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 №7

Выполнил: Шедания В. М.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc183377068)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc183377069)

[ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ 6](#_Toc183377070)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 7](#_Toc183377071)

[КОД ПРОГРАММЫ 8](#_Toc183377072)

[ВЫВОД 16](#_Toc183377073)

# **ПОСТАНОВКА ЗАДАЧИ**

Создать систему взаимодействующих классов: класс, хранения графа станций с рёбрами перегонами и хранения операций с поездом (код операции – прибытие или отправление или проследование, время операции, код станции операции), с добавлением и удалением операций; класс, определяющий дефект пропуск операции – есть прибытия, нет отправления или есть отправление и нет прибытия; класс, определяющий дефект пропуск станции – отсутствуют операции по какой-либо станции графа. Классы определения, должны срабатывать при каждом добавлении или удалении операции.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В данной лабораторной работе была реализована симуляция работы железной дороги с использованием CDI (Contexts and Dependency Injection) и событийной модели. Целью работы было продемонстрировать возможности обработки операций и проверок данных в сложной системе.

Сначала создавались основные сущности модели: станции, поезда и операции. Для упрощения работы с ними применялись аннотации из библиотеки Lombok, такие как @Data, @AllArgsConstructor, @NoArgsConstructor. Это позволило минимизировать количество кода, необходимого для создания геттеров, сеттеров и конструкторов. Также была введена перечисляемая структура OperationCode, определяющая типы операций: отправление, прибытие и следование.

Далее был разработан класс RailRoad, представляющий железнодорожную сеть. Этот класс включал в себя основные списки и мапы для хранения станций, перегонов, местоположений поездов и операций. Методами класса были реализованы добавление станций, создание перегонов, выполнение операций и их удаление, а также управление местоположениями поездов. Особенностью было использование CDI-ивентов для уведомления других компонентов системы о выполненных операциях.

Для реализации логики валидации данных использовались валидаторы: OperationsValidator и StationsValidator. Первый из них проверял корректность баланса операций (например, каждая отправка должна соответствовать одному прибытию). Второй валидатор анализировал, какие станции участвуют в операциях, и выводил список станций, которые не использовались. Эти валидаторы подписывались на события через CDI с помощью аннотации @Observes и qualifier @EditedOperations.

Сценарий лабораторной работы включал пошаговую инициализацию: добавление станций, создание перегонов, определение поездов и операций. Затем операции выполнялись с использованием метода addOperation, который не только добавлял данные, но и инициировал выполнение самой операции, включая перемещение поезда и проверку логики.

В финальной части программы использовался метод printRailRoad для вывода состояния всей системы. Этот вывод включал информацию о станциях, перегонах, местоположениях поездов и списке их операций. В завершение работы были сделаны проверки валидаторами для подтверждения корректности выполненных операций и структуры данных.

