Atividade 1: Revisitando a calculadora (1)

1. Escreva uma lista de novos casos de teste para a função de soma:

Baseado nos requisitos de "faixa de valores válidos (positivos e negativos?)", "valores extremos", "exceções para valores inválidos na entrada" e "cálculos para valores decimais", a lista de casos de teste seria:

- **Valores Válidos (Positivos e Negativos):**
    - testSomaDoisPositivos: Testa a soma de dois números positivos (ex: 5 + 2 = 7).
    - testSomaDoisNegativos: Testa a soma de dois números negativos (ex: -5 + -2 = -7).
    - testSomaPositivoNegativo: Testa a soma de um número positivo e um negativo (ex: 5 + -2 = 3).
    - testSomaComZero: Testa a soma de um número com zero (ex: 5 + 0 = 5).
- **Valores Extremos:**
    - testSomaMaxInt: Testa a soma do maior inteiro possível (Integer.MAX_VALUE) com 1. O esperado seria o lançamento de uma exceção (ArithmeticException) ou uma mudança no tipo de retorno (ex: para long).
    - testSomaMinInt: Testa a soma do menor inteiro possível (Integer.MIN_VALUE) com -1, esperando um comportamento similar (exceção).
- **Valores Decimais (Requer refatoração):**
    - testSomaDecimais: Testa a soma de dois números decimais (ex: 1.5 + 2.5 = 4.0). Isso exigiria que o método somar fosse alterado para aceitar e retornar double.
- **Valores Inválidos (Exceções):**
    - No contexto de int somar(int a, int b), a linguagem Java impede entradas inválidas (como strings). As exceções relevantes seriam as de estouro (overflow), conforme listado em "Valores Extremos".

2. Pense nas maneiras de refatorar o código (testes e funcional):

Considerando as opções do JUnit, como "testes com repetição, parametrizados e casadores Hamcrest":

- **Refatoração dos Testes:**
    - **Testes Parametrizados:** Em vez de múltiplos testes assertEquals, poderíamos usar

Atividade 1: Revisitando a calculadora (1)

@ParameterizedTest com @CsvSource no JUnit 5. Isso permitiria testar vários cenários (positivos, negativos, zero) em um único método de teste, tornando o código de teste mais limpo.

- *Exemplo:* @CsvSource({ "5, 2, 7", "-5, -2, -7", "5, 0, 5" })
    - **Casadores Hamcrest:** Para melhorar a legibilidade, assertEquals(7, c.somar(5,2)) poderia ser reescrito com Hamcrest como assertThat(c.somar(5, 2), is(equalTo(7))).
- **Refatoração do Código Funcional:**
    - **Suporte a Decimais:** Para atender ao caso de teste testSomaDecimais , o método public int somar(int a, int b) seria refatorado para public double somar(double a, double b).
    - **Lidar com Overflow (Extremos):** Para lidar com valores muito grandes, poderíamos mudar o retorno para long ou usar Math.addExact(a, b) (disponível desde o Java 8), que lança uma ArithmeticException automaticamente em caso de overflow, satisfazendo o caso de teste testSomaMaxInt.