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 №19

Выполнил: Любченкова О.А.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc180330916)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc180330917)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 5](#_Toc180330918)

[КОД ПРОГРАММЫ 6](#_Toc180330919)

[ВЫВОД 10](#_Toc180330920)

# **ПОСТАНОВКА ЗАДАЧИ**

Описать класс ЖелезнаяДорога. С помощью стандартного API валидации с использованием стандартных аннотаций произвести валидацию класса ЖелезнаяДорога. Валидацию провести с использованием аннотаций при описании класса и с указанием ограничений в XML. Обязательно использовать аннотацию @Pattern с проверкой, определяемой регулярным выражением.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

Был реализован класс CodeValidatedRailway, который содержит атрибуты, описывающие железную дорогу: название, страну, количество станций, протяженность дороги и тип пути. Для каждого атрибута использовались аннотации из пакета jakarta.validation.constraints для проверки значений. Например, аннотация @NotNull обеспечивала, что значения не будут null, а @Pattern использовалась для проверки строк с помощью регулярных выражений. Для числовых полей, таких как количество станций и протяженность дороги, применялись аннотации @Min и @Max для ограничения диапазонов значений.

Также был создан второй класс XmlValidatedRailway, который не содержит аннотаций. Валидация для этого класса описана во внешнем XML-файле constraints.xml. В этом файле указаны ограничения для каждого поля: обязательность заполнения, проверки формата строк и диапазонов чисел. Такой подход позволяет разделить логику данных и правила валидации, что делает систему более гибкой.

Для проверки обоих классов был создан основной класс с методами валидации. В нем используется фабрика валидаторов ValidatorFactory из пакета jakarta.validation. Объекты обоих классов создаются с корректными и некорректными данными, а затем проходят проверку на соответствие ограничениям. Результаты валидации выводятся в консоль, где отображаются сообщения об ошибках или подтверждение успешной проверки.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

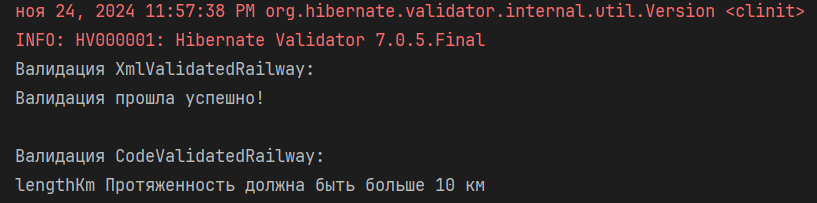


Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package Lab4\_Validation.Model;  
  
import jakarta.validation.constraints.NotNull;  
import jakarta.validation.constraints.Pattern;  
import jakarta.validation.constraints.Min;  
import jakarta.validation.constraints.Max;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
public class CodeValidatedRailway {  
  
 @NotNull(message = "Название железной дороги не должно быть пустым")  
 @Pattern(regexp = "^[А-Яа-яA-Za-z\\s]+$", message = "Название может содержать только буквы и пробелы")  
 private String name;  
  
 @NotNull(message = "Страна не должна быть пустой")  
 @Pattern(regexp = "^[А-Яа-яA-Za-z\\s]+$", message = "Страна может содержать только буквы и пробелы")  
 private String country;  
  
 @Min(value = 1, message = "Количество станций должно быть больше 0")  
 @Max(value = 1000, message = "Количество станций не может превышать 1000")  
 private int stationCount;  
  
 @Min(value = 10, message = "Протяженность должна быть больше 10 км")  
 @Max(value = 10000, message = "Протяженность не может превышать 10000 км")  
 private int lengthKm;  
  
 @NotNull(message = "Тип пути не должен быть пустым")  
 @Pattern(regexp = "^(электрифицированный|неэлектрифицированный)$", message = "Тип пути должен быть 'электрифицированный' или 'неэлектрифицированный'")  
 private String trackType;  
}

package Lab4\_Validation.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
public class XmlValidatedRailway {  
 private String name;  
 private String country;  
 private int stationCount;  
 private int lengthKm;  
 private String trackType;  
}

package Lab4\_Validation;  
  
import Lab4\_Validation.Model.CodeValidatedRailway;  
import Lab4\_Validation.Model.XmlValidatedRailway;  
import jakarta.validation.ConstraintViolation;  
import jakarta.validation.Validation;  
import jakarta.validation.Validator;  
import jakarta.validation.ValidatorFactory;  
import org.hibernate.validator.HibernateValidator;  
import org.hibernate.validator.messageinterpolation.ParameterMessageInterpolator;  
  
import java.util.Set;  
  
public class Laboratory4 {  
  
 public static void main(String[] args) {  
 // Создание фабрики валидаторов  
 ValidatorFactory factory = Validation.byProvider(HibernateValidator.class)  
 .configure()  
 .messageInterpolator(new ParameterMessageInterpolator())  
 .buildValidatorFactory();  
 Validator validator = factory.getValidator();  
  
