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 №15

Выполнил: Дронов А.А.

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 8](#_Toc180325222)

# **ПОСТАНОВКА ЗАДАЧИ**

Описать класс Университет. Поле КоличествоОбучающихсяСтудентов содержит количество студентов в университете, поле ВместимостьАудиторий содержит количество студентов способных одномоментно обучатся в аудиториях университета. Описать ограничение, проверяющие не превышение КоличествоОбучающихсяСтудентов над произведением ВместимостьАудиторий на 3.  Продемонстрировать работу с ограничением.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В данной лабораторной работе реализована проверка объектов класса University с использованием пользовательской аннотации @ValidUniversityCapacity. Эта аннотация гарантирует, что количество студентов не превышает вместимость аудиторий более чем в три раза.

Класс University представляет модель данных с двумя полями: studentCount и auditoriumCapacity. Поля дополнительно аннотированы стандартными ограничениями. Поле studentCount проверяется на отсутствие null и на минимальное значение 0, а auditoriumCapacity должно быть не null и больше 0. Пользовательская аннотация @ValidUniversityCapacity добавляет дополнительное бизнес-правило для проверки соотношения этих двух значений.

Аннотация @ValidUniversityCapacity связана с валидатором UniversityCapacityValidator, который реализует логику проверки. Если одно из полей равно null, валидатор возвращает true, так как проверка отсутствующих значений осуществляется другими аннотациями. Основное правило заключается в том, что количество студентов не может превышать тройное значение вместимости аудиторий. Если это условие нарушается, валидация не пройдена.

Основной класс программы, Laboratory5, создает два объекта университета. Первый объект валиден, так как количество студентов (300) не превышает тройное значение вместимости (100). Второй объект нарушает правило, так как количество студентов (400) превышает допустимое значение (100 \* 3). Метод validate проверяет объекты на соответствие всем правилам и выводит сообщения о нарушениях, если они есть. Для валидного объекта валидация проходит успешно без вывода сообщений.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

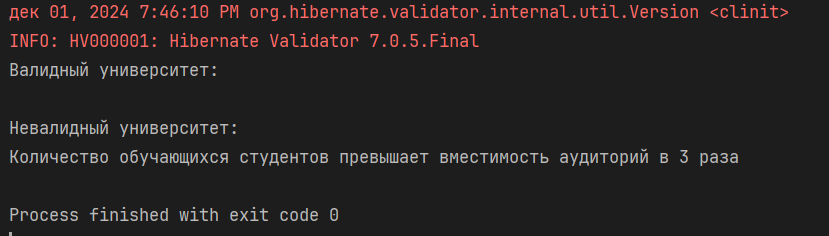


Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package Lab5\_Annotation\_Validation.Annotation;  
  
import Lab5\_Annotation\_Validation.Validator.UniversityCapacityValidator;  
import jakarta.validation.Constraint;  
import jakarta.validation.Payload;  
  
import java.lang.annotation.\*;  
  
@Documented  
@Constraint(validatedBy = UniversityCapacityValidator.class)  
@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface ValidUniversityCapacity {  
 String message() default "Количество обучающихся студентов превышает вместимость аудиторий в 3 раза";  
 Class<?>[] groups() default {};  
 Class<? extends Payload>[] payload() default {};  
}

package Lab5\_Annotation\_Validation.Model;  
  
import Lab5\_Annotation\_Validation.Annotation.ValidUniversityCapacity;  
import jakarta.validation.constraints.Min;  
import jakarta.validation.constraints.NotNull;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@ValidUniversityCapacity  
public class University {  
  
 @NotNull(message = "Количество обучающихся студентов не должно быть null")  
 @Min(value = 0, message = "Количество обучающихся студентов должно быть больше или равно 0")  
 private Integer studentCount;  
  
 @NotNull(message = "Вместимость аудиторий не должна быть null")  
 @Min(value = 1, message = "Вместимость аудиторий должна быть больше 0")  
 private Integer auditoriumCapacity;  
}

package Lab5\_Annotation\_Validation.Validator;  
  
import Lab5\_Annotation\_Validation.Annotation.ValidUniversityCapacity;  
import Lab5\_Annotation\_Validation.Model.University;  
import jakarta.validation.ConstraintValidator;  
import jakarta.validation.ConstraintValidatorContext;  
  
public class UniversityCapacityValidator implements ConstraintValidator<ValidUniversityCapacity, University> {  
  
 @Override  
 public boolean isValid(University university, ConstraintValidatorContext context) {  
 if (university.getStudentCount() == null || university.getAuditoriumCapacity() == null) {  
 return true; // Null-поля обрабатываются другими аннотациями  
 }  
 return university.getStudentCount() <= university.getAuditoriumCapacity() \* 3;  
 }  
}

package Lab5\_Annotation\_Validation;  
  
import Lab5\_Annotation\_Validation.Model.University;  
import jakarta.validation.Validation;  
import jakarta.validation.Validator;  
import jakarta.validation.ValidatorFactory;  
import org.hibernate.validator.messageinterpolation.ParameterMessageInterpolator;  
  
public class Laboratory5 {  
  
 public static void main(String[] args) {  
 // Создаем экземпляры университета  
 University validUniversity = new University(300, 100); // Валидный  
 University invalidUniversity = new University(400, 100); // Невалидный  
  
 ValidatorFactory factory = Validation.byDefaultProvider()  
 .configure()  
 .messageInterpolator(new ParameterMessageInterpolator())  
 .buildValidatorFactory();  
 Validator validator = factory.getValidator();  
  
 // Проверяем валидный университет  
 System.out.println("Valid University:");  
 validator.validate(validUniversity).forEach(violation -> System.out.println(violation.getMessage()));  
  
 // Проверяем невалидный университет  
 System.out.println("\nInvalid University:");  
 validator.validate(invalidUniversity).forEach(violation -> System.out.println(violation.getMessage()));  
 }  
}

# **ВЫВОД**

Работа демонстрирует использование пользовательской аннотации для сложной проверки на уровне модели данных. Такой подход позволяет изолировать бизнес-логику проверки от остального кода, делая её удобной для сопровождения. Использование Bean Validation в сочетании с пользовательскими валидаторами упрощает соблюдение сложных правил валидации в приложениях.