонт.  
 \*/  
 public void onRepairApplicationAdded(@Observes @AddedRepairApplication RepairApplication repairApplication) {  
 System.out.println("Событие: Подана заявка на ремонт.");  
 System.out.println("Водитель: " + repairApplication.getDriver().getName() + " " + repairApplication.getDriver().getSurname());  
 System.out.println("Автомобиль: " + repairApplication.getAutomobile().getMark() + " " + repairApplication.getAutomobile().getModel());  
 }  
  
 /\*\*  
 \* Обработчик события изменения статуса рейса.  
 \*  
 \* @param trip объект FreightTripApplication, связанный с измененным рейсом.  
 \*/  
 public void onFreightTripStatusEdited(@Observes @EditedFreightTripApplication FreightTripApplication trip) {  
 System.out.println("Событие: Статус рейса обновлен.");  
 System.out.println("Водитель: " + trip.getDriver().getName() + " " + trip.getDriver().getSurname());  
 System.out.println("Автомобиль: " + trip.getAutomobile().getMark() + " " + trip.getAutomobile().getModel());  
 System.out.println("Новый статус рейса: " + trip.getStatus());  
 }  
  
 /\*\*  
 \* Обработчик события изменения состояния автомобиля.  
 \*  
 \* @param automobile объект Automobile, связанный с обновленным состоянием.  
 \*/  
 public void onAutomobileStatusEdited(@Observes @EditedFreightTripApplication Automobile automobile) {  
 System.out.println("Событие: Состояние автомобиля обновлено.");  
 System.out.println("Автомобиль: " + automobile.getMark() + " " + automobile.getModel());  
 System.out.println("Новое состояние: " + automobile.getStatus());  
 }  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Automobile {  
 private Integer id;  
 private String mark;  
 private String model;  
 private AutomobileStatus status;  
}

package Lab3\_Event.Model;  
  
public enum AutomobileStatus {  
 OPERATIVE,  
 INOPERATIVE  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Driver {  
 private Integer id;  
 private String name;  
 private String surname;  
 private boolean isSuspendedFromWork;  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class FreightTripApplication {  
 private Driver driver;  
 private Automobile automobile;  
 private FreightTripStatus status;  
}

package Lab3\_Event.Model;  
  
public enum FreightTripStatus {  
 CREATED,  
 IN\_PROGRESS,  
 FINISHED,  
 FAILED  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class RepairApplication {  
 private Driver driver;  
 private Automobile automobile;  
}

package Lab3\_Event.Service;  
  
import Lab3\_Event.Annotation.AddedFreightTripApplication;  
import Lab3\_Event.Annotation.SuspendedDriver;  
import Lab3\_Event.Model.\*;  
  
import javax.enterprise.event.Event;  
import javax.inject.Inject;  
  
public class DispatcherService {  
  
 // Событие, связанное с добавлением заявки на рейс  
 @Inject  
 @AddedFreightTripApplication  
 private Event<FreightTripApplication> tripAddedEvent;  
  
 // Событие, связанное с отстранением водителя  
 @Inject  
 @SuspendedDriver  
 private Event<Driver> driverSuspendedEvent;  
  
 /\*\*  
 \* Назначить рейс для водителя и автомобиля.  
 \*  
 \* @param driver водитель, который будет выполнять рейс.  
 \* @param automobile автомобиль, который будет использоваться в рейсе.  
 \* @return объект FreightTripApplication, представляющий рейс.  
 \* @throws IllegalStateException если водитель отстранен от работы.  
 \*/  
 public FreightTripApplication assignTrip(Driver driver, Automobile automobile) {  
 if (driver.isSuspendedFromWork()) {  
 throw new IllegalStateException("Водитель отстранен от работы!");  
 }  
 // Создание новой заявки на рейс со статусом "CREATED"  
 FreightTripApplication trip = new FreightTripApplication(driver, automobile, FreightTripStatus.CREATED);  
 // Генерация события о добавлении рейса  
 tripAddedEvent.fire(trip);  
 return trip;  
 }  
  
 /\*\*  
 \* Отстранить водителя от работы.  
 \*  
 \* @param driver водитель, который будет отстранен.  
 \*/  
 public void suspendDriver(Driver driver) {  
 // Установка флага "отстранен от работы"  
 driver.setSuspendedFromWork(true);  
 // Генерация события об отстранении водителя  
 driverSuspendedEvent.fire(driver);  
 }  
}

package Lab3\_Event.Service;  
  
import Lab3\_Event.Annotation.AddedRepairApplication;  
import Lab3\_Event.Annotation.EditedFreightTripApplication;  
import Lab3\_Event.Model.\*;  
  
import javax.enterprise.event.Event;  
import javax.inject.Inject;  
  
public class DriverService {  
  
 @Inject  
 @AddedRepairApplication  
 private Event<RepairApplication> addedRepairApplicationEvent;  
  
