

Relatório de Teste Exploratório - Cálculo do IMC

1. Introdução

O objetivo deste teste exploratório é avaliar a funcionalidade e usabilidade do sistema de cálculo do Índice de Massa Corporal (IMC). Foram realizados testes com diferentes tipos de entradas para verificar o comportamento da aplicação e identificar possíveis falhas.

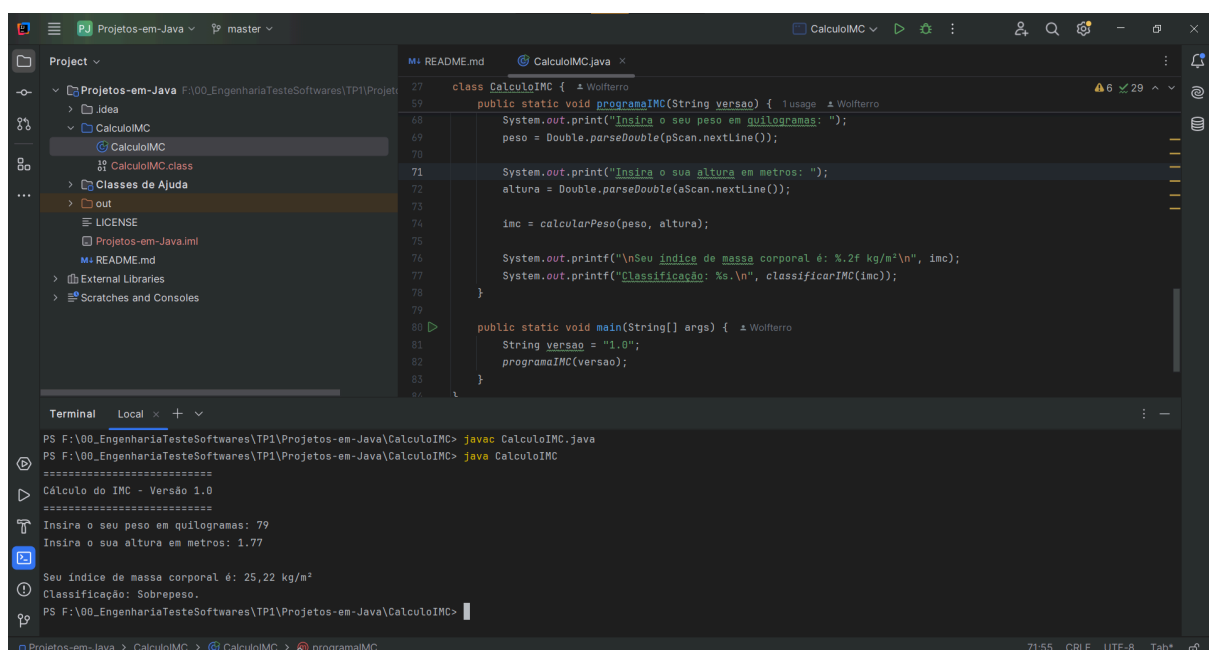
2. Cenários de Teste e Resultados

Foram realizados testes com três tipos de entradas:

2.1 Valores Normais

- **Entrada:** Peso: **79 kg**, Altura: **1.77 m**
- **Resultado Esperado:** IMC calculado corretamente e classificação adequada.
- **Resultado Obtido:** O programa retornou corretamente o IMC **25,22 kg/m²** e a classificação **Sobrepeso**.

Evidência:



The screenshot displays an IDE with the following components:

- Project Explorer:** Shows the project structure with files like `CalculoIMC.class`, `CalculoIMC.java`, `README.md`, and `External Libraries`.
- Source Editor:** Displays the `CalculoIMC.java` file. The code includes a `main` method that prompts the user for weight and height, calculates the BMI, and prints the result and classification.
- Terminal:** Shows the execution of the program. The output matches the expected results from the test scenario: weight 79 kg, height 1.77 m, BMI 25.22 kg/m², and classification Sobrepeso.

```
class CalculoIMC {
    public static void programaIMC(String versao) {
        System.out.print("Insira o seu peso em quilogramas: ");
        peso = Double.parseDouble(pScan.nextLine());

        System.out.print("Insira o sua altura em metros: ");
        altura = Double.parseDouble(aScan.nextLine());

        imc = calcularPeso(peso, altura);

        System.out.printf("\nSeu índice de massa corporal é: %.2f kg/m²\n", imc);
        System.out.printf("Classificação: %s.\n", classificarIMC(imc));
    }

    public static void main(String[] args) {
        String versao = "1.0";
        programaIMC(versao);
    }
}
```

