

# JUnit

Prof. Dr. Rafael Galvão de Mesquita

[rafael.mesquita@garanhuns.ifpe.edu.br](mailto:rafael.mesquita@garanhuns.ifpe.edu.br)

# JUnit

- Framework de testes que usa *annotations* (@) para identificar métodos que especificam um teste
- Ferramenta OpenSource
  - <https://junit.org/junit5/>
  - <https://github.com/junit-team/junit5/>

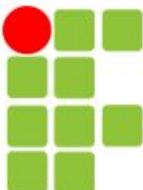
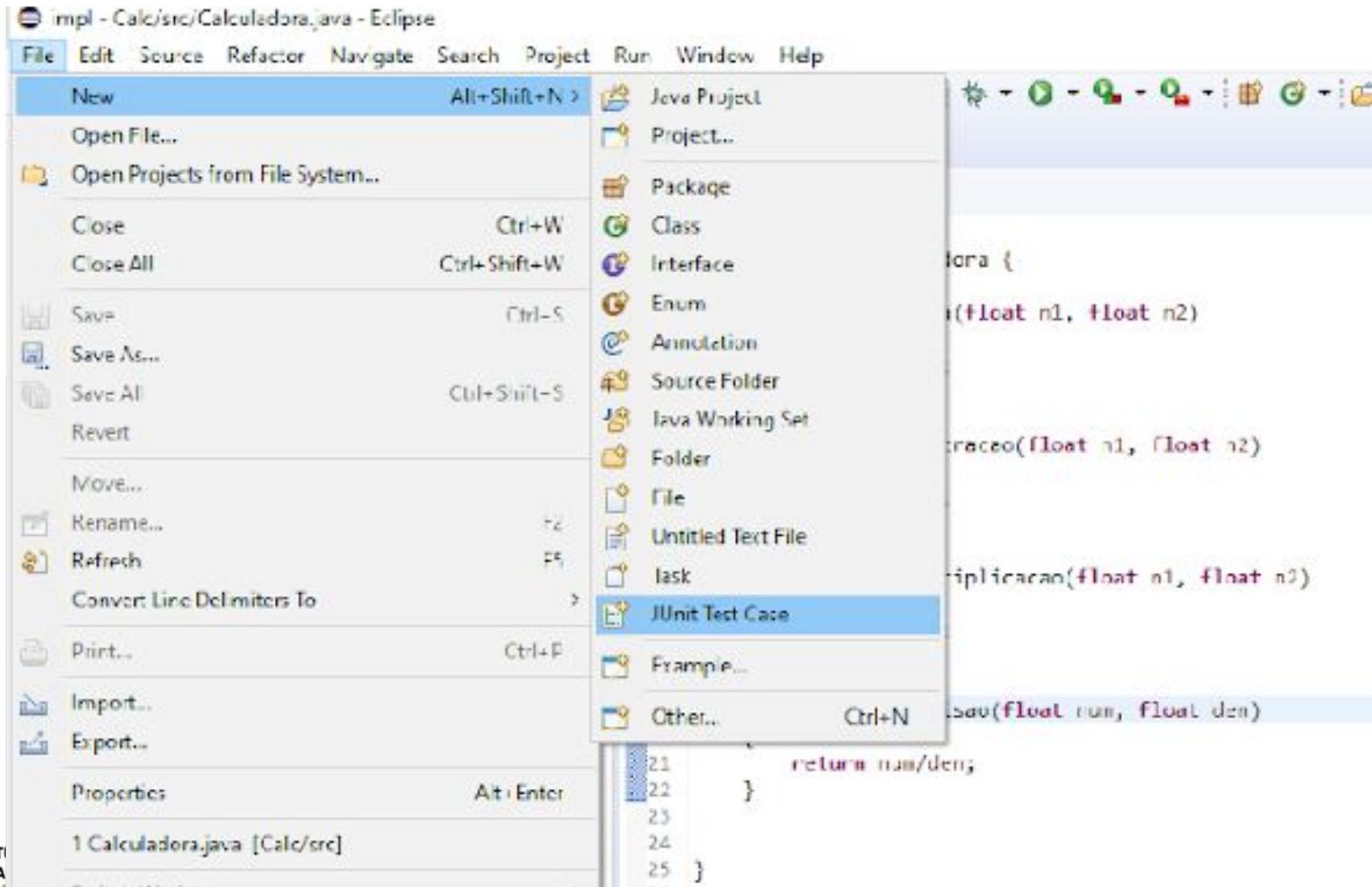
# Criando uma classe de teste