 // Создание и проверка объекта XmlValidatedRailway  
 XmlValidatedRailway xmlRailway = new XmlValidatedRailway();  
 xmlRailway.setName("Транссибирская магистраль");  
 xmlRailway.setCountry("Россия");  
 xmlRailway.setStationCount(50);  
 xmlRailway.setLengthKm(9200);  
 xmlRailway.setTrackType("электрифицированный");  
  
 System.out.println("Валидация XmlValidatedRailway:");  
 validateAndPrint(validator, xmlRailway);  
  
 // Создание и проверка объекта CodeValidatedRailway  
 CodeValidatedRailway railway = new CodeValidatedRailway();  
 railway.setName("Железная дорога");  
 railway.setCountry("Россия");  
 railway.setStationCount(200);  
 railway.setLengthKm(0);  
 railway.setTrackType("неэлектрифицированный");  
  
 System.out.println("\nВалидация CodeValidatedRailway:");  
 validateAndPrint(validator, railway);  
 }  
  
 // Метод для валидации и вывода результата валидации в консоль  
 private static <T> void validateAndPrint(Validator validator, T object) {  
 Set<ConstraintViolation<T>> violations = validator.validate(object);  
 if (violations.isEmpty()) {  
 System.out.println("Валидация прошла успешно!");  
 } else {  
 for (ConstraintViolation<T> violation : violations) {  
 System.out.println(violation.getPropertyPath() + " " + violation.getMessage());  
 }  
 }  
 }  
}

<validation-config xmlns="http://jboss.org/xml/ns/javax/validation/configuration"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/configuration-validation-configuration-1.1.xsd"  
 version="1.1">  
 <constraint-mapping>META-INF/constraints.xml</constraint-mapping>  
</validation-config>

<constraint-mappings xmlns="http://jboss.org/xml/ns/javax/validation/mapping"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/mapping validation-mapping-1.1.xsd"  
 version="1.1">  
 <bean class="Lab4\_Validation.Model.XmlValidatedRailway" ignore-annotations="false">  
 <field name="name">  
 <constraint annotation="jakarta.validation.constraints.NotNull">  
 <message>Название железной дороги не должно быть пустым</message>  
 </constraint>  
 <constraint annotation="jakarta.validation.constraints.Pattern">  
 <message>Название может содержать только буквы и пробелы</message>  
 <element name="regexp">^[А-Яа-яA-Za-z\s]+$</element>  
 </constraint>  
 </field>  
 <field name="country">  
 <constraint annotation="jakarta.validation.constraints.NotNull">  
 <message>Страна не должна быть пустой</message>  
 </constraint>  
 <constraint annotation="jakarta.validation.constraints.Pattern">  
 <message>Страна может содержать только буквы и пробелы</message>  
 <element name="regexp">^[А-Яа-яA-Za-z\s]+$</element>  
 </constraint>  
 </field>  
 <field name="stationCount">  
 <constraint annotation="jakarta.validation.constraints.Min">  
 <message>Количество станций должно быть больше 0</message>  
 <element name="value">1</element>  
 </constraint>  
 <constraint annotation="jakarta.validation.constraints.Max">  
 <message>Количество станций не может превышать 1000</message>  
 <element name="value">1000</element>  
 </constraint>  
 </field>  
 <field name="lengthKm">  
 <constraint annotation="jakarta.validation.constraints.Min">  
 <message>Протяженность должна быть больше 10 км</message>  
 <element name="value">10</element>  
 </constraint>  
 <constraint annotation="jakarta.validation.constraints.Max">  
 <message>Протяженность не может превышать 10000 км</message>  
 <element name="value">10000</element>  
 </constraint>  
 </field>  
 <field name="trackType">  
 <constraint annotation="jakarta.validation.constraints.NotNull">  
 <message>Тип пути не должен быть пустым</message>  
 </constraint>  
 <constraint annotation="jakarta.validation.constraints.Pattern">  
 <message>Тип пути должен быть 'электрифицированный' или 'неэлектрифицированный'</message>  
 <element name="regexp">^(электрифицированный|неэлектрифицированный)$</element>  
 </constraint>  
 </field>  
 </bean>  
</constraint-mappings>

# **ВЫВОД**

В ходе выполнения лабораторной работы были изучены и применены различные подходы к валидации данных в Java. Реализованы два класса: один с использованием аннотаций валидации, а другой с настройкой ограничений во внешнем XML-файле. Это позволило продемонстрировать, как стандартный API валидации Jakarta Validation может быть использован для обеспечения корректности данных.

Класс с аннотациями показал удобство и компактность кода, так как правила валидации находятся непосредственно в структуре класса. Однако подход с XML-файлом обеспечил большую гибкость, позволяя отделить модель данных от логики валидации, что упрощает изменения в ограничениях, не затрагивая код приложения.

Проведенные тесты показали, что оба подхода корректно проверяют данные на соответствие заданным ограничениям. При вводе правильных данных валидация успешно проходила, а при вводе некорректных значений отображались детализированные сообщения об ошибках, что свидетельствует о правильной настройке ограничений.

Таким образом, лабораторная работа позволила не только освоить базовые возможности Jakarta Validation, но и понять, как выбор подхода к валидации может влиять на архитектуру и поддержку приложения.