JUnit

▼ O que é

JUnit é um framework de testes unitários de código aberto para a linguagem de programação Java. Ele facilita a criação e a execução de testes automatizados, garantindo a qualidade do seu código.

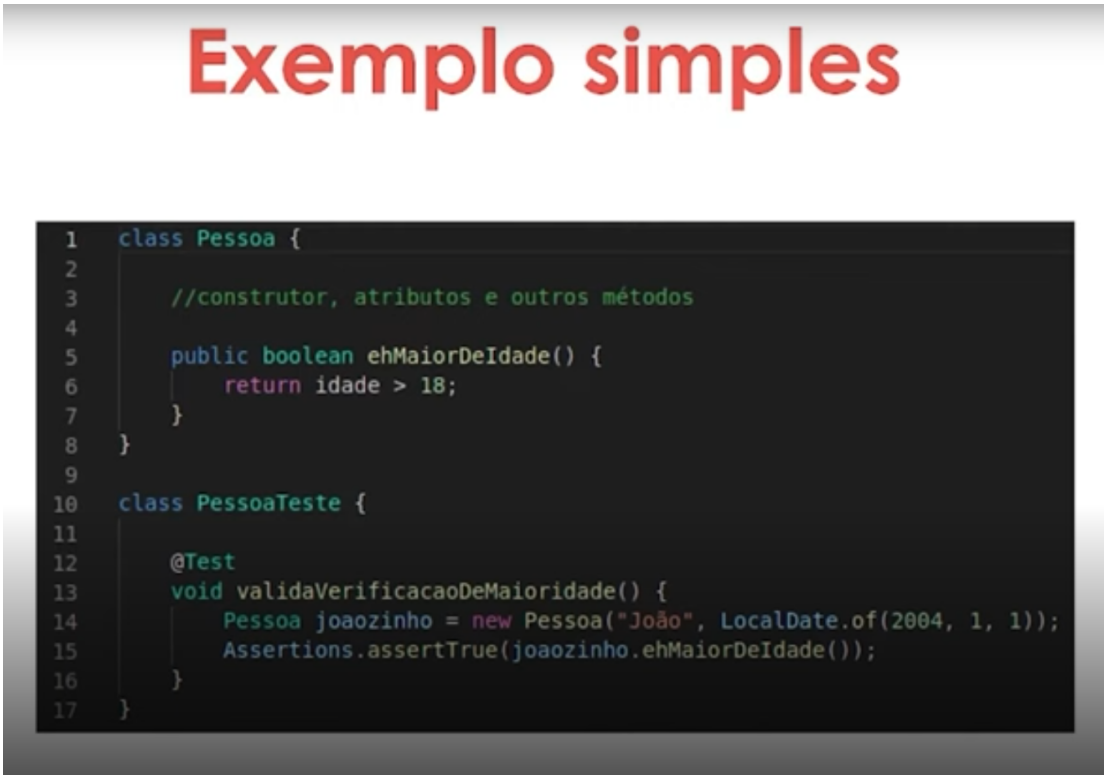
▼ Para que serve

JUnit permite:

- **Testar unidades de código** (classes, métodos) de forma **isolada**, garantindo que funcionem como esperado.
- **Automatizar a execução** dos testes, economizando tempo e esforço.
- **Obter feedback rápido** sobre a qualidade do código, identificando erros e falhas.
- **Melhorar a confiabilidade** e a robustez do software.

