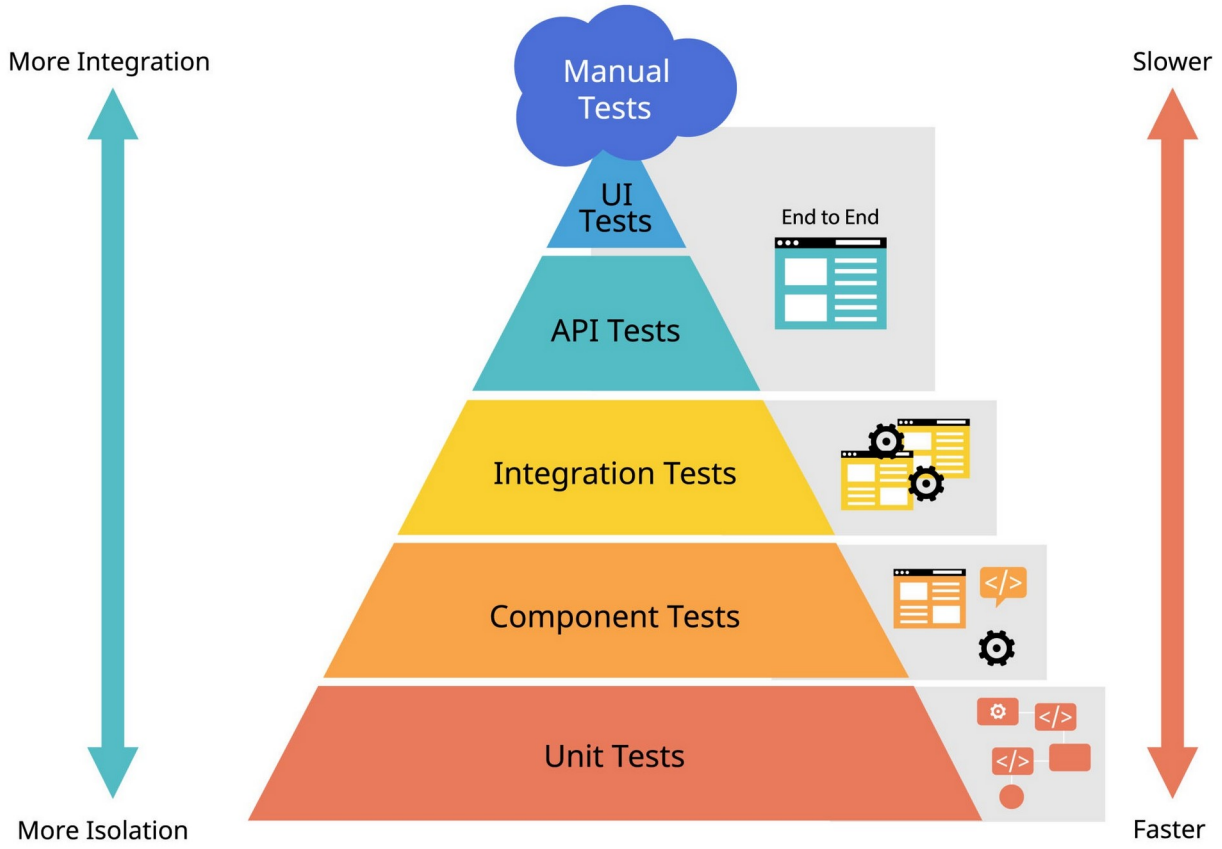
🚀 Tutorial Prático: Primeiros Passos com JUnit 5 e Padrão AAA

Disciplina: Testes Automatizados (Tecnologia em Sistemas para Internet - IFTM) **Professor:** Prof. Dr. Bruno Queiroz Pinto

Neste tutorial, vamos abandonar os testes manuais e criar nossa primeira suíte de testes automatizados utilizando o framework **JUnit 5**. Vamos explorar o padrão de estruturação de testes **AAA** e ver na prática o que acontece quando um código possui *bugs*.



Lembre-se da Pirâmide de Testes: os testes de unidade que faremos hoje formam a base sólida de qualquer software com qualidade!

Pré-requisitos

Como estamos utilizando o ecossistema do **Spring Boot**, você não precisa se preocupar em baixar dependências manualmente. O pacote `spring-boot-starter-test` já inclui o JUnit 5 (Jupiter) pronto para uso!

Passo 1: Criando a Classe de Negócio (A Unidade)

Vamos criar uma classe simples chamada `Calculadora`. Para fins didáticos, **vamos inserir erros intencionais** nos métodos `subtrair` e `multiplicar` para vermos os testes falharem mais à frente.