# **ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ**

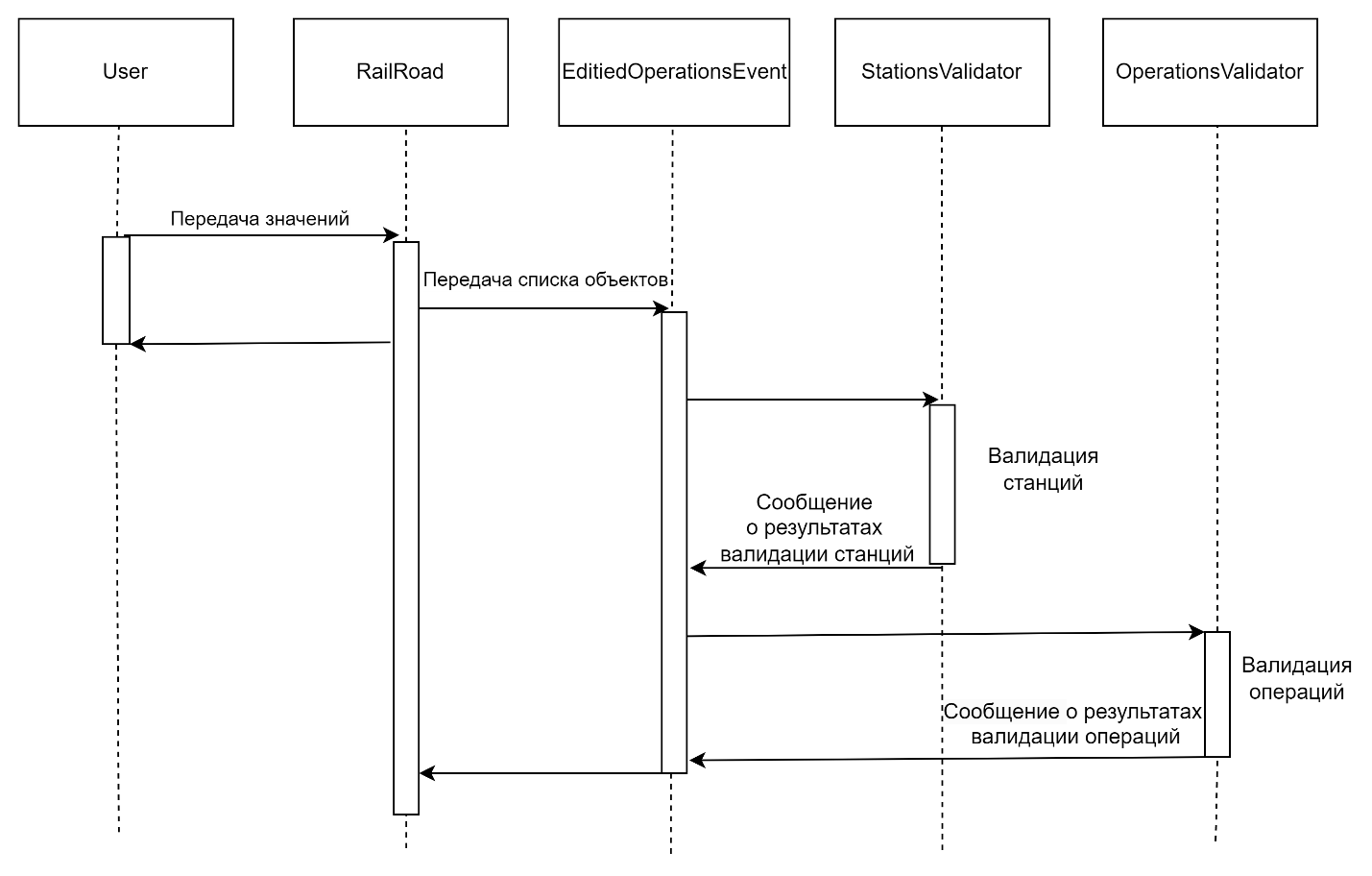


Рисунок 1 – Диаграмма последовательности

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

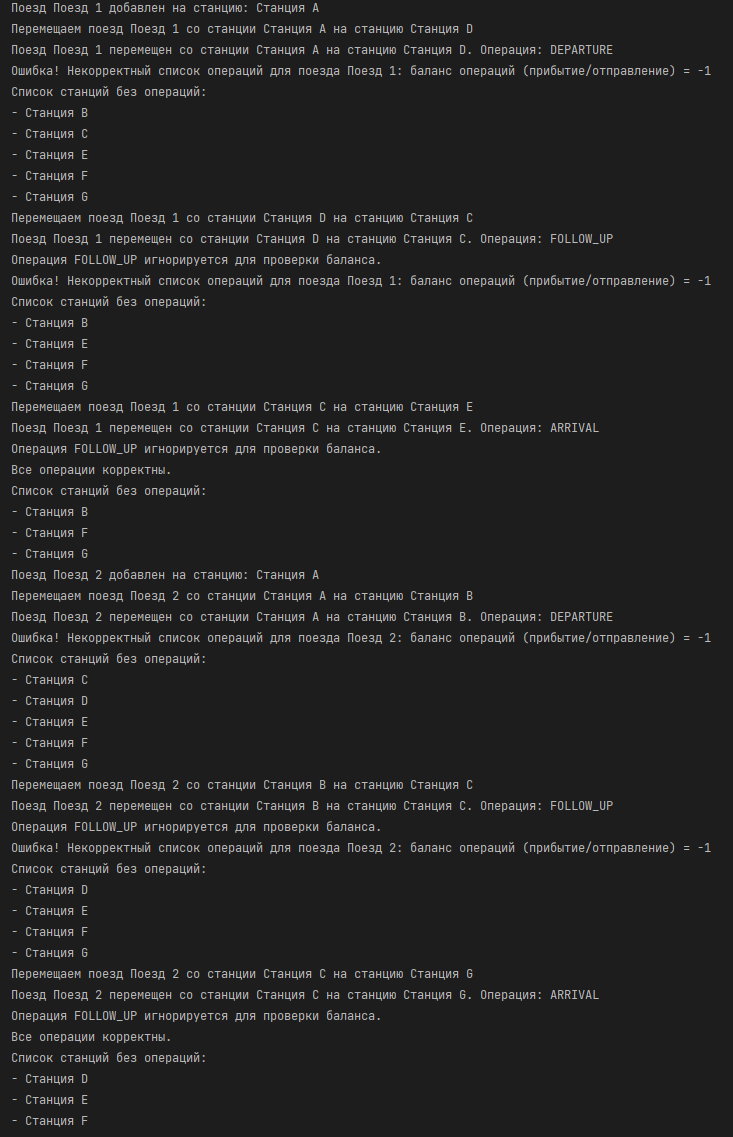


Рисунок 2 – Результат работы программы

# **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 EditedOperations {  
}

package Lab3\_Event.Event;  
  