▼ Exemplo

Exemplo simples



Classe Java

```
public class Pessoa {
private String nome;
private int idade;

public Pessoa(String nome, int idade) {
    this.nome = nome;
    this.idade = idade;
}

public boolean isMaiorDeIdade() {
    return idade >= 18;
}
}
```

Classe para teste

```
@Test
public void testPessoaMaiorDeIdade() {
    // Cria uma nova pessoa com nome "João" e idade 2
0
    Pessoa pessoa = new Pessoa("João", 20);

    // Verifica se a pessoa é maior de idade
    if (!pessoa.isMaiorDeIdade()) {
        // Lança uma exceção AssertionError se a pess
oa não for maior de idade
        throw new AssertionError("A pessoa " + pesso
a.nome + " não é maior de idade!");
    }

    // Imprime uma mensagem se a pessoa for maior de
idade
    System.out.println("A pessoa " + pessoa.nome + "
é maior de idade!");
}
}
```

▼ Versões

Versões do JUnit:

Versão mais nova:

A versão mais nova do JUnit, em fevereiro de 2024, é a **JUnit 5.9.0**. Essa versão foi lançada em dezembro de 2023 e inclui diversas melhorias e correções de bugs.

Versão mais utilizada:

A versão mais utilizada do JUnit, ainda em fevereiro de 2024, é a **JUnit 4.13.2**. Essa versão foi lançada em junho de 2017 e ainda é muito popular devido à sua maturidade e grande base de usuários.

É dividido em:

▼ **JUnit Platform**

- É a base para o desenvolvimento de frameworks de testes em Java.
- Fornece APIs para executar testes e gerenciar o ciclo de vida dos testes.
- Permite a integração com diferentes ferramentas de build e IDEs.

▼ **JUnit Jupiter**

- É um framework de testes moderno que implementa a JUnit Platform.
- Fornece anotações para escrever testes unitários de forma simples e intuitiva.
- Suporta diversos recursos, como testes parametrizados, injeção de dependências e assertions.

▼ **JUnit Vintage**

- É um framework de testes que permite executar testes JUnit 3 e JUnit 4 na JUnit Platform.
- É útil para projetos que ainda possuem testes escritos em versões antigas do JUnit

Em resumo:

- JUnit Platform: base para frameworks de testes.
- JUnit Jupiter: framework de testes moderno.
- JUnit Vintage: permite executar testes JUnit 3 e JUnit 4 na JUnit Platform.

▼ Como Instalar

Para utilizar JUnit em seus projetos, deve-se adicioná-lo nas dependências de um gerenciador de dependências **(Maven ou Gradle)**. Ao adentrar o site MVN Repository, deve-se procurar por JUnit Engine e adiciona-lo ao projeto

Versão mais recente:
https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine/5.10.2

Dependência Maven:

<dependency>
<groupId>org.junit.jupiter</groupId>
<artifactId>junit-jupiter-engine</artifactId>
<version>5.10.2</version>
<scope>test</scope>
</dependency>

Como funciona

Comandos

Assertion

No contexto do JUnit, assertions (afirmações) são verificações que você faz sobre o estado do seu código após a execução de um teste. Elas permitem que você valide se o código está se comportando como esperado.

Tipos

Exemplos de assertions:

Verificar se um valor é igual a outro:

```
Assertions.assertEquals(10, soma);
```

Verificar se um valor é nulo:

```
Assertions.assertNull(objeto);
```

Verificar se uma exceção foi lançada:

```
Assertions.assertThrows(ArithmeticException.c
lass, () -> {
    divisaoPorZero();
});
```

▼ **assertTrue:**

- Leva um argumento booleano.
- Se o argumento for **verdadeiro**, o teste passa silenciosamente.
- Se o argumento for **falso**, o teste **falha** e uma mensagem de erro é lançada, indicando a condição que falhou.

  Exemplo:

  ```
  int age = 25;
  assertTrue(age >= 18); // Passa porque 25
  é maior ou igual a 18
  ```

▼ **assertFalse:**

- Leva um argumento booleano.
- Se o argumento for **falso**, o teste passa silenciosamente.
- Se o argumento for **verdadeiro**, o teste **falha** e uma mensagem de erro é lançada, indicando a condição que falhou.

Exemplo:

```
String name = "John";
assertFalse(name.isEmpty()); // Passa porque
"John" não é vazio
```

▼ assertArrayequals

O método `assertArrayEquals` do JUnit é usado para verificar se dois arrays são iguais. Ele compara cada elemento dos arrays para garantir que tenham os mesmos valores e a mesma ordem. Se os arrays forem diferentes, o teste falhará e uma mensagem de erro será exibida indicando a diferença entre os arrays.