- Classe a ser testada: Calculadora

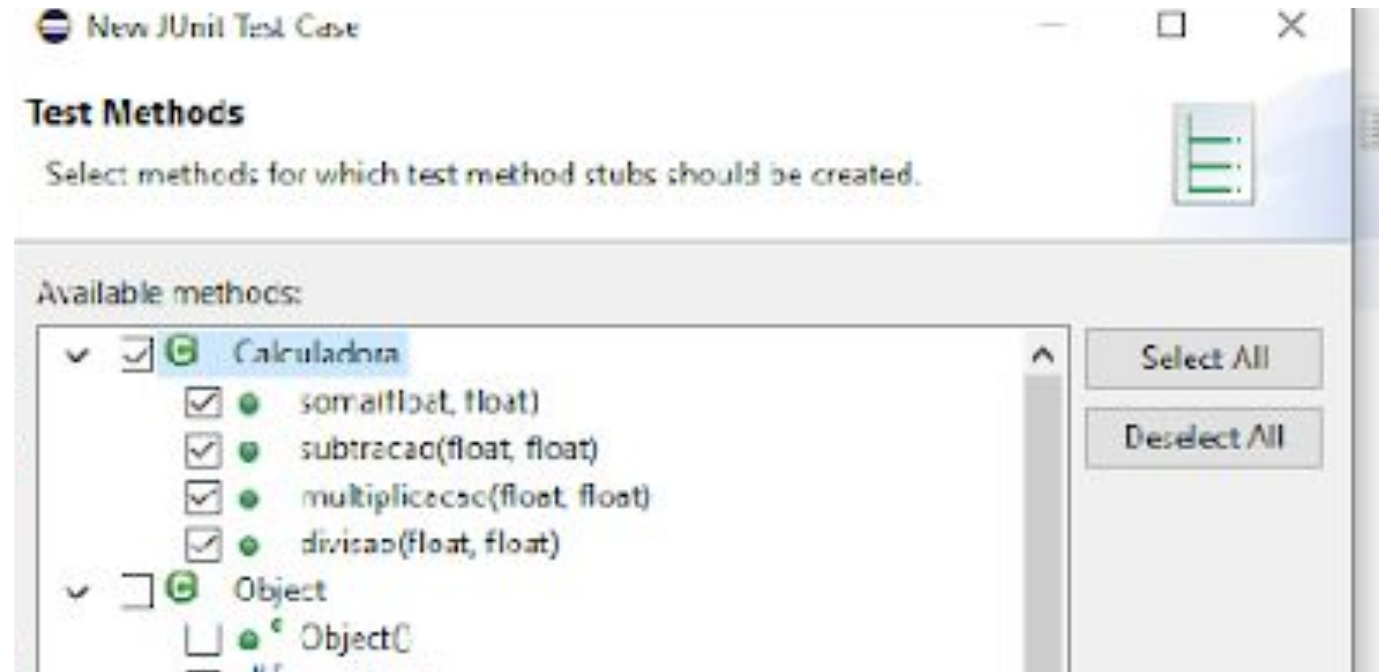
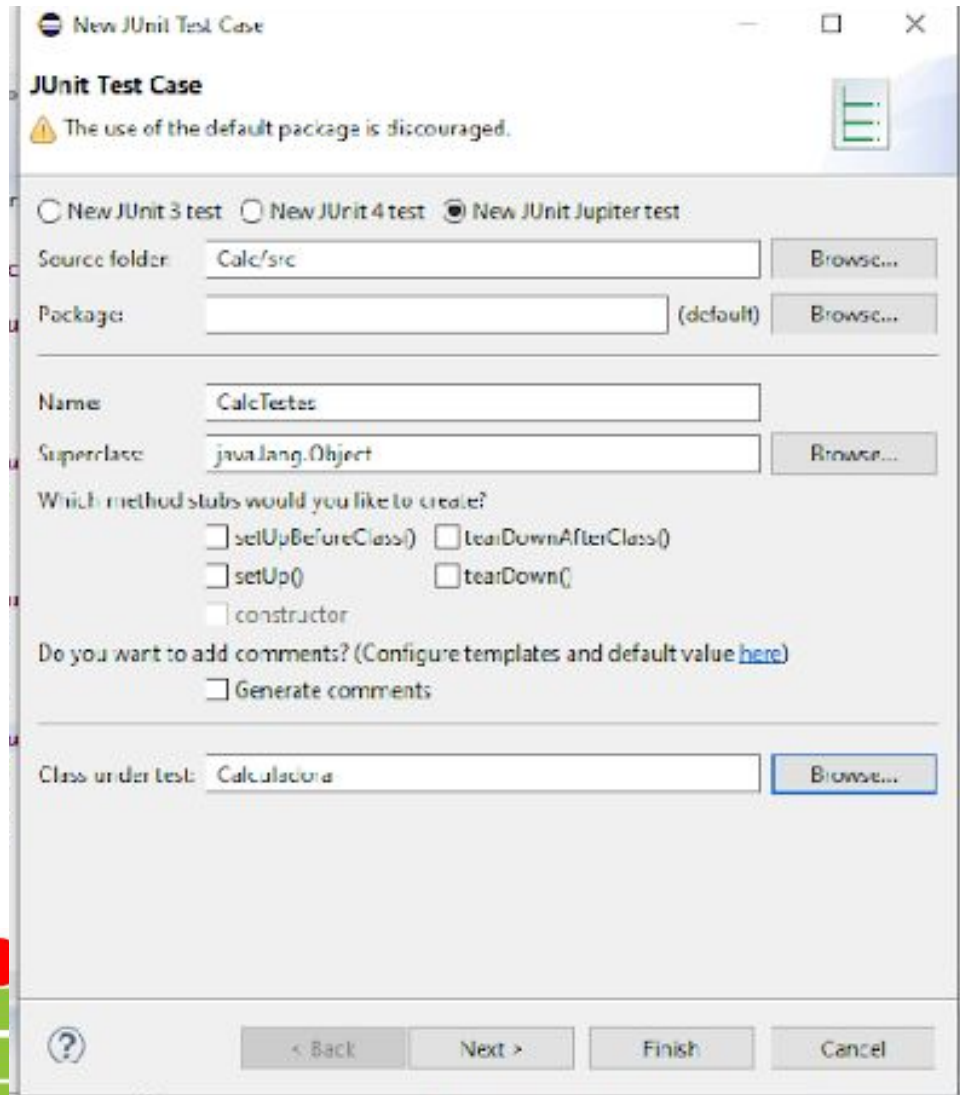
```
2 public class Calculadora {  
3  
4 public float soma(float n1, float n2)[]  
8  
9 public float subtracao(float n1, float n2)[]  
13  
14 public float multiplicacao(float n1, float n2)[]  
18  
19 public float divisao(float num, float den) []  
23  
24 }  
25
```



