МИНИСТЕРСТВО ТРАНСПОРТА РФ

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

Кафедра ЦТУТП

**Отчёт**

По лабораторной работе №4  
по дисциплине «Корпоративные информационные системы»

Тема: «Валидация с использованием стандартных аннотаций»

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 4](#_Toc180325220)

[КОД ПРОГРАММЫ 7](#_Toc180325221)

[ВЫВОД 10](#_Toc180325222)

# **ПОСТАНОВКА ЗАДАЧИ**

Описать класс Парк. С помощью стандартного API валидации с использованием стандартных аннотаций произвести валидацию класса Парк. 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**

В данной лабораторной работе продемонстрирована валидация Java-объектов с использованием API Bean Validation (Jakarta Validation) и реализации Hibernate Validator. Работа включает два подхода к определению ограничений: программный (аннотации в коде) и декларативный (описание ограничений в XML-файле).

Класс CodeValidatedPark представляет объект, валидация которого осуществляется с использованием аннотаций. Каждое поле помечено ограничениями:

* Поле name проверяется на null и соответствие регулярному выражению (только буквы и пробелы).
* Поле address не может быть пустым.
* Поле numberOfTrees имеет минимальное значение 1 и максимальное 10,000.
* Поле parkType должно быть одним из перечисленных значений: "Национальный", "Городской", "Природный".

Класс XmlValidatedPark также представляет объект для валидации, но ограничения для его полей задаются декларативно в файле constraints.xml. Для каждого поля прописаны те же ограничения, что и для CodeValidatedPark.

Основной класс программы, Laboratory4, выполняет валидацию двух объектов:

1. **Объект XmlValidatedPark** создается и заполняется корректными значениями. При вызове метода validateAndPrint ошибок не возникает, так как объект соответствует всем ограничениям.
2. **Объект CodeValidatedPark** заполняется некорректными значениями, нарушающими несколько ограничений. Метод validateAndPrint выводит список ошибок, каждая из которых описывает несоответствие значению конкретного поля заданным условиям.

Метод validateAndPrint использует объект Validator для проверки объекта. Если нарушений ограничений нет, выводится сообщение о успешной валидации. В противном случае отображается перечень всех нарушений с указанием поля и сообщения об ошибке.

Для конфигурации Hibernate Validator используется файл validation-config, который связывает декларативные ограничения из файла constraints.xml с процессом валидации.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

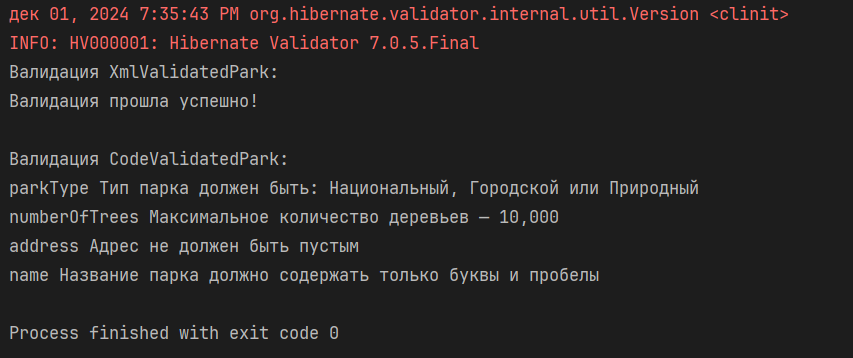


Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package Lab4\_Validation.Model;  
  
import jakarta.validation.constraints.\*;  
import lombok.\*;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class CodeValidatedPark {  
  
 @NotNull(message = "Название парка не должно быть пустым")  
 @Pattern(regexp = "^[A-Za-zА-Яа-яЁё\\s]+$", message = "Название парка должно содержать только буквы и пробелы")  
 private String name;  
  
 @NotNull(message = "Адрес не должен быть пустым")  
 private String address;  
  