Exemplo:

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions;

public class AssertionsTest {

    @Test
     public void testsAssertion(){
    int[] firstArray = {10, 20, 30};
    int[] secondArray = {1, 2, 3};

    Assertions.assertArrayEquals(firstArray, se
```

▼ assertArrayNotEquals

O método `assertArrayNotEquals` do JUnit é usado para verificar se dois arrays são diferentes. Ele compara cada elemento dos arrays para garantir que tenham valores diferentes ou uma ordem diferente. Se os arrays forem iguais, o teste falhará e uma mensagem de erro será exibida indicando a igualdade entre os arrays.

Exemplo:

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions;

public class AssertionsTest {

    @Test
```

```
     public void testsAssertion(){
    int[] firstArray = {10, 20, 30};
    int[] secondArray = {1, 2, 3};


    Assertions.assertNotEquals(firstArray, seco

    }
}
```

▼ assertNotNull

O método `assertNotEquals` do JUnit é usado para verificar se dois valores são diferentes. Se os valores forem iguais, o teste falhará e uma mensagem de erro será exibida indicando a igualdade entre os valores.

Exemplo:

```
@Test
    public void personNullTest(){
        Person james = new Person("James", Loca
        Assertions.assertNotNull(james);
    }
```

▼ assertDoesNotThrow

é uma função utilizada em testes unitários com JUnit em Java para verificar se um trecho de código não lança exceções. Em outras palavras, ela verifica se uma parte específica do seu código é capaz de executar sem gerar uma exceção.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.

public class MinhaClasseTest {
```

```
    @Test
    public void testMetodoQueNaoDeveLancarExcec
        assertDoesNotThrow(() -> {
            // Trecho de código que não deve la
            MinhaClasse.metodoQueNaoDeveLancarE
        });
    }
}
```

▼ **assertThrows**

é uma função utilizada em testes unitários com JUnit em Java para verificar se um trecho de código lança uma exceção esperada. Em outras palavras, ela permite testar se uma parte específica do seu código está gerando a exceção correta quando uma condição específica é atendida.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.

public class MinhaClasseTest {

    @Test
    public void testMetodoQueDeveLancarExcecao(
        assertThrows(IllegalArgumentException.c
            // Trecho de código que deve lançar
            MinhaClasse.metodoQueDeveLancarExce
        });
    }
}
```

▼ **Como diminuir o código**

Importando todos os Assertions exclui a necessidade de toda vez escrever *"Assertions.**comando**"*

```
import static org.junit.jupiter.api.Assertions.*;
```

Ficando assim os códigos de **Assertion:**

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions;

import java.time.LocalDateTime;

import static org.junit.jupiter.api.Assertions.*;


public class AssertionsTest {

    @Test
     public void testsAssertion(){
    int[] firstArray = {10, 20, 30};
    int[] secondArray = {1, 2, 3};

    assertArrayEquals(firstArray, secondArray);
    assertNotEquals(firstArray, secondArray);

    }

    @Test
    public void personNullTest(){
        Person james = new Person("James", LocalDa
        assertNotNull(james);
    }
}
```

▼ **Exemplos**

Classe Person

```
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;

public class Person {
    private String name;

    private LocalDateTime birthday;

    public Person(String name, LocalDateTime birth
        this.name = name;
        this.birthday = birthday;
    }

    public int getAge(){
        //ChronoUnit.YEARS.between is used to prov
        // subject is between two parameters
        return (int) ChronoUnit.YEARS.between(birt
    }

    public boolean isLegalAge(){
        return getAge() >=18;
    }
}
```

Classe TestPerson (Para realizar os testes)

```
import org.junit.Test;
import org.junit.jupiter.api.Assertions;

import java.time.LocalDateTime;

public class TestPerson {

    @Test
   public void CalculateAgeCorrectly(){
        Person joey = new Person("Joey", LocalDate
```