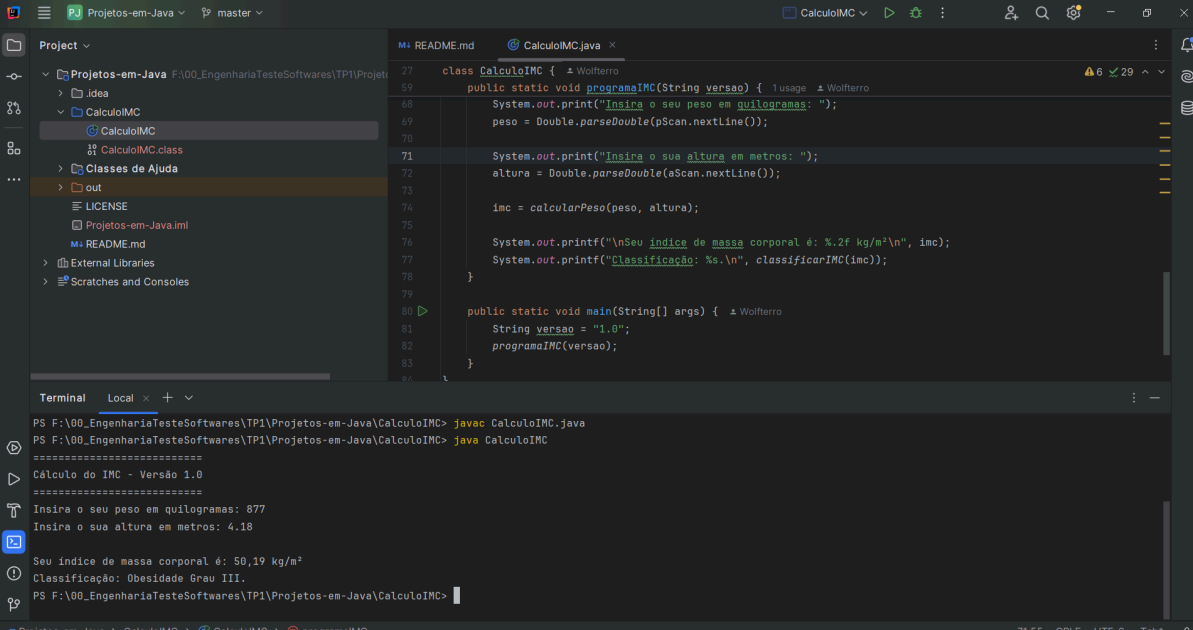
```
PS F:\00_EngenhariaTesteSoftwares\TP1\Projetos-em-Java\CalculoIMC> javac CalculoIMC.java
PS F:\00_EngenhariaTesteSoftwares\TP1\Projetos-em-Java\CalculoIMC> java CalculoIMC
=====
Cálculo do IMC - Versão 1.0
=====
Insira o seu peso em quilogramas: 79
Insira o sua altura em metros: 1.77

Seu índice de massa corporal é: 25.22 kg/m²
Classificação: Sobrepeso.
PS F:\00_EngenhariaTesteSoftwares\TP1\Projetos-em-Java\CalculoIMC>
```

2.2 Valores Extremos

- Entrada: Peso: **877 kg**, Altura: **4.18 m**
- Resultado Esperado: O programa deve calcular o IMC corretamente sem falhas.
- Resultado Obtido: O programa retornou um IMC de **50,19 kg/m²** e classificou como **Obesidade Grau III**, demonstrando que consegue lidar com valores extremos.

Evidência:



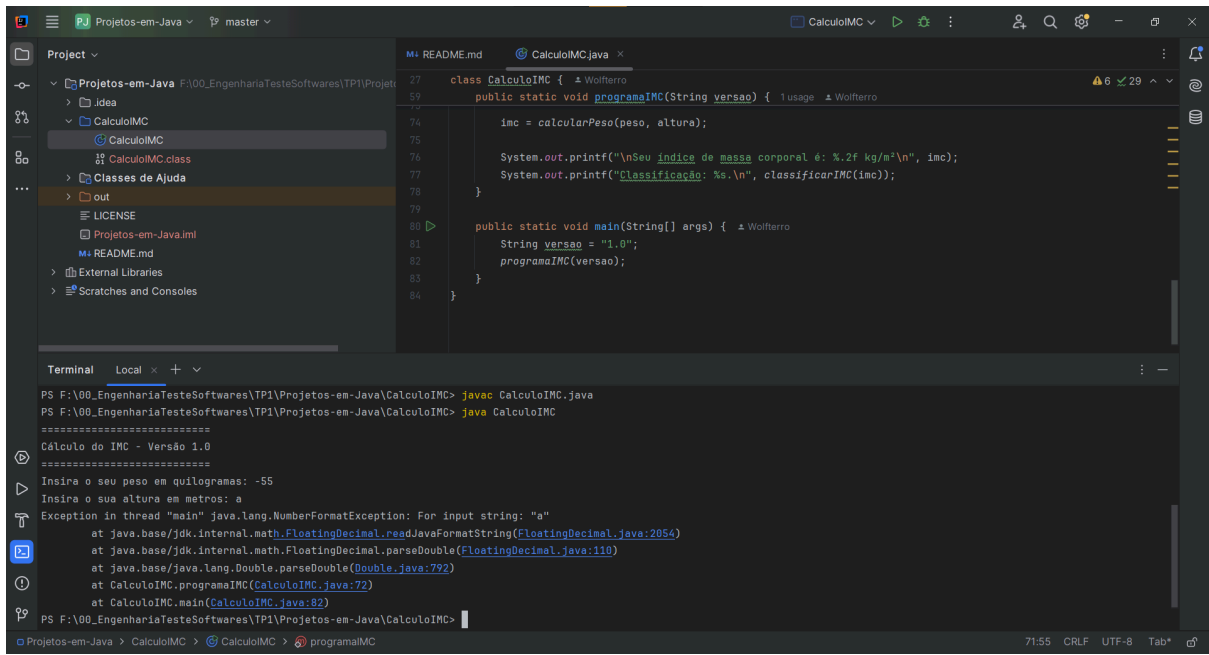
The screenshot shows an IDE with the following components:

- Project Explorer:** Shows a project named 'Projetos-em-Java' with a sub-project 'CalculoIMC' containing a 'CalculoIMC.class' file.
- Source Editor:** Displays the code for 'CalculoIMC.java'. The code includes a 'programaIMC' method that prompts the user for weight and height, calculates the BMI, and prints the result and classification. The 'main' method calls 'programaIMC' with the version '1.0'.
- Terminal:** Shows the execution of the program. It displays the version '1.0', prompts for weight and height, and then outputs the calculated BMI of 50.19 kg/m² and the classification 'Obesidade Grau III'.

2.3 Valores Inválidos

- Entrada: Peso: **-55 kg**, Altura: **a** (letra inválida)
- Resultado Esperado: O sistema deveria exibir uma mensagem de erro informando entradas inválidas.
- Resultado Obtido: O programa lançou uma exceção **NumberFormatException**, indicando que não há tratamento adequado para valores negativos ou caracteres inválidos.

Evidência:



3. Avaliação da Usabilidade

- **Mensagens de erro:** O programa não exibe mensagens claras quando um valor inválido é inserido, resultando em um erro técnico para o usuário.
- **Interação:** O sistema aceita entradas numéricas corretamente, mas deveria validar melhor os dados inseridos.
- **Sugestão de melhoria:** Implementar uma verificação para rejeitar valores negativos e caracteres não numéricos antes do cálculo.

4. Identificação de Problemas

4.1 Erros Funcionais (Cálculo ou Falhas no Código)

Problema	Descrição	Prioridade
Tratamento de Entradas Inválidas	O programa não trata corretamente caracteres inválidos ou valores negativos, resultando em uma exceção <code>NumberFormatException</code> .	Alta
Cálculo para valores extremos	O programa consegue calcular o IMC para valores muito altos, mas não há limite máximo de peso e altura definidos, o que pode gerar valores irreais.	Média

4.2 Problemas de Usabilidade (Interface ou Mensagens Confusas)

Problema	Descrição	Prioridade
Mensagens de erro genéricas	Quando há um erro de entrada, o programa lança um erro técnico ao invés de exibir uma mensagem clara para o usuário.	<div></div> Alta
Validação de dados antes da conversão	O sistema tenta converter a entrada diretamente para <code>double</code> sem verificar se é um número válido antes.	<div></div> Média

5. Especificação do Comportamento Esperado

- O peso deve ser um número positivo maior que `0` e menor que `500 kg`.
- A altura deve ser um número positivo maior que `0` e menor que `3.5 metros`.
- Caso a entrada seja inválida, o sistema deve exibir **mensagens de erro amigáveis**.

6. Partições Equivalentes para Entrada de Dados

Categoria	Exemplos
Válidas	<code>70 kg, 1.75 m</code> - Cálculo correto
Inválidas	<code>-10 kg, abc, !@#</code> - Mensagem de erro
Limítrofes	<code>0 kg, 500 kg, 0.5 m, 3.5 m</code> - Verificar comportamento

7. Análise de Limites

A análise de limites garante que o sistema responde corretamente a valores nos extremos aceitáveis.