import Lab3\_Event.Model.Operation;  
import Lab3\_Event.Model.Station;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
  
import java.util.List;  
  
@Data  
@AllArgsConstructor  
public class OperationsAndStationsEvent {  
 private List<Operation> operations;  
 private List<Station> stations;  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Operation {  
 private Train train;  
 private Station from;  
 private Station to;  
 private OperationCode operationCode;  
}

package Lab3\_Event.Model;  
  
public enum OperationCode {  
 DEPARTURE,  
 ARRIVAL,  
 FOLLOW\_UP  
}

package Lab3\_Event.Model;  
  
import Lab3\_Event.Annotation.EditedOperations;  
import Lab3\_Event.Event.OperationsAndStationsEvent;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.enterprise.event.Event;  
import javax.inject.Inject;  
import java.util.\*;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class RailRoad {  
 // Список станций  
 private List<Station> stations = new ArrayList<>();  
 // Список перегонов  
 private List<TrackSection> trackSections = new ArrayList<>();  
 // Местоположение поездов  
 private Map<Train, Station> trainLocations = new HashMap<>();  
 // Операции для каждого поезда  
 private Map<Train, List<Operation>> trainOperations = new HashMap<>();  
  
 @Inject  
 @EditedOperations  
 private Event<OperationsAndStationsEvent> operationsEvent;  
  
 // Добавить станцию  
 public void addStation(Station station) {  
 if (!stations.contains(station)) {  
 stations.add(station);  
 }  
 }  
  
 // Добавить перегон  
 public void addTrackSection(Station from, Station to) {  
 TrackSection trackSection = new TrackSection(from, to);  
 trackSections.add(trackSection);  
 }  
  
 // Найти перегон  
 public TrackSection findTrackSection(Station from, Station to) {  
 return trackSections.stream()  
 .filter(track -> (track.getFrom().equals(from) && track.getTo().equals(to)) ||  
 (track.getFrom().equals(to) && track.getTo().equals(from)))  
 .findFirst()  
 .orElseThrow(() -> new IllegalStateException("Перегон между станциями "  
 + from.getName() + " и "  
 + to.getName() + " не найден."));  
 }  
  
 // Добавить операцию  
 public void addOperation(Operation operation) {  
 Train train = operation.getTrain();  
 Station from = operation.getFrom();  
 Station to = operation.getTo();  
  
 // Проверка перегона  
 findTrackSection(from, to);  
  
 // Установить начальное местоположение  
 if (!trainLocations.containsKey(train)) {  
 trainLocations.put(train, from);  
 System.out.println("Поезд " + train.getName() + " добавлен на станцию: " + from.getName());  
 }  
  
 // Добавить операцию  
 trainOperations.computeIfAbsent(train, k -> new ArrayList<>()).add(operation);  
  
 // Выполнить операцию  
 executeLastOperation(train);  
  
 // Отправить событие  
 List<Operation> operations = trainOperations.get(train);  
 OperationsAndStationsEvent event = new OperationsAndStationsEvent(operations, new ArrayList<>(stations));  
 operationsEvent.fire(event);  
 }  
  
 // Удалить операцию  
 public void removeOperation(Operation operation) {  
 Train train = operation.getTrain();  
  
 if (!trainOperations.containsKey(train)) {  
 System.out.println("Для поезда " + train.getName() + " нет операций для удаления.");  
 return;  
 }  
  
 List<Operation> operations = trainOperations.get(train);  
  
 if (!operations.remove(operation)) {  
 System.out.println("Операция не найдена для удаления: " + operation.getOperationCode());  
 return;  
 }  
  
 System.out.println("Операция удалена для поезда " + train.getName() + ": " + operation.getOperationCode());  
  
 // Если список операций для поезда пуст, удаляем запись о поезде  
 if (operations.isEmpty()) {  
 trainOperations.remove(train);  
 trainLocations.remove(train);  
 System.out.println("Все операции для поезда " + train.getName() + " удалены.");  
 }  
  