 @Inject  
 @EditedFreightTripApplication  
 private Event<FreightTripApplication> editedFreightTripApplicationEvent;  
  
 @Inject  
 @EditedFreightTripApplication  
 private Event<Automobile> editedAutomobileStatusEvent;  
  
 /\*\*  
 \* Отправить заявку на ремонт.  
 \* @param automobile автомобиль, нуждающийся в ремонте.  
 \* @param driver водитель, подающий заявку.  
 \*/  
 public void submitRepairApplication(Automobile automobile, Driver driver) {  
 RepairApplication repairApplication = new RepairApplication(driver, automobile);  
 addedRepairApplicationEvent.fire(repairApplication); // Генерация события о создании заявки на ремонт  
 }  
  
 /\*\*  
 \* Обновить статус рейса.  
 \* @param trip рейс, который нужно обновить.  
 \* @param newStatus новый статус рейса.  
 \*/  
 public void updateFreightTripStatus(FreightTripApplication trip, FreightTripStatus newStatus) {  
 trip.setStatus(newStatus);  
 editedFreightTripApplicationEvent.fire(trip); // Генерация события об обновлении статуса рейса  
 }  
  
 /\*\*  
 \* Завершить рейс.  
 \* @param trip завершенный рейс.  
 \* @param automobile автомобиль, участвовавший в рейсе.  
 \* @param automobileStatus состояние автомобиля после рейса.  
 \*/  
 public void completeFreightTrip(FreightTripApplication trip, Automobile automobile, AutomobileStatus automobileStatus) {  
 trip.setStatus(FreightTripStatus.FINISHED);  
 editedFreightTripApplicationEvent.fire(trip); // Генерация события об обновлении рейса  
 System.out.println("Рейс завершен. Состояние автомобиля: " + automobileStatus);  
 // Обновляем состояние автомобиля  
 automobile.setStatus(automobileStatus);  
 editedAutomobileStatusEvent.fire(automobile); // Генерация события об изменении состояния автомобиля  
 }  
}

package Lab3\_Event;  
  
import Lab3\_Event.Model.\*;  
import Lab3\_Event.Service.DispatcherService;  
import Lab3\_Event.Service.DriverService;  
import org.jboss.weld.environment.se.Weld;  
import org.jboss.weld.environment.se.WeldContainer;  
  
public class Laboratory3 {  
 protected static Weld weld;  
 protected static WeldContainer container;  
  
 public static void main(String[] args) throws Exception {  
 weld = new Weld();  
 container = weld.initialize();  
  
 try {  
 // Получение экземпляров служб через контейнер Weld  
 DispatcherService dispatcherService = container.select(DispatcherService.class).get();  
 DriverService driverService = container.select(DriverService.class).get();  
  
 // Создание тестовых данных  
 Driver driver = new Driver(1, "Иван", "Иванов", false);  
 Automobile automobile = new Automobile(1, "Toyota", "Corolla", AutomobileStatus.OPERATIVE);  
  
 // 1. Добавление заявки на рейс  
 FreightTripApplication trip = dispatcherService.assignTrip(driver, automobile);  
 System.out.println("--------------------------------------------------------");  
  
 // 2. Изменение статуса рейса  
 driverService.updateFreightTripStatus(trip, FreightTripStatus.IN\_PROGRESS);  
 System.out.println("--------------------------------------------------------");  
  
 // 3. Завершение рейса  
 driverService.completeFreightTrip(trip, automobile, AutomobileStatus.INOPERATIVE);  
 System.out.println("--------------------------------------------------------");  
  
 // 4. Подача заявки на ремонт  
 driverService.submitRepairApplication(automobile, driver);  
 System.out.println("--------------------------------------------------------");  
  
 // 5. Отстранение водителя  
 dispatcherService.suspendDriver(driver);  
 System.out.println("--------------------------------------------------------");  
  
 } finally {  
 weld.shutdown();  
 }  
 }  
}

}

# **ВЫВОД**

Работа демонстрирует использование событийного подхода для управления процессами в приложении. Реализация на основе CDI упрощает отделение бизнес-логики от обработки событий, повышая модульность и читаемость кода. Пользовательские аннотации обеспечивают типизацию событий, а использование подписчиков позволяет гибко реагировать на изменения в системе. Такой подход удобен для построения масштабируемых систем с четкой организацией обработки событий.