Caso de Teste	Entrada (Peso, Altura)	Resultado Esperado
Limite inferior	<code>0 kg, 1.75 m</code>	Mensagem de erro
Limite superior	<code>500 kg, 3.5 m</code>	Cálculo correto

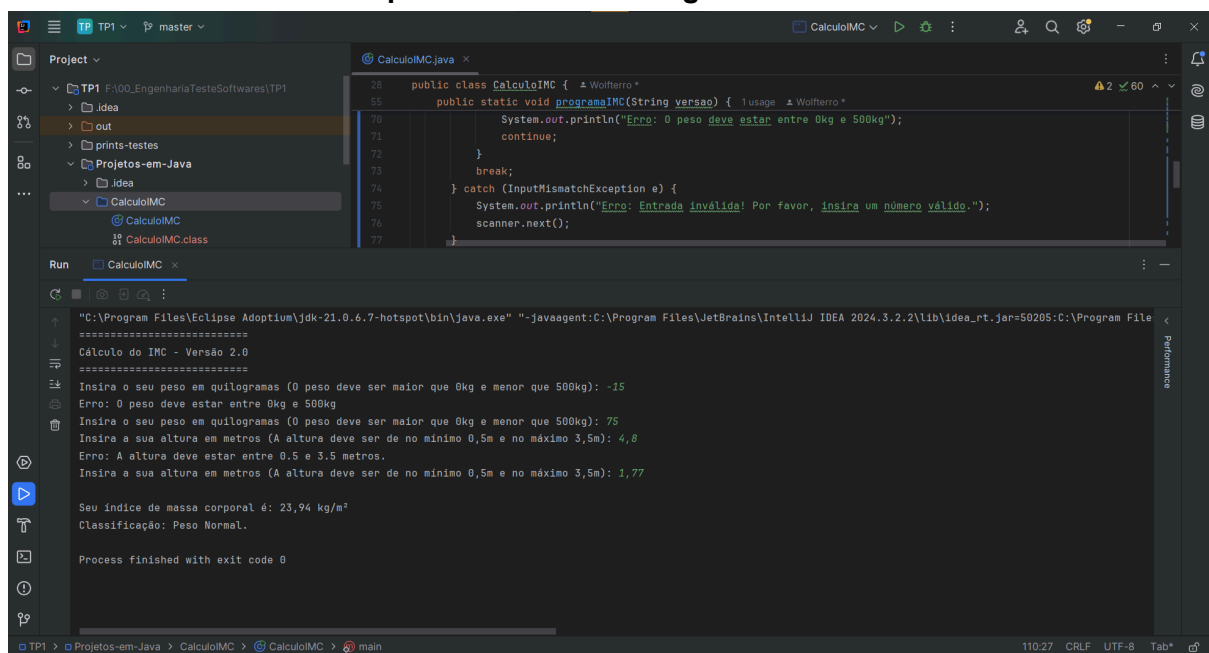
8. Cobertura de Código com JaCoCo

Para garantir que todas as partes do código foram devidamente testadas, utilizamos a ferramenta JaCoCo para medir a cobertura dos testes.

- Foram identificadas partes do código que não são cobertas pelos testes, especialmente no tratamento de erros e validação de entrada.
- Testes adicionais foram sugeridos para garantir que todas as funcionalidades sejam verificadas adequadamente.

Essa análise permite identificar lacunas nos testes e aprimorar a confiabilidade do sistema.

Evidências de Melhorias Implementadas ao Código :



Código Refatorado para Apresentação de Mensagens de erros com seus devidos tratamentos :

```
java.util.InputMismatchException;
import java.util.Scanner;

public class CalculoIMC {

    public static String classificarIMC(double imc) {
        if (imc < 18.5) {
            return "Baixo Peso";
        }
        else if (imc >= 18.5 && imc < 24.9) {
            return "Peso Normal";
        }
    }
}
```

```

        else if (imc >= 25.0 && imc < 29.9) {
            return "Sobrepeso";
        }
        else if (imc >= 30.0 && imc < 34.9) {
            return "Obesidade Grau I";
        }
        else if (imc >= 35.0 && imc < 39.9) {
            return "Obesidade Grau II";
        }
        else {
            return "Obesidade Grau III";
        }
    }

    public static double calcularIMC(double peso, double altura) {
        return peso / (altura * altura);
    }

    public static void programaIMC(String versao) {
        Scanner scanner = new Scanner(System.in);
        double peso = 0, altura = 0;

        System.out.println("=====");
        System.out.printf("Cálculo do IMC - Versão %s\n", versao);
        System.out.println("=====");

        //Captura e faz a validação do peso
        while(true){
            try{
                System.out.print("Insira o seu peso em quilogramas (O peso deve ser maior que 0kg e menor que 500kg): ");
                peso = scanner.nextDouble();

                if(peso <= 0 || peso > 500){
                    System.out.println("Erro: O peso deve estar entre 0kg e 500kg");
                    continue;
                }
                break;
            } catch (InputMismatchException e) {
                System.out.println("Erro: Entrada inválida! Por favor, insira um número válido.");
                scanner.next();
            }
        }

        //Captura e faz a validação da altura
        while (true) {