Crie o arquivo `Calculadora.java`:

```
public class Calculadora {

    public int somar(int num1, int num2) {
        return num1 + num2;
    }

    // ATENÇÃO: Bug lógico inserido intencionalmente (está somando em vez de
subtrair)
    public int subtrair(int num1, int num2) {
        return num1 + num2;
    }

    // ATENÇÃO: Bug lógico inserido intencionalmente (está somando em vez de
multiplicar)
    public int multiplicar(int num1, int num2) {
        return num1 + num2;
    }

    public int dividir(int num1, int num2) {
        return num1 / num2;
    }
}
```

Passo 2: Entendendo o Padrão AAA

Antes de escrever o teste, precisamos entender como organizá-lo. O padrão da indústria é o **AAA** (Arrange, Act, Assert).

1. **Arrange (Preparar):** É onde instanciamos objetos, preparamos o banco de dados em memória, criamos variáveis. É o "palco" do teste.
2. **Act (Agir):** É a chamada do método que queremos testar de fato.
3. **Assert (Verificar/Afirmar):** É onde checamos se o resultado da ação (Act) é igual ao que esperávamos.

Passo 3: Escrevendo o Teste com JUnit 5

Agora, vamos criar a classe de teste `CalculadoraTest.java` na pasta `src/test/java`.
Dica: No JUnit 5, as classes e os métodos de teste não precisam da palavra `public`.

```
class CalculadoraTest {

    @Test
    @DisplayName("Deve somar dois números corretamente")
    void testeSomar() {
        // 1. Arrange (Preparar) - Variáveis bem nomeadas documentam o cenário
        Calculadora calc = new Calculadora();
```

```
        int parcela1 = 10;
        int parcela2 = 5;
        int resultadoEsperado = 15;

        // 2. Act (Agir)
        int resultadoObtido = calc.somar(parcela1, parcela2);

        // 3. Assert (Verificar)
        Assertions.assertEquals(resultadoEsperado, resultadoObtido);
    }

    @Test
    @DisplayName("Deve subtrair dois números corretamente")
    void testeSubtrair() {
        // Arrange
        Calculadora calc = new Calculadora();
        int minuendo = 10;
        int subtraendo = 5;
        int resultadoEsperado = 5;

        // Act
        int resultadoObtido = calc.subtrair(minuendo, subtraendo);

        // Assert - Esperamos 5, mas vai falhar por causa do bug na classe!
        Assertions.assertEquals(resultadoEsperado, resultadoObtido);
    }
}
```

Passo 4: Executando e Analisando os Resultados

Rode a sua classe de teste na IDE.

- O teste de **Soma** vai passar .
- O teste de **Subtração** vai falhar , pois o método implementado retornou `15` (ele somou), mas o teste exigia `5` (resultado do Assert).

🛠️ Passo 5: O Desafio (Exercícios da Aula 03)

Agora é com você! Baseado no que vimos:

1. Vá até a classe `Calculadora` e **conserte** a lógica dos métodos `subtrair` e `multiplicar`.
2. Rode os testes novamente e veja se eles ficam verdes.
3. Crie um novo método na calculadora chamado `ePar(int numero)` que retorna `true` se for par e `false` se for ímpar.
4. Crie os testes para esse novo método garantindo os dois cenários (um número par e um número ímpar).

✅ Gabarito do Professor

Se você já tentou fazer sozinho, confira abaixo como deve ficar a implementação correta:

1. Correção e implementação do novo método na `Calculadora.java`:

```
    // Lógica corrigida!
    public int subtrair(int num1, int num2) {
        return num1 - num2;
    }

    // Lógica corrigida!
    public int multiplicar(int num1, int num2) {
        return num1 * num2;
    }

    // Novo método do exercício
    public boolean ePar(int numero) {
        return numero % 2 == 0;
    }
```

2. Novos testes na `CalculadoraTest.java` para o método `ePar`:

```
@Test
    @DisplayName("Deve retornar TRUE quando o número for par")
    void testeEParRetornaTrue() {
        // Arrange
        Calculadora calc = new Calculadora();
        int numeroPar = 4;

        // Act
        boolean resultadoObtido = calc.ePar(numeroPar);

        // Assert
        Assertions.assertTrue(resultadoObtido, "O número " + numeroPar + "
deveria ser identificado como par.");
    }

    @Test
    @DisplayName("Deve retornar FALSE quando o número for ímpar")
    void testeEParRetornaFalse() {
        // Arrange
        Calculadora calc = new Calculadora();
        int numeroImpar = 7;

        // Act
        boolean resultadoObtido = calc.ePar(numeroImpar);

        // Assert
        Assertions.assertFalse(resultadoObtido, "O número " + numeroImpar + "
deveria ser identificado como ímpar.");
    }
```