# Criando uma classe de teste



# Criando uma classe de teste



# Criando uma classe de teste

```
1 import static org.junit.jupiter.api.Assertions.*;
4
5 class CalcTestes {
6
7     @Test
8     void testSoma() {
9         fail("Not yet implemented");
10    }
11
12    @Test
13    void testSubtracao() {
14        fail("Not yet implemented");
15    }
16
17    @Test
18    void testMultiplicacao() {
19        fail("Not yet implemented");
20    }
21
22    @Test
23    void testDivisao() {
24        fail("Not yet implemented");
25    }
26
27 }
28
```



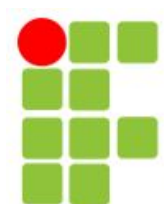
# Programando um método de teste

- O que é um método de teste?
  - Método que se propõe a testar parte de outro código
- Classe de teste
  - Classe que possua pelo menos um método de teste
- *Annotation* @Test
  - Denota que um método é um método de teste
- Método assertEquals
  - Testa se dois valores são equivalentes



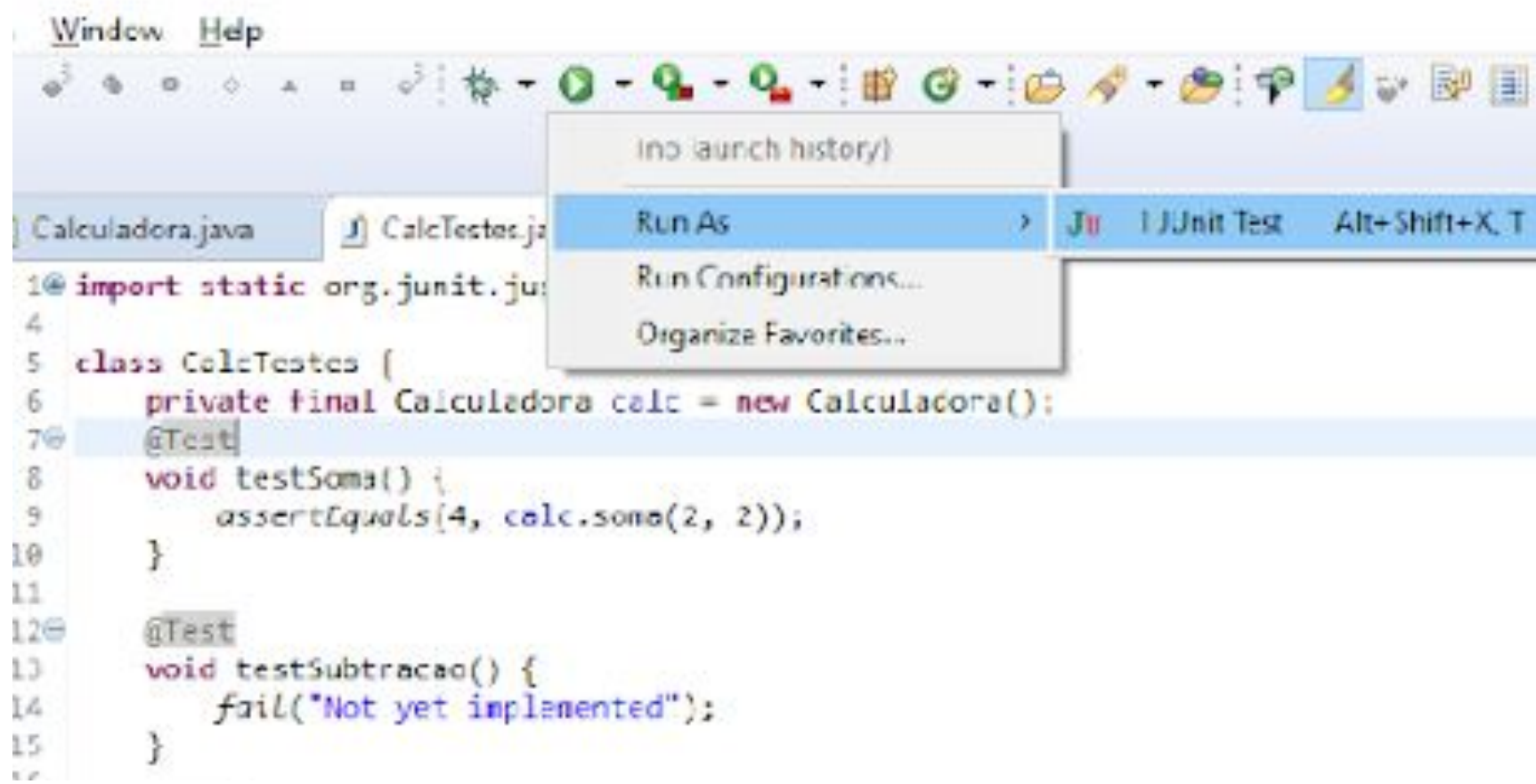
# Programando um método de teste

```
class CalcTestes {  
    private final Calculadora calc = new Calculadora();  
    @Test  
    void testSoma() {  
        assertEquals(4, calc.soma(2, 2));  
    }  
    ...  
}
```

