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 №11

Выполнил: Щелкунов Р. Я.

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 9](#_Toc180325222)

# **ПОСТАНОВКА ЗАДАЧИ**

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

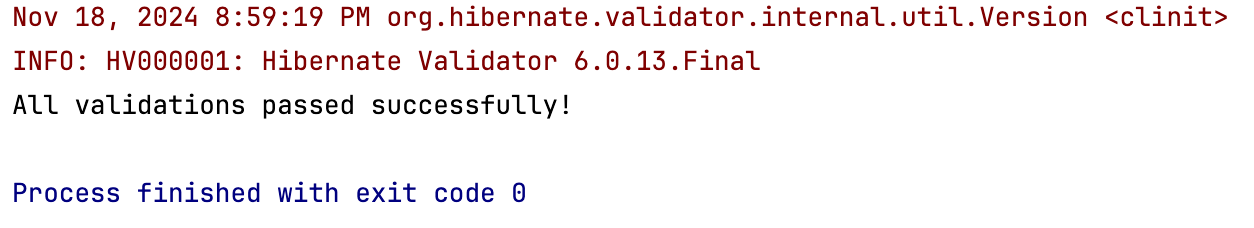
В рамках выполнения лабораторной работы была создана система валидации данных для класса PathSegment с использованием Jakarta Validation и Hibernate Validator. На начальном этапе был разработан класс PathSegment, который включает три поля: pathId, pathName и pathLength. Для каждого из полей были заданы ограничения с использованием стандартных аннотаций Jakarta Validation. Поле pathId ограничено регулярным выражением с помощью аннотации @Pattern для проверки содержания только букв, цифр, подчеркиваний и дефисов. Для поля pathName применена аннотация @Size, которая определяет минимальную длину строки в три символа и максимальную в сто символов. Поле pathLength валидируется на положительное значение с использованием аннотации @Min, задающей минимальное значение равным единице.

Помимо валидации через аннотации, те же ограничения были описаны в конфигурационном XML-файле constraint-mappings. В этом файле для каждого поля класса PathSegment указаны ограничения с соответствующими сообщениями об ошибках. XML-конфигурация была подключена к проекту через стандартный файл настройки валидации validation-config.xml, что позволило задействовать альтернативный способ описания правил валидации, не требующий изменения исходного кода.

В ходе тестирования был создан класс Main, где фабрика валидаторов ValidatorFactory предоставляла объект Validator для проверки экземпляров класса PathSegment. Вначале объект создавался с некорректными значениями, такими как наличие запрещенных символов в pathId, слишком короткое имя в pathName и отрицательное значение длины пути pathLength. Затем метод validator.validate() выполнял проверку, а найденные нарушения выводились в консоль с указанием соответствующих сообщений об ошибках. После этого был создан объект с корректными данными, который успешно проходил проверку на соответствие всем ограничениям.

Разработанная система продемонстрировала возможность использования как аннотационного подхода, так и XML-конфигурации для валидации данных, что обеспечивает гибкость и удобство в управлении ограничениями.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**



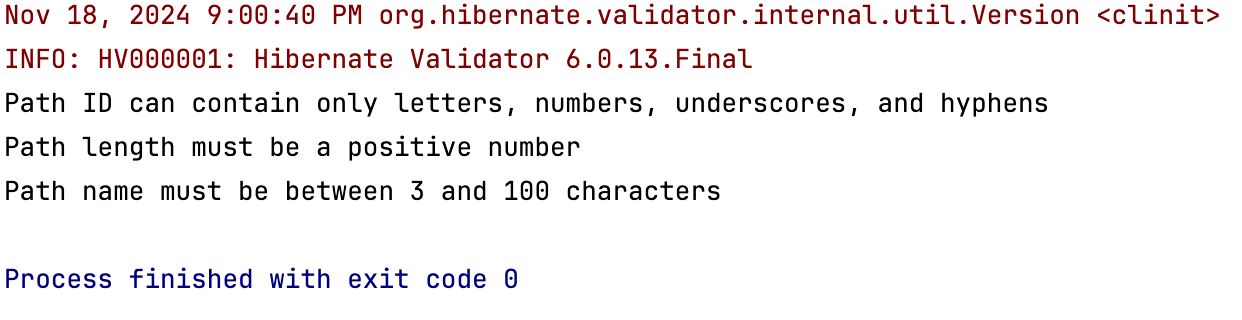


Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package com;  
  
import javax.validation.Validation;  
import javax.validation.Validator;  
import javax.validation.ValidatorFactory;  
import javax.validation.ConstraintViolation;  
import java.util.Set;  
  
public class Main {  
 public static void main(String[] args) {  
 PathSegment pathSegment = new PathSegment("A123!", "A", 0);  
 PathSegment pathSegment2 = new PathSegment("A123", "Abcd", 1);  
  
 *// Валидация с использованием аннотаций  
 validateWithAnnotations*(pathSegment);  
  
 *// Валидация с использованием XML  
 validateWithXML*(pathSegment2);  
 }  
  
 private static void validateWithAnnotations(PathSegment pathSegment) {  
 System.*out*.println("Validating with annotations:");  
 ValidatorFactory factory = Validation.*buildDefaultValidatorFactory*();  
 Validator validator = factory.getValidator();  
  