 // Отправить событие  
 OperationsAndStationsEvent event = new OperationsAndStationsEvent(operations, new ArrayList<>(stations));  
 operationsEvent.fire(event);  
 }  
  
  
 // Выполнить последнюю операцию  
 public void executeLastOperation(Train train) {  
 if (!trainOperations.containsKey(train)) {  
 System.out.println("Для поезда не найдено операций: " + train.getName());  
 return;  
 }  
  
 List<Operation> operations = trainOperations.get(train);  
  
 if (operations.isEmpty()) {  
 System.out.println("Нет операций для поезда " + train.getName());  
 return;  
 }  
  
 Operation lastOperation = operations.get(operations.size() - 1);  
 moveTrain(lastOperation);  
 System.out.println("Поезд " + train.getName() +  
 " перемещен со станции " + lastOperation.getFrom().getName() +  
 " на станцию " + lastOperation.getTo().getName() +  
 ". Операция: " + lastOperation.getOperationCode());  
 }  
  
 // Переместить поезд  
 private void moveTrain(Operation operation) {  
 Train train = operation.getTrain();  
 Station from = operation.getFrom();  
 Station to = operation.getTo();  
  
 if (!trainLocations.get(train).equals(from)) {  
 throw new IllegalStateException("Поезд не находится в ожидаемом месте.");  
 }  
  
 findTrackSection(from, to);  
  
 System.out.println("Перемещаем поезд " + train.getName() +  
 " со станции " + from.getName() +  
 " на станцию " + to.getName());  
  
 trainLocations.put(train, to);  
 }  
  
 // Получить текущее местоположение  
 public Station getTrainLocation(Train train) {  
 return trainLocations.get(train);  
 }  
  
 // Печать состояния  
 public void printRailRoad() {  
 System.out.println();  
 System.out.println("ЖЕЛЕЗНАЯ ДОРОГА");  
 System.out.println("Станции:");  
 for (Station station : stations) {  
 System.out.println("- " + station);  
 }  
 System.out.println("\nПерегоны:");  
 for (TrackSection track : trackSections) {  
 System.out.println("- Из: " + track.getFrom().getName() +  
 " В: " + track.getTo().getName());  
 }  
 System.out.println("\nМестоположения поездов:");  
 for (Map.Entry<Train, Station> entry : trainLocations.entrySet()) {  
 System.out.println("- Поезд: " + entry.getKey().getName() +  
 " Местоположение: " + entry.getValue().getName());  
 }  
 System.out.println("\nОперации поездов:");  
 for (Map.Entry<Train, List<Operation>> entry : trainOperations.entrySet()) {  
 System.out.println("- Поезд: " + entry.getKey().getName());  
 for (Operation operation : entry.getValue()) {  
 System.out.println(" - Из: " + operation.getFrom().getName() +  
 " В: " + operation.getTo().getName() +  
 " Операция: " + operation.getOperationCode());  
 }  
 }  
 }  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Station {  
 private Integer id;  
 private String name;  
 private String code;  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class TrackSection {  
 private Station from;  
 private Station to;  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Train {  
 private Integer id;  
 private String name;  
 private String code;  
}

package Lab3\_Event.Util;  
  
import Lab3\_Event.Annotation.EditedOperations;  
import Lab3\_Event.Event.OperationsAndStationsEvent;  
import Lab3\_Event.Model.Operation;  
  
import javax.enterprise.event.Observes;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class OperationsValidator {  
  
 public void validateOperations(@Observes @EditedOperations OperationsAndStationsEvent event) {  
 List<Operation> operations = event.getOperations();  
  
 // Карта для отслеживания баланса операций (прибытие/отправление) по поездам  
 Map<String, Integer> operationBalance = new HashMap<>();  
  
 // Обработка операций  
 for (Operation operation : operations) {  
 String trainName = operation.getTrain().getName();  
  
 switch (operation.getOperationCode()) {  
 case ARRIVAL:  
 operationBalance.put(trainName, operationBalance.getOrDefault(trainName, 0) + 1);  
 break;  
  
 case DEPARTURE:  
 operationBalance.put(trainName, operationBalance.getOrDefault(trainName, 0) - 1);  
 break;  
  
 case FOLLOW\_UP:  
 System.out.println("Операция FOLLOW\_UP игнорируется для проверки баланса.");  
 break;  
  
 default:  
 System.out.println("Неизвестный код операции: " + operation.getOperationCode());  
 break;  
 }  
 }  
  