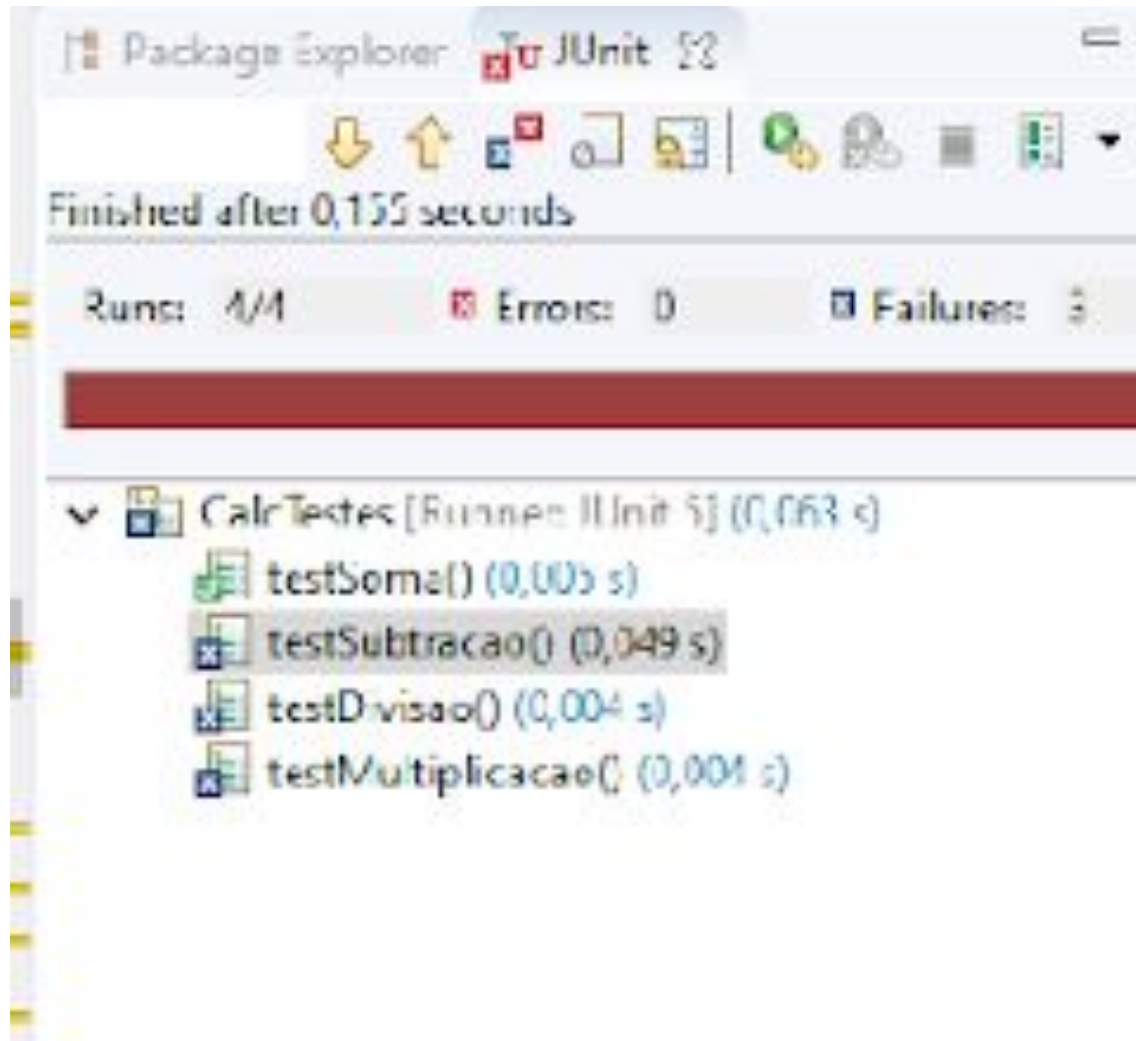







# Executando um método de teste



# Executando um método de teste



# Resultado de um teste

- Sucesso 
  - Caso de teste é executado por completo, como previsto, e não detecta falhas
- Falha 
  - Caso de teste detecta alguma inconsistência entre um valor esperado e um valor que foi obtido
