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: «Создание аннотации для валидации»

Вариант №14

Выполнил: Сафонов Г.

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 6](#_Toc180330919)

[ВЫВОД 8](#_Toc180330920)

# **ПОСТАНОВКА ЗАДАЧИ**

Описать класс НаселённыйПункт. Поле КоличествоЧеловек содержит количество человек в населённом пункте, поле Площадь содержит количество квадратных километров, поле ПлотностьНаселения содержит количество человек на квадратный километр. Описать ограничение, проверяющие что ПлотностьНаселения соответствует делению КоличествоЧеловек на Площадь.  Продемонстрировать работу с ограничением.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В начале была создана пользовательская аннотация @ValidPopulationDensity. Эта аннотация помечена мета-аннотацией @Constraint, указывающей на валидатор PopulationDensityValidator, который реализует основную логику проверки. Аннотация применяется на уровне класса, так как проверяемое условие относится к нескольким связанным полям модели.

Далее был реализован класс PopulationDensityValidator, который имплементирует интерфейс ConstraintValidator. Метод isValid проверяет, соответствует ли плотность населения (поле populationDensity) результату деления населения на площадь. Для вычислений используется допуск на погрешность 0.00010.00010.0001, чтобы избежать ошибок, связанных с представлением чисел с плавающей точкой. Также предусмотрено условие, что площади, равные или меньшие нуля, не проверяются валидатором, поскольку это может быть обработано другими аннотациями.

Модель Settlement описывает населённый пункт, включающий три параметра: население, площадь и плотность населения. Класс снабжён аннотацией Lombok для генерации геттеров, сеттеров и конструкторов, а также аннотацией @ValidPopulationDensity для автоматической проверки корректности данных.

В главном классе Laboratory5 созданы два экземпляра класса Settlement. Первый — валидный объект, где плотность населения равна результату деления населения на площадь. Второй — объект с некорректным значением плотности, не соответствующим формуле. Для валидации объектов был использован объект Validator, созданный с помощью фабрики валидаторов.

Для каждого объекта вызван метод validator.validate(...), который выводит сообщения об ошибках на консоль, если данные объекта не удовлетворяют условиям аннотации.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

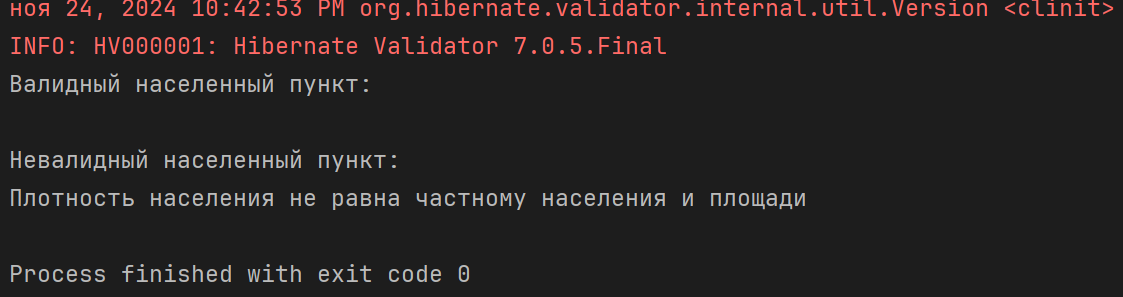


Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package Lab5\_Annotation\_Validation.Annotation;  
  
import Lab5\_Annotation\_Validation.Validator.PopulationDensityValidator;  
import jakarta.validation.Constraint;  
import jakarta.validation.Payload;  
  
import java.lang.annotation.\*;  
  
@Documented  
@Constraint(validatedBy = PopulationDensityValidator.class)  
@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface ValidPopulationDensity {  
 String message() default "Плотность населения не равна частному населения и площади";  
 Class<?>[] groups() default {};  
 Class<? extends Payload>[] payload() default {};  
}

package Lab5\_Annotation\_Validation.Model;  
  
import Lab5\_Annotation\_Validation.Annotation.ValidPopulationDensity;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@ValidPopulationDensity  
public class Settlement {  
 private long population; // Количество человек  
 private double area; // Площадь в квадратных километрах  
 private double populationDensity; // Плотность населения (человек/км²)  
}

package Lab5\_Annotation\_Validation.Validator;  
  
import Lab5\_Annotation\_Validation.Model.Settlement;  
import Lab5\_Annotation\_Validation.Annotation.ValidPopulationDensity;  
import jakarta.validation.ConstraintValidator;  
import jakarta.validation.ConstraintValidatorContext;  
  
public class PopulationDensityValidator implements ConstraintValidator<ValidPopulationDensity, Settlement> {  
  
 @Override  
 public boolean isValid(Settlement settlement, ConstraintValidatorContext context) {  
 if (settlement.getArea() <= 0) {  
 return true; // Площадь <= 0 не проверяется, это можно валидировать другими аннотациями  
 }  
 double calculatedDensity = settlement.getPopulation() / settlement.getArea();  
 return Math.abs(calculatedDensity - settlement.getPopulationDensity()) < 0.0001; // Допуск на погрешность вычислений  
 }  
}

package Lab5\_Annotation\_Validation;  
  
import Lab5\_Annotation\_Validation.Model.Settlement;  
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) {  
 Settlement validSettlement = new Settlement(1000, 2.0, 500.0); // Плотность = 1000 / 2 = 500  
 Settlement invalidSettlement = new Settlement(1000, 2.0, 600.0); // Неверная плотность  
  
 ValidatorFactory factory = Validation.byDefaultProvider()  
 .configure()  
 .messageInterpolator(new ParameterMessageInterpolator())  
 .buildValidatorFactory();  
 Validator validator = factory.getValidator();  
  
 System.out.println("Валидный населенный пункт:");  
 validator.validate(validSettlement).forEach(violation -> System.out.println(violation.getMessage()));  
  
 System.out.println("\nНевалидный населенный пункт:");  
 validator.validate(invalidSettlement).forEach(violation -> System.out.println(violation.getMessage()));  
 }  
}

# **ВЫВОД**

Работа успешно демонстрирует использование аннотаций и кастомных валидаторов в Jakarta Validation. Созданный механизм позволяет эффективно проверять сложные зависимости между полями объекта, централизуя логику проверки и упрощая поддержку кода. Это является важным шагом в разработке приложений, где требуется строгий контроль целостности данных.