```
        Assertions.assertEquals(24, joey.getAge())
    }
}
```

Resultado do teste:

▼ Before & After

▼ Exemplo:

▼ Classe Database

```
import java.util.logging.Logger;

public class DataBase {
    private static final Logger LOGGER = Logger

    public void startConnection(){
        LOGGER.info("Start Connection");
    }

    public void closeConnection(){
        LOGGER.info("Close Connection");
    }

    public void addData(){
        LOGGER.info("Add Data Test");
    }

    public void dropData(){
        LOGGER.info("Drop Data Test");
```

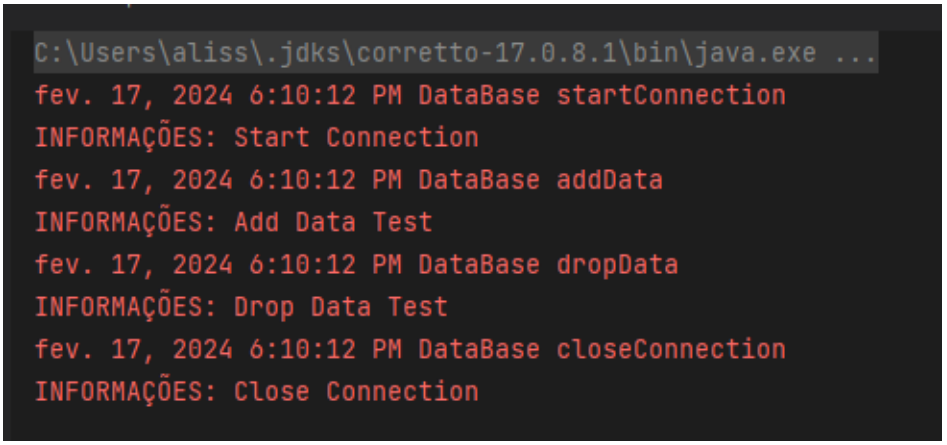
```
    }
}
```

▼ **Classe utilizando Before & After**

```
import org.junit.jupiter.api.*;

import java.time.LocalDateTime;

import static org.junit.jupiter.api.Assertions.

public class DataBaseTest {
    static DataBase dataBase = new DataBase();
    Person tester = new Person("Joey", LocalDat

    @BeforeAll
    static void startConnection(){
        dataBase.startConnection();
    }

    @BeforeEach
    void addData(){
        dataBase.addData(tester);
    }

    @Test
    void testDatabase(){
        assertNotNull(tester);
    }

    @AfterEach
    void dropData(){
        dataBase.dropData();
    }

    @AfterAll
    static void closeConnection(){
        dataBase.closeConnection();
```