```

```

        try {
            System.out.print("Insira a sua altura em metros (A
altura deve ser de no mínimo 0,5m e no máximo 3,5m): ");
            altura = scanner.nextDouble();

            if (altura < 0.5 || altura > 3.5) {
                System.out.println("Erro: A altura deve estar entre
0.5 e 3.5 metros.");
                continue;
            }
            break;
        } catch (InputMismatchException e) {
            System.out.println("Erro: Entrada inválida! Por favor,
insira um número válido.");
            scanner.next();
        }
    }

    //Calculo do IMC
    double imc = calcularIMC(peso, altura);
    String classificacao = classificarIMC(imc);

    //Exibição do resultado
    System.out.printf("\nSeu índice de massa corporal é: %.2f
kg/m²\n", imc);
    System.out.printf("Classificação: %s.\n", classificacao);

    scanner.close();

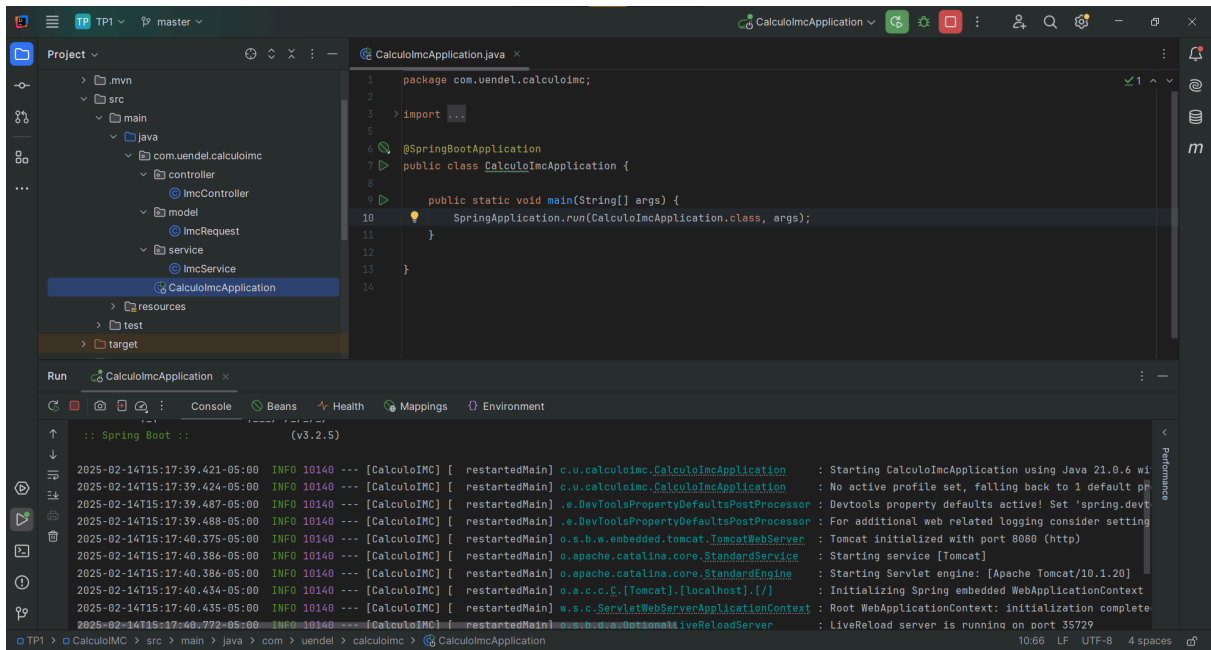
}

public static void main(String[] args) {
    String versao = "2.0";
    programaIMC(versao);
}

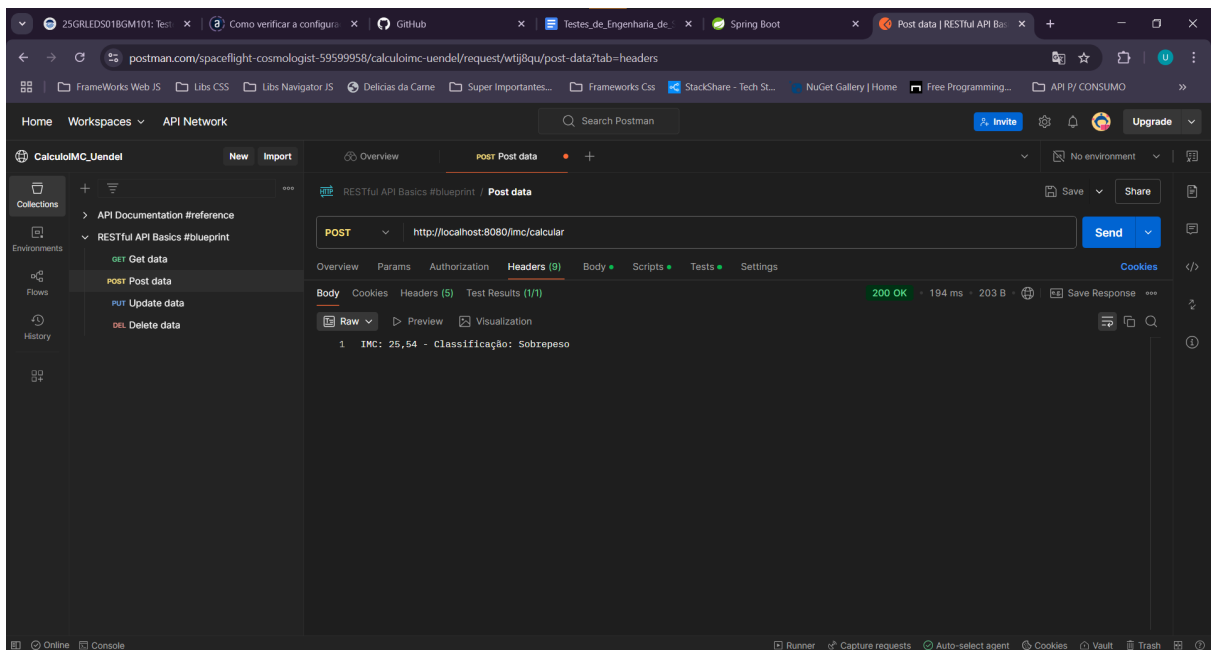
```