- Erro 
  - Erros não esperados que ocorreram, como um bug no próprio código que executa o teste, ou uma exceção inesperada lançada pelo código testado que não tenha sido tratada no código do teste



# Simulando uma falha

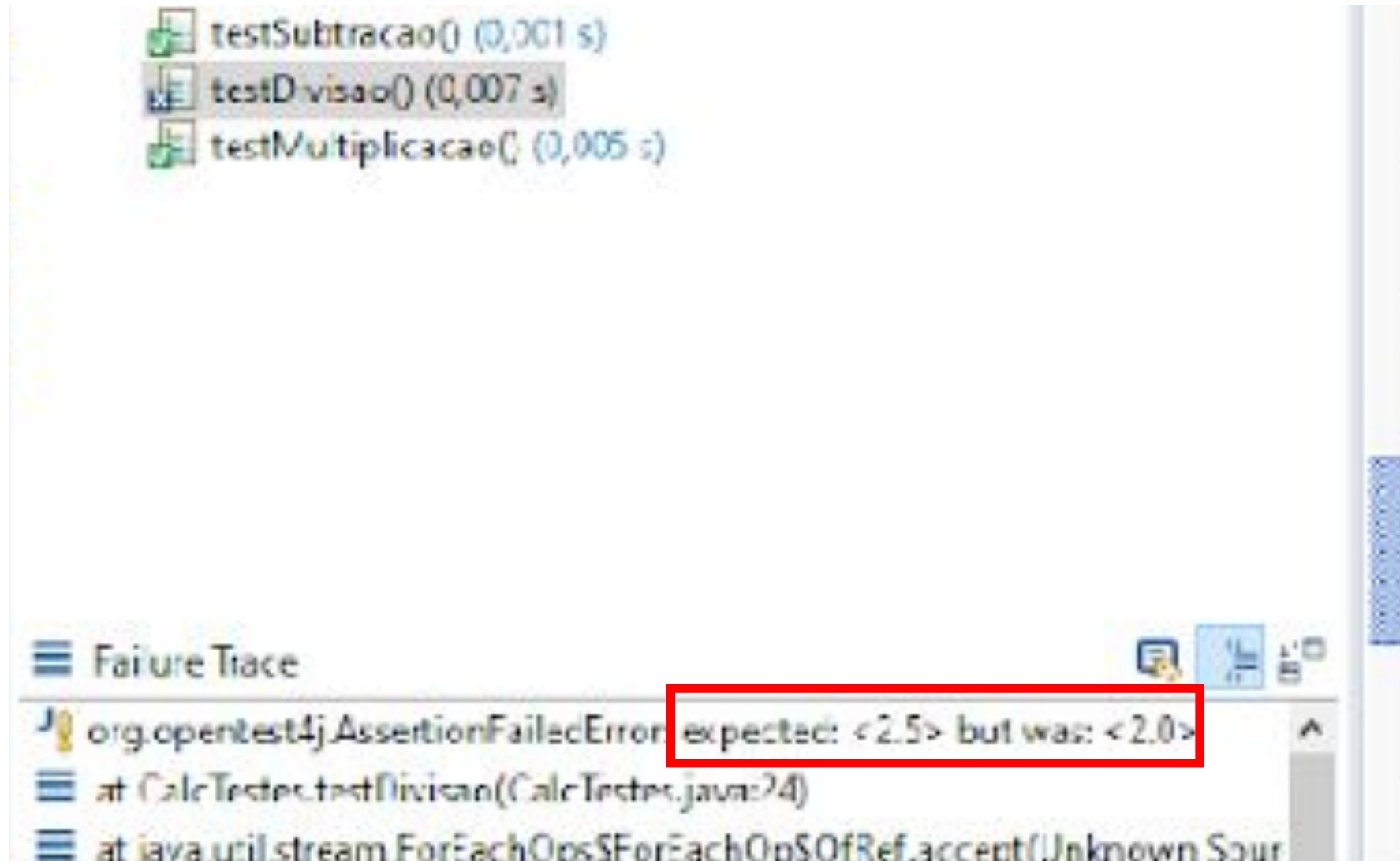
```
public float divisao(int num, int den)
{
    return num/den;
}
```