 // Проверка баланса операций для каждого поезда  
 for (Map.Entry<String, Integer> entry : operationBalance.entrySet()) {  
 String trainName = entry.getKey();  
 int balance = entry.getValue();  
  
 if (balance != 0) {  
 System.out.println("Ошибка! Некорректный список операций для поезда " + trainName +  
 ": баланс операций (прибытие/отправление) = " + balance);  
 }  
 }  
  
 // Если все корректно  
 if (operationBalance.values().stream().allMatch(balance -> balance == 0)) {  
 System.out.println("Все операции корректны.");  
 }  
 }  
}

package Lab3\_Event.Util;  
  
import Lab3\_Event.Annotation.EditedOperations;  
import Lab3\_Event.Event.OperationsAndStationsEvent;  
import Lab3\_Event.Model.Operation;  
import Lab3\_Event.Model.Station;  
  
import javax.enterprise.event.Observes;  
import java.util.HashSet;  
import java.util.List;  
import java.util.Set;  
  
public class StationsValidator {  
  
 public void validateStations(@Observes @EditedOperations OperationsAndStationsEvent event) {  
 List<Operation> operations = event.getOperations();  
 List<Station> stations = event.getStations();  
  
 // Собираем все станции, которые участвовали в операциях  
 Set<Station> involvedStations = new HashSet<>();  
 for (Operation operation : operations) {  
 involvedStations.add(operation.getFrom());  
 involvedStations.add(operation.getTo());  
 }  
  
 // Сравниваем с полным списком станций  
 System.out.println("Список станций без операций:");  
 for (Station station : stations) {  
 if (!involvedStations.contains(station)) {  
 System.out.println("- " + station.getName());  
 }  
 }  
 }  
}

package Lab3\_Event;  
  
import Lab3\_Event.Model.\*;  
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 {  
 RailRoad railRoad = container.select(RailRoad.class).get();  
  
 // Инициализация станций  
 Station stationA = new Station(1, "Станция A", "A");  
 Station stationB = new Station(2, "Станция B", "B");  
 Station stationC = new Station(3, "Станция C", "C");  
 Station stationD = new Station(4, "Станция D", "D");  
 Station stationE = new Station(5, "Станция E", "E");  
 Station stationF = new Station(6, "Станция F", "F");  
 Station stationG = new Station(7, "Станция G", "G");  
  
 // Добавление станций в железную дорогу  
 railRoad.addStation(stationA);  
 railRoad.addStation(stationB);  
 railRoad.addStation(stationC);  
 railRoad.addStation(stationD);  
 railRoad.addStation(stationE);  
 railRoad.addStation(stationF);  
 railRoad.addStation(stationG);  
  
 // Инициализация перегонов  
 railRoad.addTrackSection(stationA, stationB);  
 railRoad.addTrackSection(stationA, stationC);  
 railRoad.addTrackSection(stationA, stationD);  
 railRoad.addTrackSection(stationB, stationC);  
 railRoad.addTrackSection(stationD, stationC);  
 railRoad.addTrackSection(stationC, stationE);  
 railRoad.addTrackSection(stationC, stationF);  
 railRoad.addTrackSection(stationC, stationG);  
 railRoad.addTrackSection(stationE, stationF);  
  
 // Инициализация поездов  
 Train train1 = new Train(1, "Поезд 1", "T1");  
 Train train2 = new Train(2, "Поезд 2", "T2");  
  
 // Инициализация операций  
 Operation operation1 = new Operation(train1, stationA, stationD, OperationCode.DEPARTURE);  
 Operation operation2 = new Operation(train1, stationD, stationC, OperationCode.FOLLOW\_UP);  
 Operation operation3 = new Operation(train1, stationC, stationE, OperationCode.ARRIVAL);  
  
 Operation operation4 = new Operation(train2, stationA, stationB, OperationCode.DEPARTURE);  
 Operation operation5 = new Operation(train2, stationB, stationC, OperationCode.FOLLOW\_UP);  
 Operation operation6 = new Operation(train2, stationC, stationG, OperationCode.ARRIVAL);  
  
 // Добавление операций в железную дорогу  
 railRoad.addOperation(operation1);  
 railRoad.addOperation(operation2);  
 railRoad.addOperation(operation3);  
 railRoad.addOperation(operation4);  
 railRoad.addOperation(operation5);  
 railRoad.addOperation(operation6);  
  
 // Вывод состояния железной дороги  
 railRoad.printRailRoad();  
  
 } finally {  
 weld.shutdown();  
 }  
 }  
}

# **ВЫВОД**

Лабораторная работа показала, как можно эффективно использовать события и инъекцию зависимостей для построения сложных взаимодействующих систем. Такой подход позволяет создавать гибкие и расширяемые системы с четким разделением ответственности.