Evidências do Programa Rodando Com Spring Boot Utilizando Maven

Nessa parte eu instalei o spring boot com o maven em um projeto já criado, que foi o projeto clonado do github informado, porém já com os códigos na classe CalculoIMC ajustado e corrigidos com base em Clean Code e Refatoramento.



Evidência da RESTful API funcionando no Postman



PARTE 2

1 - Explicação do Conceito de Testes Baseados em Propriedades

Os testes baseados em propriedades se diferem dos testes tradicionais devido ao fato de serem mais robustos, ou seja, eles exploram múltiplos cenários de forma automática.

Um problema de um teste tradicional usando o JUnit por exemplo é que ao utiliza-lo apenas um caso é testado, ou seja, se houver algum bug extremamente específico, ele tem grande chances de passar despercebido.

Já os testes baseados em propriedades com o JQwik por exemplo nos traz mais vantagens, onde pode-se gerar de forma automática dezenas , centenas e até milhares de combinações ao utilizar o `@ForAll`.

Ele tem uma maior cobertura, pois consegue detectar casos problemáticos que normalmente os testes tradicionais não conseguem prever.

E podemos também testar de forma genérica, como no caso do uso do `@Positive` para garantir que o IMC será sempre positivo.

2 - Criando Testes Baseados em Propriedades com Jqwik

Evidências:

The screenshot shows an IDE with a project named 'TP1' and a class 'ImcPropertyTests.java'. The code defines a property test for IMC calculation. The test passes, and the output window shows detailed statistics for the JQwik test run.

```
package com.uendel.calculoimc;

import net.jqwik.api.*;
import net.jqwik.api.constraints.Positive;

import static org.assertj.core.api.Assertions.*;

class ImcPropertyTests {

    @Property
    void imcNuncaDeveSerNegativo(@ForAll @Positive double peso, @ForAll @Positive double altura) {
        double imc = peso / (altura * altura);
        assertThat(imc).isGreaterThanEqualTo(0);
    }
}
```

