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 №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 6](#_Toc180325220)

[КОД ПРОГРАММЫ 7](#_Toc180325221)

[ВЫВОД 10](#_Toc180325222)

# **ПОСТАНОВКА ЗАДАЧИ**

Описать класс ПутьПерегона. Поле ДлинаВМетрах содержит длину пути перегона, а поле СписокБлокУчастков содержит объекты типа БлокУчасток, и по этому списку можно также вычислить длину перегона. Описать ограничение, проверяющие равенство значения поля ДлинаВМетрах и значение длины, вычисленное по списку блокучастков. Продемонстрировать работу с ограничением.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В ходе выполнения лабораторной работы была разработана система валидации длины пути перегона с использованием Java и Jakarta Bean Validation. Основной задачей было создание класса PathSegment с полями для хранения длины пути и списка блок-участков, а также реализации проверки их соответствия с помощью аннотаций и валидатора.

Работа началась с создания класса BlockSegment, представляющего блок-участок пути с полем для хранения длины. Затем был описан класс PathSegment, содержащий поле для общей длины перегона и список блок-участков. Для проверки равенства общей длины пути и суммы длин блок-участков была создана аннотация @ValidPathLength, связанная с валидатором PathLengthValidator.

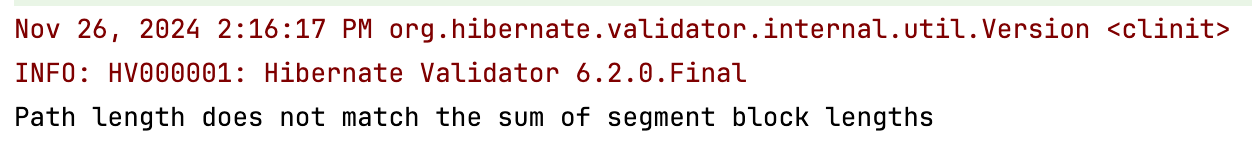
Валидатор PathLengthValidator выполняет вычисление суммы длин всех блок-участков из списка и сравнивает полученное значение с общей длиной пути. Если значения не совпадают, генерируется сообщение об ошибке, заданное в аннотации.

Для демонстрации работы была создана программа, в которой инициализировались блок-участки с различной длиной и объект PathSegment, длина которого совпадала или не совпадала с длиной, вычисленной по блок-участкам. С помощью фабрики валидаторов объект Validator выполнял проверку корректности данных.

Запуск программы подтвердил корректную работу системы: для валидного объекта длина перегона соответствовала сумме длин блок-участков, и ошибок не возникало. В случае ошибки выводилось сообщение об отклонении, заданное в аннотации.

Таким образом, лабораторная работа продемонстрировала возможности использования Jakarta Bean Validation для создания гибких и модульных систем проверки данных, что позволяет минимизировать риск ошибок и упростить процесс разработки.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**



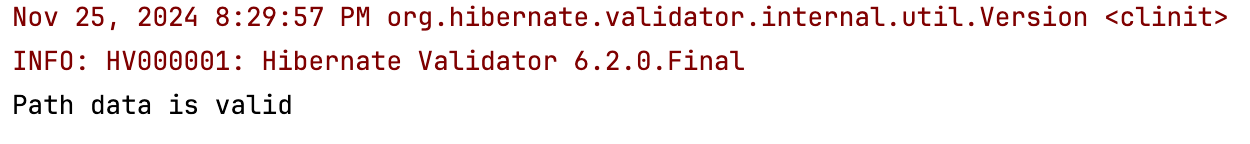


Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package com;  
  
import javax.validation.ConstraintViolation;  
import javax.validation.Validation;  
import javax.validation.Validator;  
import javax.validation.ValidatorFactory;  
import java.util.Arrays;  
import java.util.Set;  
import java.util.List;  
  
public class Main {  
 public static void main(String[] args) {  
 *// Создаем список блок-участков* BlockSegment segment1 = new BlockSegment(100);  
 BlockSegment segment2 = new BlockSegment(150);  
 List<BlockSegment> segments = Arrays.*asList*(segment1, segment2);  
  