 Set<ConstraintViolation<PathSegment>> violations = validator.validate(pathSegment);  
  
 *printViolations*(violations);  
 }  
  
 private static void validateWithXML(PathSegment pathSegment) {  
 System.*out*.println("Validating with XML:");  
 ValidatorFactory factory = Validation.*byDefaultProvider*()  
 .configure()  
 .addMapping(Main.class.getResourceAsStream("/META-INF/constraints.xml"))  
 .buildValidatorFactory();  
 Validator validator = factory.getValidator();  
  
 Set<ConstraintViolation<PathSegment>> violations = validator.validate(pathSegment);  
  
 *printViolations*(violations);  
 }  
  
 private static void printViolations(Set<ConstraintViolation<PathSegment>> violations) {  
 if (violations.isEmpty()) {  
 System.*out*.println("All validations passed successfully!");  
 } else {  
 for (ConstraintViolation<PathSegment> violation : violations) {  
 System.*out*.println("Field: " + violation.getPropertyPath());  
 System.*out*.println("Invalid value: " + violation.getInvalidValue());  
 System.*out*.println("Error message: " + violation.getMessage());  
 System.*out*.println();  
 }  
 }  
 }  
}

package com;  
  
  
import javax.validation.constraints.Min;  
import javax.validation.constraints.NotNull;  
import javax.validation.constraints.Pattern;  
import javax.validation.constraints.Size;  
  
public class PathSegment {  
  
 @NotNull(message = "Path ID cannot be null. It is required and should contain only letters, numbers, underscores, or hyphens.")  
 @Pattern(  
 regexp = "^[A-Za-z0-9\_\\-]+$",  
 message = "Path ID must contain only letters, numbers, underscores, and hyphens without any spaces or special characters."  
 )  
 private String pathId;  
  
 @NotNull(message = "Path name cannot be null. Please provide a valid name.")  
 @Size(  
 min = 3,  
 max = 100,  
 message = "Path name must be between 3 and 100 characters. Please ensure the name is within this range."  
 )  
 private String pathName;  
  
 @Min(value = 1, message = "Path length must be a positive number greater than or equal to 1.")  
 private int pathLength;  
  
 *// Constructor* public PathSegment(String pathId, String pathName, int pathLength) {  
 this.pathId = pathId;  
 this.pathName = pathName;  
 this.pathLength = pathLength;  
 }  
  
 *// Getters and Setters* public String getPathId() {  
 return pathId;  
 }  
  
 public void setPathId(String pathId) {  
 this.pathId = pathId;  
 }  
  
 public String getPathName() {  
 return pathName;  
 }  
  
 public void setPathName(String pathName) {  
 this.pathName = pathName;  
 }  
  
 public int getPathLength() {  
 return pathLength;  
 }  
  
 public void setPathLength(int pathLength) {  
 this.pathLength = pathLength;  
 }  
}

*<?*xml version="1.0" encoding="UTF-8"*?>*<validation-config  
 xmlns="http://jboss.org/xml/ns/javax/validation/configuration"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 version="1.1">  
 <constraint-mapping>META-INF/constraints.xml</constraint-mapping>  
</validation-config>

*<?*xml version="1.0" encoding="UTF-8"*?>*<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  
 http://jboss.org/xml/ns/javax/validation/mapping-1.1.xsd"  
 version="1.1">  
  
 <bean class="com.PathSegment">  
 <field name="pathId">  
 <constraint annotation="javax.validation.constraints.Pattern">  
 <message>Path ID can contain only letters, numbers, underscores, and hyphens</message>  
 <element name="regexp">^[A-Za-z0-9\_\-]+$</element>  
 </constraint>  
 </field>  
 <field name="pathName">  
 <constraint annotation="javax.validation.constraints.Size">  
 <message>Path name must be between 3 and 100 characters</message>  
 <element name="min">3</element>  
 <element name="max">100</element>  
 </constraint>  
 </field>  
 <field name="pathLength">  
 <constraint annotation="javax.validation.constraints.Min">  
 <message>Path length must be a positive number</message>  
 <element name="value">1</element>  
 </constraint>  
 </field>  
 </bean>  
</constraint-mappings>

# **ВЫВОД**

В ходе выполнения лабораторной работы были изучены механизмы валидации данных в Java с использованием Jakarta Validation и Hibernate Validator. Реализованный подход продемонстрировал возможность проверки данных как через аннотации, встроенные в код, так и через отдельные XML-конфигурации. Это позволило исследовать преимущества и недостатки каждого подхода.

Аннотационный метод валидации предоставляет удобный способ описания ограничений непосредственно рядом с полями класса, что делает код более читаемым и самодокументируемым. В то же время использование XML-конфигурации обеспечивает гибкость в управлении правилами валидации, особенно в случае изменения требований без необходимости модификации исходного кода.

Проведенные эксперименты подтвердили корректность работы валидатора. При использовании некорректных данных система корректно выявляла нарушения и предоставляла информативные сообщения об ошибках, что облегчает процесс диагностики и исправления данных. Система успешно обработала и корректные данные, подтвердив их соответствие заданным ограничениям.

Работа продемонстрировала эффективность Jakarta Validation как инструмента для обеспечения целостности данных и удобство его использования в приложениях, где требуется строгая проверка ввода.