 @Min(value = 1, message = "Минимальное количество деревьев должно быть 1")  
 @Max(value = 10000, message = "Максимальное количество деревьев — 10,000")  
 private int numberOfTrees;  
  
 @NotNull(message = "Тип парка не должен быть пустым")  
 @Pattern(regexp = "^(Национальный|Городской|Природный)$", message = "Тип парка должен быть: Национальный, Городской или Природный")  
 private String parkType;  
}

package Lab4\_Validation.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class XmlValidatedPark {  
 private String name;  
 private String address;  
 private int numberOfTrees;  
 private String parkType;  
}

package Lab4\_Validation;  
  
import Lab4\_Validation.Model.XmlValidatedPark;  
import Lab4\_Validation.Model.CodeValidatedPark;  
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) {  
 ValidatorFactory factory = Validation.byProvider(HibernateValidator.class)  
 .configure()  
 .messageInterpolator(new ParameterMessageInterpolator())  
 .buildValidatorFactory();  
 Validator validator = factory.getValidator();  
  
 XmlValidatedPark xmlPark = new XmlValidatedPark();  
 xmlPark.setName("Центральный");  
 xmlPark.setAddress("г. Москва, ул. Тверская, 1");  
 xmlPark.setNumberOfTrees(500);  
 xmlPark.setParkType("Городской");  
  
 System.out.println("Валидация XmlValidatedPark:");  
 validateAndPrint(validator, xmlPark);  
  
 CodeValidatedPark codePark = new CodeValidatedPark();  
 codePark.setName("1234"); // Некорректное имя  
 codePark.setAddress(null); // Поле не должно быть пустым  
 codePark.setNumberOfTrees(15000); // Некорректное значение  
 codePark.setParkType("Промышленный"); // Некорректный тип  
  
 System.out.println("\nВалидация CodeValidatedPark:");  
 validateAndPrint(validator, codePark);  
 }  
  
 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.XmlValidatedPark" ignore-annotations="false">  
 <field name="name">  
 <constraint annotation="jakarta.validation.constraints.NotNull">  
 <message>Название парка не должно быть пустым</message>  
 </constraint>  
 <constraint annotation="jakarta.validation.constraints.Pattern">  
 <message>Название парка должно содержать только буквы и пробелы</message>  
 <element name="regexp">^[A-Za-zА-Яа-яЁё\\s]+$</element>  
 </constraint>  
 </field>  
 <field name="address">  
 <constraint annotation="jakarta.validation.constraints.NotNull">  
 <message>Адрес не должен быть пустым</message>  
 </constraint>  
 </field>  
 <field name="numberOfTrees">  
 <constraint annotation="jakarta.validation.constraints.Min">  
 <message>Минимальное количество деревьев должно быть 1</message>  
 <element name="value">1</element>  
 </constraint>  
 <constraint annotation="jakarta.validation.constraints.Max">  
 <message>Максимальное количество деревьев — 10,000</message>  
 <element name="value">10000</element>  
 </constraint>  
 </field>  
 <field name="parkType">  
 <constraint annotation="jakarta.validation.constraints.NotNull">  
 <message>Тип парка не должен быть пустым</message>  
 </constraint>  
 <constraint annotation="jakarta.validation.constraints.Pattern">  
 <message>Тип парка должен быть: Национальный, Городской или Природный</message>  
 <element name="regexp">^(Национальный|Городской|Природный)$</element>  
 </constraint>  
 </field>  
 </bean>  
</constraint-mappings>

# **ВЫВОД**

Работа демонстрирует два подхода к валидации данных: использование аннотаций в коде и декларативное описание ограничений. Оба подхода обеспечивают гибкость и читаемость кода. Валидация данных является важным этапом разработки приложений, так как позволяет выявлять ошибки и предотвращать некорректную работу системы при обработке пользовательского ввода. Реализация универсальна и может быть расширена для других объектов.