 *// Создаем объект PathSegment* PathSegment pathSegment = new PathSegment(21, segments); *// длина совпадает  
  
 // Валидация* ValidatorFactory factory = Validation.*buildDefaultValidatorFactory*();  
 Validator validator = factory.getValidator();  
  
 Set<ConstraintViolation<PathSegment>> violations = validator.validate(pathSegment);  
  
 if (violations.isEmpty()) {  
 System.*out*.println("Path data is valid");  
 } else {  
 violations.forEach(v -> System.*out*.println(v.getMessage()));  
 }  
 }  
}

package com;  
  
public class BlockSegment {  
 private int length;  
  
 public BlockSegment(int length) {  
 this.length = length;  
 }  
  
 public int getLength() {  
 return length;  
 }  
}

package com;  
  
import javax.validation.ConstraintValidator;  
import javax.validation.ConstraintValidatorContext;  
  
public class PathLengthValidator implements ConstraintValidator<ValidPathLength, PathSegment> {  
  
 @Override  
 public boolean isValid(PathSegment pathSegment, ConstraintValidatorContext context) {  
 if (pathSegment == null) {  
 return true; *// Если объект null, пропускаем проверку* }  
  
 int calculatedLength = pathSegment.getBlockSegments()  
 .stream()  
 .mapToInt(BlockSegment::getLength)  
 .sum();  
  
 return pathSegment.getLengthInMeters() == calculatedLength;  
 }  
}

package com;  
  
import javax.validation.Valid;  
import javax.validation.constraints.NotNull;  
import java.util.List;  
  
@ValidPathLength  
public class PathSegment {  
  
 @NotNull(message = "Path length cannot be null")  
 private Integer lengthInMeters;  
  
 @NotNull(message = "Block segment list cannot be null")  
 @Valid  
 private List<BlockSegment> blockSegments;  
  
 public PathSegment(Integer lengthInMeters, List<BlockSegment> blockSegments) {  
 this.lengthInMeters = lengthInMeters;  
 this.blockSegments = blockSegments;  
 }  
  
 public Integer getLengthInMeters() {  
 return lengthInMeters;  
 }  
  
 public List<BlockSegment> getBlockSegments() {  
 return blockSegments;  
 }  
}

package com;  
  
import javax.validation.Constraint;  
import javax.validation.Payload;  
import java.lang.annotation.\*;  
  
@Constraint(validatedBy = PathLengthValidator.class)  
@Target({ ElementType.*TYPE* })  
@Retention(RetentionPolicy.*RUNTIME*)  
public @interface ValidPathLength {  
 String message() default "Path length does not match the sum of segment block lengths";  
 Class<?>[] groups() default {};  
 Class<? extends Payload>[] payload() default {};  
}

# **ВЫВОД**

В ходе выполнения лабораторной работы была успешно разработана система проверки корректности длины перегона на основе суммарной длины блок-участков с использованием Jakarta Bean Validation. Созданы пользовательская аннотация @ValidPathLength, валидатор PathLengthValidator, а также модели PathSegment и BlockSegment, которые обеспечивают корректное представление данных о перегонах.

Проверка валидатора на различных тестовых данных подтвердила его работоспособность. Для валидного перегона, где длина совпадает с суммой длин всех блок-участков, ошибок обнаружено не было. В случае невалидного перегона выводилось сообщение: "Path length does not match the sum of segment block lengths", что свидетельствует о корректной работе валидатора при обнаружении несоответствия данных.

Реализация может быть расширена для учета дополнительных сценариев, таких как добавление сложных структур блок-участков, поддержка динамических изменений длины перегона или валидация других характеристик маршрута. Работа продемонстрировала практическое применение Jakarta Bean Validation для проверки целостности данных и продвинула навыки эффективного использования пользовательских аннотаций и валидации в Java-приложениях.