МИНИСТЕРСТВО ТРАНСПОРТА РФ

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

Кафедра ЦТУТП

**Отчёт**

По лабораторной работе №5  
по дисциплине «Корпоративные информационные системы»

Тема: «Создание аннотации для валидации»

Вариант №4

Выполнил: Иванов Н. С.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc180325218)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc180325219)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 5](#_Toc180325220)

[КОД ПРОГРАММЫ 6](#_Toc180325221)

[ВЫВОД 8](#_Toc180325222)

# **ПОСТАНОВКА ЗАДАЧИ**

Описать класс Парк. Поезд может быть принят в любой парк, поэтому предусматривается приоритетность приёма поезда в парк, для этого используются поля ПриоритетПриёмаТранзитногоПоезда и ПриоритетПриёмаПоездаВРасформирование. Создать ограничение для валидации класса Парк, которое проверяет, что одновременно значение этих полей не равно нулю. Продемонстрировать работу с ограничением.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В лабораторной работе реализована валидация объектов с использованием пользовательской аннотации @ValidParkPriority. Эта аннотация проверяет бизнес-правило, согласно которому приоритеты приема поездов на железнодорожной станции (транзитного и расформировочного) не могут одновременно быть равны нулю.

Класс Park представляет собой модель данных с двумя приоритетами: transitTrainPriority и disbandTrainPriority. Поля дополнительно аннотированы стандартными ограничениями @NotNull, чтобы обеспечить их заполненность. На уровне класса используется аннотация @ValidParkPriority, которая передает контроль валидации классу ParkPriorityValidator.

Класс ParkPriorityValidator реализует интерфейс ConstraintValidator и содержит логику проверки. Если одно из полей равно null, валидатор возвращает true, так как проверка на null осуществляется другими аннотациями. Если оба поля заполнены, валидатор проверяет, что хотя бы одно из них имеет ненулевое значение. В противном случае валидация не пройдена.

В классе Laboratory5 создаются три объекта: два валидных и один невалидный. Для каждого объекта вызывается метод validate, который проверяет соответствие правилам и возвращает список нарушений. В случае ошибок выводится сообщение, соответствующее нарушению. Например, для объекта с обоими нулевыми приоритетами возвращается сообщение о недопустимости такого состояния.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

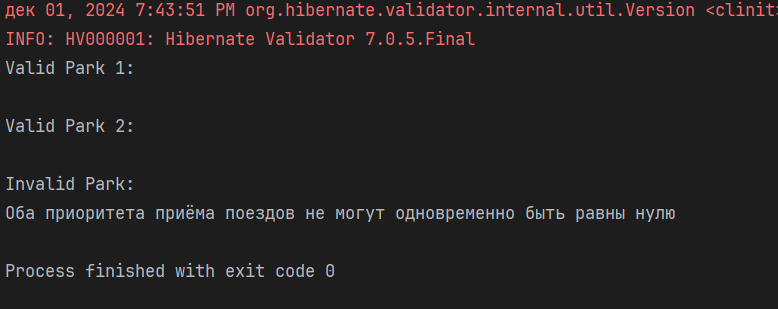


Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package Lab5\_Annotation\_Validation.Annotation;  
  
import Lab5\_Annotation\_Validation.Validator.ParkPriorityValidator;  
import jakarta.validation.Constraint;  
import jakarta.validation.Payload;  
  
import java.lang.annotation.\*;  
  
@Documented  
@Constraint(validatedBy = ParkPriorityValidator.class)  
@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface ValidParkPriority {  
 String message() default "Оба приоритета приёма поездов не могут одновременно быть равны нулю";  
 Class<?>[] groups() default {};  
 Class<? extends Payload>[] payload() default {};  
}

package Lab5\_Annotation\_Validation.Model;  
  
import Lab5\_Annotation\_Validation.Annotation.ValidParkPriority;  
import jakarta.validation.constraints.NotNull;  
import lombok.\*;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@ValidParkPriority  
public class Park {  
  
 @NotNull(message = "Приоритет приёма транзитного поезда не должен быть null")  
 private Integer transitTrainPriority;  
  
 @NotNull(message = "Приоритет приёма поезда в расформирование не должен быть null")  
 private Integer disbandTrainPriority;  
}

package Lab5\_Annotation\_Validation.Validator;  
  
import Lab5\_Annotation\_Validation.Annotation.ValidParkPriority;  
import Lab5\_Annotation\_Validation.Model.Park;  
import jakarta.validation.ConstraintValidator;  
import jakarta.validation.ConstraintValidatorContext;  
  
public class ParkPriorityValidator implements ConstraintValidator<ValidParkPriority, Park> {  
  
 @Override  
 public boolean isValid(Park park, ConstraintValidatorContext context) {  
 if (park.getTransitTrainPriority() == null || park.getDisbandTrainPriority() == null) {  
 return true; // Null-поля обрабатываются другими аннотациями  
 }  
 // Проверяем, что хотя бы одно из полей не равно нулю  
 return park.getTransitTrainPriority() != 0 || park.getDisbandTrainPriority() != 0;  
 }  
}

package Lab5\_Annotation\_Validation;  
  
import Lab5\_Annotation\_Validation.Model.Park;  
import jakarta.validation.Validation;  
import jakarta.validation.Validator;  
import jakarta.validation.ValidatorFactory;  
import org.hibernate.validator.messageinterpolation.ParameterMessageInterpolator;  
  
public class Laboratory5 {  
  
 public static void main(String[] args) {  
 // Создаем экземпляры парков  
 Park validPark1 = new Park(1, 0); // Валидный  
 Park validPark2 = new Park(0, 1); // Валидный  
 Park invalidPark = new Park(0, 0); // Невалидный  
  
 ValidatorFactory factory = Validation.byDefaultProvider()  
 .configure()  
 .messageInterpolator(new ParameterMessageInterpolator())  
 .buildValidatorFactory();  
 Validator validator = factory.getValidator();  
  
 // Проверяем валидный парк с транзитным приоритетом  
 System.out.println("Valid Park 1:");  
 validator.validate(validPark1).forEach(violation -> System.out.println(violation.getMessage()));  
  
 // Проверяем валидный парк с приоритетом на расформирование  
 System.out.println("\nValid Park 2:");  
 validator.validate(validPark2).forEach(violation -> System.out.println(violation.getMessage()));  
  
 // Проверяем невалидный парк  
 System.out.println("\nInvalid Park:");  
 validator.validate(invalidPark).forEach(violation -> System.out.println(violation.getMessage()));  
 }  
}

# **ВЫВОД**

Работа демонстрирует возможности расширения Bean Validation через создание пользовательских аннотаций. Пользовательская аннотация позволяет описывать сложные правила валидации на уровне модели, обеспечивая независимость и повторное использование логики проверки. Такой подход упрощает добавление новых правил без изменения структуры модели данных.