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 №13

Выполнил: Веселков М.Ю.

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 Магазин. С помощью стандартного API валидации с использованием стандартных аннотаций произвести валидацию класса Магазин. 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 начале лабораторной работы настраивается среда для валидации данных, в которой используется Jakarta Validation с провайдером Hibernate Validator. Для этого создается фабрика валидаторов ValidatorFactory, настраивается интерполятор сообщений, и затем создается валидатор Validator для проверки объектов.

# Далее в проекте реализуются две модели данных — XmlValidatedShop и CodeValidatedShop. В классе CodeValidatedShop используется аннотационная валидация полей с помощью Jakarta Validation, чтобы задать ограничения на значения каждого поля: например, проверка на пустое значение, соответствие регулярным выражениям для названия магазина и адреса, а также минимальное и максимальное значение для количества сотрудников. В XmlValidatedShop валидация не задается аннотациями — вместо этого ограничения прописаны в XML-файле constraints.xml.

# В файле конфигурации валидации validation-config.xml подключается XML-мэппинг ограничений constraints.xml. В этом XML-файле указаны правила валидации для полей класса XmlValidatedShop: ограничения на непустые значения, соответствие формату, а также минимальные и максимальные значения. Это позволяет задать все ограничения без использования аннотаций в коде, только через XML.

# В основном классе Laboratory1 создается экземпляр XmlValidatedShop с корректными данными и вызывается метод validateAndPrint, который проверяет объект на соответствие установленным правилам. Поскольку данные валидны, вывод подтверждает успешное прохождение валидации.

# Затем создается объект CodeValidatedShop с некорректными данными, где нарушены правила валидации, такие как формат названия магазина, адреса, наличие контактного телефона и количество сотрудников. После запуска метода validateAndPrint выводится информация о всех нарушениях валидации с соответствующими сообщениями об ошибках для каждого поля.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package Lab4\_Validation.Model;  
  
import jakarta.validation.constraints.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
public class Shop {  
  
 @NotNull(message = "Название магазина не может быть пустым")  
 private String name;  
  
  
  
 @NotNull(message = "Адрес магазина не должен быть пустым")  
 private String address;  
  
 @NotNull(message = "Номер телефона магазина не должен быть пустым")  
 @Pattern(regexp = "^\\+?\\d{10,15}$", message = "Номер телефона должен содержать от 10 до 15 цифр и может начинаться с +")  
 private String phoneNumber;  
  
 @Min(value = 0, message = "Количество товаров не может быть отрицательным")  
 private int productCount;  
}

package Lab4\_Validation.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
public class XmlValidatedShop {  
  
 private String name;  
 private String address;  
 private String phoneNumber;  
 private int productCount;  
}

package Lab4\_Validation;  
  
import Lab4\_Validation.Model.XmlValidatedShop;  
import Lab4\_Validation.Model.Shop;  
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();  
  
 // Пример для XmlValidatedShop  
 XmlValidatedShop xmlShop = new XmlValidatedShop();  
 xmlShop.setName("SDAS DDFGSD");  
 xmlShop.setAddress("Москва, Тверская 12");  
 xmlShop.setPhoneNumber("+792612а34567");  
 xmlShop.setProductCount(100);  
  
 System.*out*.println("Валидация XmlValidatedShop:");  
 *validateAndPrint*(validator, xmlShop);  
  
 // Пример для Shop с некорректными данными  
 Shop shop = new Shop();  
 shop.setName("Магазин Света"); // Некорректное название  
 shop.setAddress("Москва, Тверская 12"); // Адрес не должен быть пустым  
 shop.setPhoneNumber("+79261234567"); // Некорректный номер телефона  
 shop.setProductCount(10); // Некорректное количество товаров  
  
 System.*out*.println("\nВалидация Shop:");  
 *validateAndPrint*(validator, shop);  
 }  
  
 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());  
 }  
 }  
 }  
}

# **ВЫВОД**

В ходе выполнения лабораторной работы были изучены принципы валидации данных в Java с использованием Jakarta Validation и Hibernate Validator. Были реализованы два подхода к валидации: аннотационная валидация в классе CodeValidatedShop и XML-настройки валидации для класса XmlValidatedShop. Это позволило продемонстрировать гибкость Jakarta Validation и возможность отделения логики валидации от кода с помощью XML-файлов.

Результаты экспериментов показали, что валидация корректно обрабатывает различные типы ограничений: проверку на пустые значения, соответствие регулярным выражениям и диапазоны числовых значений. В случае некорректных данных валидатор выводит сообщения об ошибках, что облегчает диагностику и исправление введенных данных.

Работа подтвердила эффективность Jakarta Validation для обеспечения корректности данных на уровне моделей. Использование аннотаций позволяет сократить объем кода и сосредоточить правила валидации непосредственно рядом с полями. В то же время, конфигурация в XML дает возможность гибко управлять валидацией без изменения исходного кода, что особенно полезно при изменении требований.