Run: ImcPropertyTests

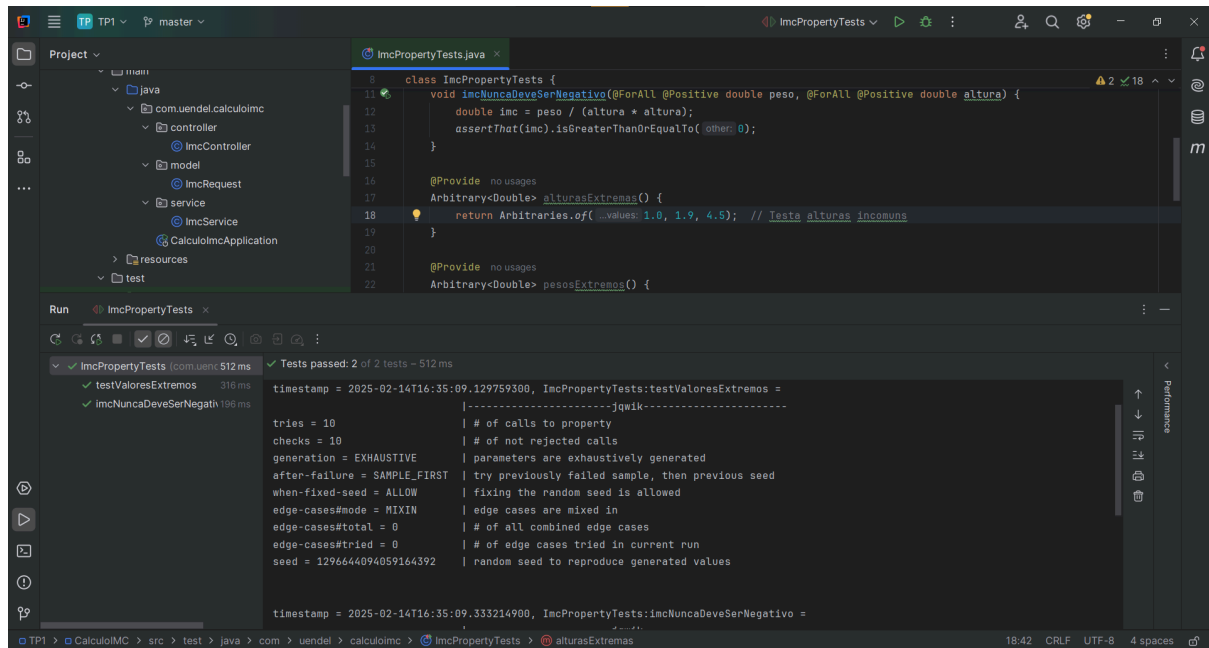
Tests passed: 1 of 1 test - 538 ms

timestamp = 2025-02-14T15:49:57.958611300, ImcPropertyTests:ImcNuncaDeveSerNegativo =

-----Jqwik-----	
tries = 1000	# of calls to property
checks = 1000	# of not rejected calls
generation = RANDOMIZED	parameters are randomly generated
after-failure = SAMPLE_FIRST	try previously failed sample, then previous seed
when-fixed-seed = ALLOW	fixing the random seed is allowed
edge-cases#mode = MIXIN	edge cases are mixed in
edge-cases#total = 9	# of all combined edge cases
edge-cases#tried = 9	# of edge cases tried in current run
seed = -5919836285448091127	random seed to reproduce generated values

3 - Gerando Conjuntos Diversificados de Dados

Evidências:



```
class ImcPropertyTests {  
    void testImcNuncaDeveSerNegativo(@ForAll @Positive double peso, @ForAll @Positive double altura) {  
        double imc = peso / (altura * altura);  
        assertThat(imc).isGreaterThanOrEqualTo(0);  
    }  
  
    @Provide no usages  
    Arbitrary<Double> alturasExtremas() {  
        return Arbitraries.of(1.0, 1.9, 4.5); // Testa alturas incomuns  
    }  
  
    @Provide no usages  
    Arbitrary<Double> pesosExtremos() {  
        return Arbitraries.of(0.1, 0.5, 3.0); // Testa pesos extremos  
    }  
}
```

Run: ImcPropertyTests

Tests passed: 2 of 2 tests - 512 ms

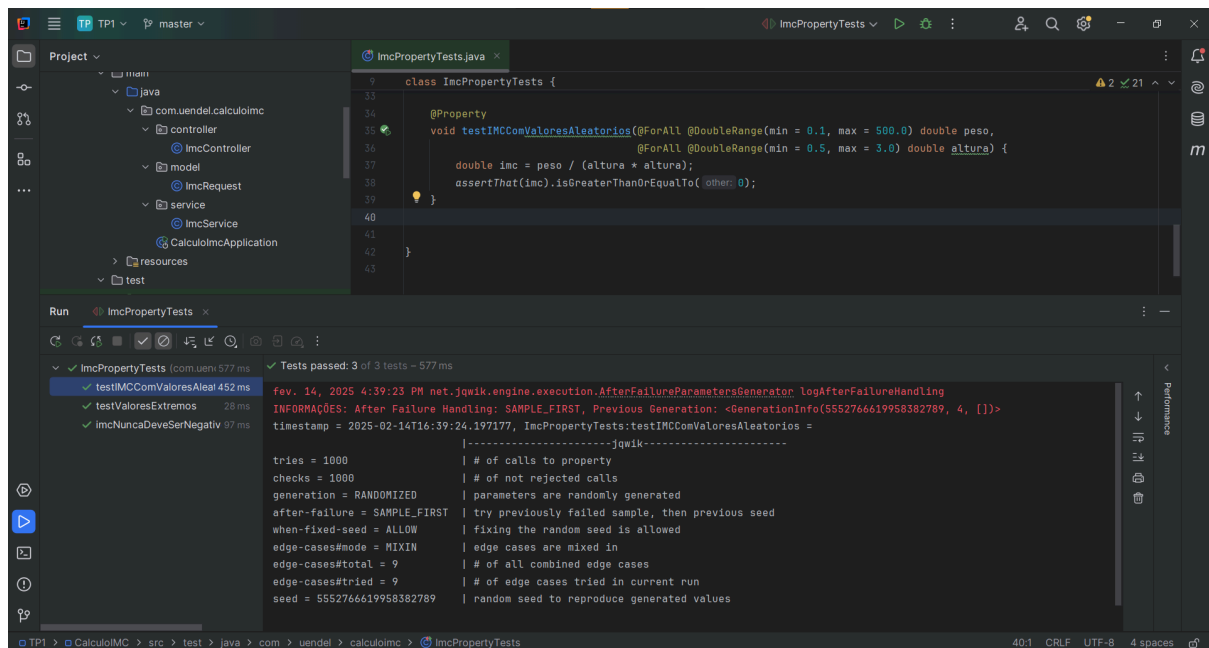
timestamp = 2025-02-14T16:35:09.129759300, ImcPropertyTests:testValoresExtremos =