Divisão inteira!

```
@Test
void testDivisao() {
    assertEquals(2.5, calc.divisao(5, 2));
}
```



# Simulando uma falha



# Testando uma Exceção

- Podemos testar que uma determinada exceção foi lançada
  - `assertThrows`
- Também podemos testar a mensagem de uma exceção lançada
  - `assertEquals`, comparando a mensagem da exceção capturada e o valor de uma mensagem esperada

# Testando uma exceção

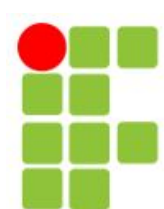
```
public class DivisaoPorZeroException extends Exception {  
  
    private static final long serialVersionUID = 1L;  
  
    public DivisaoPorZeroException(String msg)  
    {  
        super(msg);  
    }  
}
```



# Testando uma exceção

- Definição da exceção

```
public class DivisaoPorZeroException extends Exception {  
  
    private static final long serialVersionUID = 1L;  
  
    public DivisaoPorZeroException(String msg)  
    {  
        super(msg);  
    }  
}
```

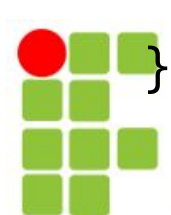




# Testando uma exceção

- Método testado

```
public class Calculadora {  
  
    ...  
  
    public float divisao(float num, float den) throws DivisaoPorZeroException  
    {  
        if(den == 0)  
            throw new DivisaoPorZeroException("denominador nulo!");  
        return num/den;  
    }  
}
```



# Testando uma exceção

- Método de teste

```
class CalcTestes {  
    private final Calculadora calc = new Calculadora();  
    ...  
  
    @Test  
    void testDivisaoPorZero() throws DivisaoPorZeroException {  
        DivisaoPorZeroException exception =  
            assertThrows(DivisaoPorZeroException.class, () -> calc.divisao(5, 0));  
        assertEquals("denominador nulo!", exception.getMessage());  
    }  
}
```



# Função lambda (extra)

- Uma função lambda é uma função sem declaração
  - Não é necessário definir nome, tipo de retorno nem modificador de acesso
  - Uma função lambda pode ser passada como argumento para outras funções
- Sintaxe
  - (argumentos) -> (corpo)
  - No exemplo passado, usamos o seguinte:
    - `assertThrows(DivisaoPorZeroException.class, () -> calc.divisao(5, 0));`
  - Isso quer dizer teremos uma função lambda “`() -> calc.divisao(5, 0)`” sendo passada como segundo argumento do método `assertThrows`
    - A função lambda utilizada como exemplo não recebe parâmetros e executa a chamada de `calc.divisao(5,0)`
    - A função lambda será executada dentro do método `assertThrows`



# Exercício prático

- Crie testes para as 4 operações básicas da calculadora
- Crie uma função potencia, que recebe parâmetros  $x$  e  $y$  e retorna  $x^y$ 
  - Não use funções prontas, como `Math.pow`
  - O expoente deve ser um número inteiro e a base um real
  - Crie casos de teste para testar essa função

# Duvidas?

