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 №12

Выполнил: Бугаева А. Л.

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 4](#_Toc180330918)

[КОД ПРОГРАММЫ 7](#_Toc180330919)

[ВЫВОД 9](#_Toc180330920)

# **ПОСТАНОВКА ЗАДАЧИ**

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

Цель данной лабораторной работы – разработка и использование механизмов валидации объектов в Java с применением библиотеки Hibernate Validator. В рамках работы исследуется валидация с использованием аннотаций для различных типов данных, таких как строки и числа, а также проверка объектов с помощью функционала валидации Jakarta Bean Validation.

Для начала была создана модель CodeValidatedComputer, которая включает аннотации для валидации полей. Использованы такие аннотации, как @NotNull для проверки на null значения, @Pattern для проверки соответствия строк определенным регулярным выражениям, а также @Min и @Max для установки ограничений на значения числовых полей. Например, в классе CodeValidatedComputer задается ограничение для объема ОЗУ (от 1 до 128 ГБ) и для жесткого диска (от 10 до 5000 ГБ). Эти аннотации обеспечивают автоматическую проверку полей объекта на соответствие заданным требованиям.

Для проверки валидации объектов был создан второй класс XmlValidatedComputer, который не использует аннотации для валидации, а предназначен для демонстрации того, как выглядит объект без явной валидации. Для него валидация будет выполняться вручную с использованием валидатора, создаваемого в основной программе.

В основной программе (Laboratory4) инициализируется фабрика валидаторов с использованием Hibernate Validator и специфичной настройки для интерполяции сообщений об ошибках. Затем создаются два объекта: XmlValidatedComputer, который не имеет аннотаций для валидации, и CodeValidatedComputer, для которого валидация настроена через аннотации. Для каждого объекта вызывается метод validateAndPrint, который выполняет валидацию с помощью валидатора и выводит результаты проверки в консоль. Метод проверяет объект на соответствие всем установленным ограничениям и выводит сообщение о наличии ошибок или успешной валидации.

Валидация работает следующим образом: для каждого объекта проверяются все поля с аннотациями, и если какое-то из полей не удовлетворяет условиям (например, строка пустая или число выходит за пределы допустимых значений), то выводится сообщение с указанием нарушения и типа ошибки. Если объект проходит валидацию успешно, выводится сообщение "Валидация прошла успешно!".

**Описание шагов:**

1. Создание валидатора: Используя Hibernate Validator, был настроен фабричный метод для создания экземпляра валидатора с поддержкой интерполяции сообщений.
2. Создание объектов: Созданы два объекта для проверки: один с аннотациями валидации (CodeValidatedComputer), а второй без них (XmlValidatedComputer).
3. Валидация объектов: Для каждого объекта была проведена валидация с использованием метода validateAndPrint, который проверяет поля объекта и выводит сообщения о нарушениях.
4. Вывод результатов: Результаты валидации выводятся в консоль, где для каждого объекта либо выводится сообщение об успешной валидации, либо подробности о нарушениях (например, неправильный формат строки или значение, выходящее за пределы допустимых значений).

В результате работы была продемонстрирована способность Hibernate Validator автоматически проверять объекты на соответствие бизнес-логике, что позволяет избежать ошибок в данных и повысить надежность приложения.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

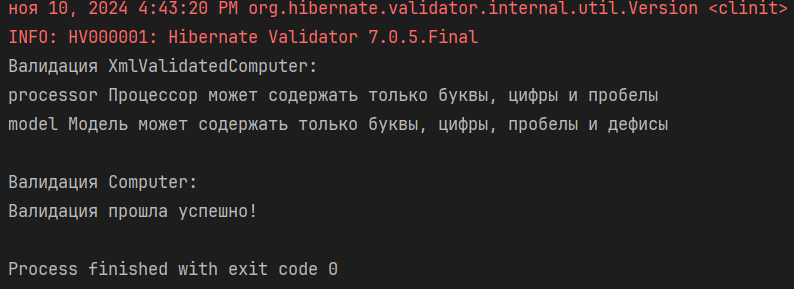


Рисунок 1 – Результат работы программы

# **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 CodeValidatedComputer {  
  
 @NotNull(message = "Модель не должна быть пустой")  
 @Pattern(regexp = "^[A-Za-z0-9\\s-]+$", message = "Модель может содержать только буквы, цифры, пробелы и дефисы")  
 private String model;  
  
 @NotNull(message = "Процессор не должен быть пустым")  
 @Pattern(regexp = "^[A-Za-z0-9\\s]+$", message = "Процессор может содержать только буквы, цифры и пробелы")  
 private String processor;  
  
 @Min(value = 1, message = "Объем ОЗУ должен быть больше 1 ГБ")  
 @Max(value = 128, message = "Объем ОЗУ не может превышать 128 ГБ")  
 private int ram;  
  
 @Min(value = 10, message = "Объем жесткого диска должен быть больше 10 ГБ")  
 @Max(value = 5000, message = "Объем жесткого диска не может превышать 5000 ГБ")  
 private int hdd;  
  
 @NotNull(message = "Операционная система не должна быть пустой")  
 private String operatingSystem;  
}

package Lab4\_Validation.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
public class XmlValidatedComputer {  
  
 private String model;  
 private String processor;  
 private int ram;  
 private int hdd;  
 private String operatingSystem;  
}

package Lab4\_Validation;  
  
import Lab4\_Validation.Model.CodeValidatedComputer;  
import Lab4\_Validation.Model.XmlValidatedComputer;  
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 XmlValidatedComputer  
 XmlValidatedComputer xmlComputer = new XmlValidatedComputer();  
 xmlComputer.setModel("MacBook Pro");  
 xmlComputer.setProcessor("Intel i9");  
 xmlComputer.setRam(32);  
 xmlComputer.setHdd(512);  
 xmlComputer.setOperatingSystem("macOS");  
  
 System.out.println("Валидация XmlValidatedComputer:");  
 validateAndPrint(validator, xmlComputer);  
  
 // Создание и проверка объекта Computer  
 CodeValidatedComputer computer = new CodeValidatedComputer();  
 computer.setModel("Dell XPS 13");  
 computer.setProcessor("Intel i7");  
 computer.setRam(16);  
 computer.setHdd(512);  
 computer.setOperatingSystem("Windows 10");  
  
 System.out.println("\nВалидация Computer:");  
 validateAndPrint(validator, computer);  
 }  
  
 // 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());  
 }  
 }  
 }  
}

# **ВЫВОД**

В ходе выполнения лабораторной работы была продемонстрирована возможность использования Jakarta Bean Validation и Hibernate Validator для валидации объектов в Java. В рамках работы мы рассмотрели использование различных аннотаций, таких как @NotNull, @Pattern, @Min, и @Max, для определения ограничений на поля классов. Эти аннотации обеспечивают автоматическую проверку корректности данных, что снижает вероятность ошибок и упрощает поддержку кода.

Валидация с использованием аннотаций была реализована в классе CodeValidatedComputer, который содержит аннотации для проверки каждого из полей. В классе XmlValidatedComputer, не использующем аннотаций, валидация была проведена вручную с помощью созданного валидатора.

В ходе работы было продемонстрировано, как Hibernate Validator может эффективно проверять объекты на соответствие заданным требованиям, таким как правильность формата строки, соответствие числовых значений заданным диапазонам и отсутствие пустых значений. Кроме того, метод validateAndPrint был использован для вывода результатов валидации, что позволяет легко отслеживать ошибки и корректировать данные.

Таким образом, лабораторная работа показала, как использование механизмов валидации помогает обеспечить корректность данных в приложениях, улучшая качество и надежность программного обеспечения.