```
    }
}
```

Console:



- **@BeforeAll**

  Código rodado antes de todos os testes

- **@BeforeEach**

  Código rodado antes de cada teste

- **@Test**

  Código realizado em cada teste

- **@AfterEach**

  Código rodado depois de cada teste individualmente

- **@AfterAll**

  Código rodado depois de todos os testes

- **Assumption**
  - **Explicação**

Em JUnit, "assumptions" (ou "suposições") são declarações que ajudam a verificar as condições necessárias para que um teste possa ser executado com sucesso. Se uma suposição falha durante a execução de um teste, o teste é interrompido e marcado como "ignorado", pois as condições necessárias para sua execução não foram atendidas. Isso é útil quando você tem pré-requisitos que devem ser cumpridos para que um teste tenha sentido, e se esses pré-requisitos não forem atendidos, não faz sentido continuar com o teste.

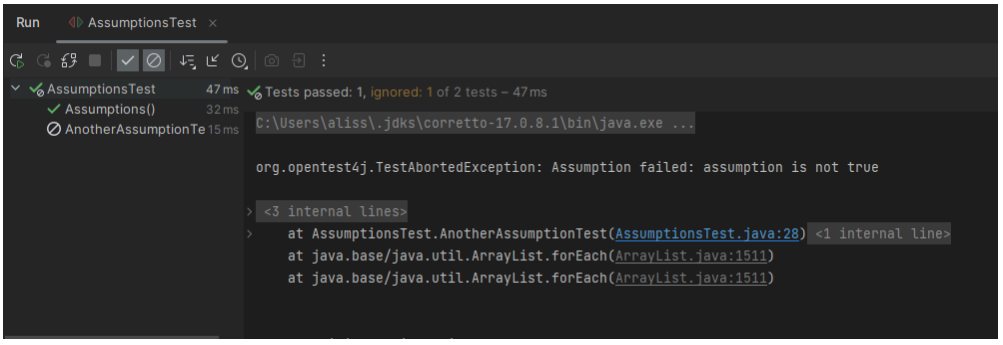
▼ **Exemplo**

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Assumptions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.condition.*;

import java.time.LocalDateTime;

public class AssumptionsTest {
    @Test
    //A usage of Assumptions to make sure this tes
    @EnabledOnOs(OS.WINDOWS)
    //A usage of Assumptions to make sure this tes
    //@EnabledOnJre(JRE.JAVA_17)
    //A usage of Assumptions to make sure this tes
    // between Java 10 and Java 17
    @EnabledForJreRange(min = JRE.JAVA_10, max = J
    void Assumptions(){
        Person person1 = new Person("Joelmer", Loc
        Person person2 = new Person("John", LocalD
        Assertions.assertNotEquals(person2, person
    }

    @Test
    void AnotherAssumptionTest(){
        Person person1 = new Person("Joelmer", Loc
        //A usage of Assumptions to make the test
        // if "aliss" == "aliss"
```

```
        Assumptions.assumeTrue("aliss".equals("ali
        Assertions.assertNotNull(person1);
    }
}
```

Console (Passando nas Assumptions)

Console (Passando apenas em uma assumption)



▼ Exceptions

▼ Explicação

Em Java, as exceptions são usadas para lidar com situações excepcionais ou erros durante a execução do programa. No contexto do JUnit, as exceptions são usadas para verificar se o comportamento esperado do código está ocorrendo corretamente.

▼ Exemplo

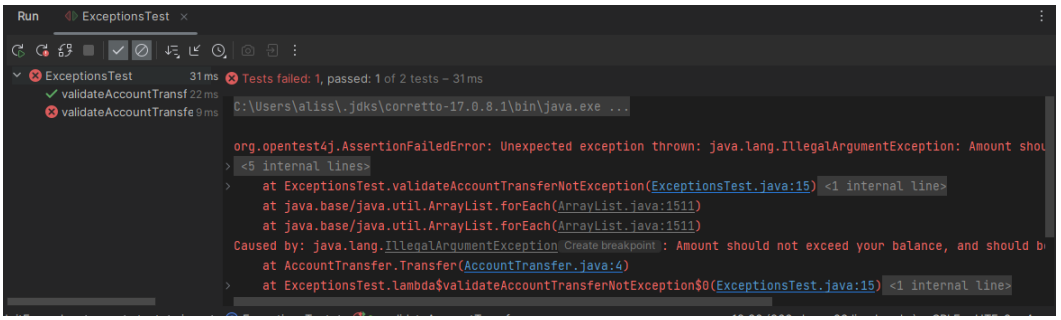
```
public class AccountTransfer {
    public void Transfer(Account account1, Account
        if (amount <= 0 || amount < account1.getBa
            throw new IllegalArgumentException("Am
        }
    }
}
```

Uma exceção será lançada caso o valor seja menor ou igual a zero, ou se a conta não tiver o valor. Para testar se a exceção está sendo lançada:

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assert

public class ExceptionsTest {
   @Test
    void validateAccountTransferNotException(){
       Account account1 = new Account("Karl", 10);
       Account account2 = new Account("Joseph", 100);