tries = 10 | # of calls to property
checks = 10 | # of not rejected calls
generation = EXHAUSTIVE | parameters are exhaustively generated
after-failure = SAMPLE_FIRST | try previously failed sample, then previous seed
when-fixed-seed = ALLOW | fixing the random seed is allowed
edge-cases#mode = MIXIN | edge cases are mixed in
edge-cases#total = 0 | # of all combined edge cases
edge-cases#tried = 0 | # of edge cases tried in current run
seed = 1296644094059164392 | random seed to reproduce generated values

timestamp = 2025-02-14T16:35:09.333214900, ImcPropertyTests:ImcNuncaDeveSerNegativo =

4 - Analisar Contraprovações

Evidências:



```
class ImcPropertyTests {  
    @Property  
    void testIMComValoresAleatorios(@ForAll @DoubleRange(min = 0.1, max = 500.0) double peso,  
        @ForAll @DoubleRange(min = 0.5, max = 3.0) double altura) {  
        double imc = peso / (altura * altura);  
        assertThat(imc).isGreaterThanOrEqualTo(0);  
    }  
}
```

Run: ImcPropertyTests

Tests passed: 3 of 3 tests - 577 ms

timestamp = 2025-02-14T16:39:24.197177, ImcPropertyTests:testIMComValoresAleatorios =

tries = 1000 | # of calls to property
checks = 1000 | # of not rejected calls
generation = RANDOMIZED | parameters are randomly generated
after-failure = SAMPLE_FIRST | try previously failed sample, then previous seed
when-fixed-seed = ALLOW | fixing the random seed is allowed
edge-cases#mode = MIXIN | edge cases are mixed in
edge-cases#total = 9 | # of all combined edge cases
edge-cases#tried = 9 | # of edge cases tried in current run
seed = 5552766619958382789 | random seed to reproduce generated values

Caso seja gerado um valor equivalente a 0 , teremos o erro de divisão zero.

5 - Teste Com Casos Específicos

The screenshot displays an IDE interface with the following components:

- Project Explorer:** Shows the project structure for `com.uendel.calculoimc`, including `controller`, `model`, `service`, `resources`, and `test` directories. The `IMCServiceTest` class is selected under `test/java/com.uendel.calculoimc`.
- Code Editor:** Displays the `IMCServiceTest.java` file with the following code:

```
1 package com.uendel.calculoimc;
2
3 import static org.mockito.Mockito.*;
4
5 import com.uendel.calculoimc.service.IMCService;
6 import org.junit.jupiter.api.Test;
7 import static org.assertj.core.api.Assertions.*;
8
9 class IMCServiceTest {
10
11     @Test
12     void testCalculoIMCComMock() {
13         IMCService imcService = mock(IMCService.class);
14         when(imcService.calcularIMC(peso: 80, altura: 1.80)).thenReturn(24.69);
15
16         double imc = imcService.calcularIMC(peso: 80, altura: 1.80);
17         assertThat(imc).isEqualTo(expected: 24.69);
18     }
19 }
```
- Run Console:** Shows the execution of the `IMCServiceTest` class. The output indicates that the test passed successfully:

```
IMCServiceTest [con 1 sec 405 ms]
testCalculoIMCComMock [con 1 sec 405 ms]
```
- Output Console:** Displays the command used to run the test and the resulting output:

```
"C:\Program Files\Eclipse Adoptium\jdk-21.0.6-hotspot\bin\java.exe" ...
[0.662s][error][attach] failure (232) writing result of operation jcmd to pipe \\.\pipe\javatool-1364283448
```