       AccountTransfer accountTransfer = new AccountT

       //AssertDoesNotThrow verifies if no exception
       Assertions.assertDoesNotThrow(() -> accountTra
   }

    @Test
    void validateAccountTransfer(){
        Account account1 = new Account("Karl", 10);
        Account account2 = new Account("Joseph", 100)

        AccountTransfer accountTransfer = new Account
```

```
        //AssertThrows verifies if the specified exce
        assertThrows(IllegalArgumentException.class,
    }
}
```

Console:



O primeiro teste falha, pois ele joga uma exceção, enquanto o segundo passa, pois verifica se uma exceção será lançada

▼ Como ordenar Testes

▼ OrderAnnotation

Ordena escolhendo individualmente as ordens, utilizando @TestMethodOrder(MethodOrderer.OrderAnnotation.class)

```
import org.junit.jupiter.api.*;


@TestMethodOrder(MethodOrderer.OrderAnnotation.cla
public class SortTests {

    @Order(1)
    @Test
    void ValidateA(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
```

```
    @Order(4)
    @Test
    void ValidateB(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
    @Order(2)
    @Test
    void ValidateC(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
    @Order(3)
    @Test
    void ValidateD(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
}
```

▼ MethodName

Ordena de forma alfabética, pelo nome dos métodos, utilizando

`@TestMethodOrder(MethodOrderer.MethodName.class)`

```
import org.junit.jupiter.api.*;


//@TestMethodOrder(MethodOrderer.OrderAnnotation.c
@TestMethodOrder(MethodOrderer.MethodName.class)
public class SortTests {

   // @Order(1)
    @Test
```

```
    void ValidateA(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
  //  @Order(4)
    @Test
    void ValidateB(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
   // @Order(2)
    @Test
    void ValidateC(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
   // @Order(3)
    @Test
    void ValidateD(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
}
```

▼ Random

Ordena de forma aleatória, utilizando

`@TestMethodOrder(MethodOrderer.Random.class)`

```
import org.junit.jupiter.api.*;


//@TestMethodOrder(MethodOrderer.OrderAnnotation.c
//@TestMethodOrder(MethodOrderer.MethodName.class)
@TestMethodOrder(MethodOrderer.Random.class)
public class SortTests {

   // @Order(1)
    @Test
    void ValidateA(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
  //  @Order(4)
    @Test
    void ValidateB(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
   // @Order(2)
    @Test
    void ValidateC(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
   // @Order(3)
    @Test
    void ValidateD(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
}
```

▼ **DisplayName**

Ordena de forma alfabética, a partir do displayName dado, utilizando

@TestMethodOrder(MethodOrderer.DisplayName.class)

```
import org.junit.jupiter.api.*;


//@TestMethodOrder(MethodOrderer.OrderAnnotation.c
//@TestMethodOrder(MethodOrderer.MethodName.class)
//@TestMethodOrder(MethodOrderer.Random.class)
@TestMethodOrder(MethodOrderer.DisplayName.class)
public class SortTests {

   // @Order(1)
    @DisplayName("D")
    @Test
    void ValidateA(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
  //  @Order(4)
  @DisplayName("B")
    @Test
    void ValidateB(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
   // @Order(2)
   @DisplayName("C")
    @Test
    void ValidateC(){
        Account account1 = new Account("test1", 2)
        Account account2 = new Account("test2", 2)
        Assertions.assertNotEquals(account1, accou
    }
```

```
    // @Order(3)
    @DisplayName("A")
     @Test
     void ValidateD(){
         Account account1 = new Account("test1", 2)
         Account account2 = new Account("test2", 2)
         Assertions.assertNotEquals(account1, accou
     }
}
```

▼ Material de Apoio

▼ Slides

https://docs.google.com/presentation/d/1XJfYH_8L2d79y1PbSust98XP19BJf8r3/edit#slide=id.p2

▼ Exemplos

https://github.com/AlissonMM/Unit-Testing-with-JUnit

https://github.com/willyancaetano/junit5-exemplos