

Guia Basico JUnit

Raul Nogueira¹

¹Universidade Federal do Pampa – UNIPAMPA
Alegrete, RS – Brasil

Abstract. *The document attempts both to explain the concept of testing and the implementation of the JUnit testing framework.*

Resumo. *Esse documento tem o proposito de apresentar de forma sucinta o JUnit e também o conceito de testes.*

1. Introdução

Teste é o processo de checar as funcionalidades de uma aplicação para garantir que todos os requisitos são atendidos, isso nos leva aos conceitos de Test Driven Development (TDD) e Behavior Driven Development (BDD) ambos são metodologias que unem praticas ageis, com a finalidade de entregar o software com o maior nivel de qualidade possivel.

Com a crescente popularização das metodologias Ageis, a demanda por sistemas de qualidade de software aumentaram gradativamente conforme os anos e junto com essa popularização o JUnit se tornou um dos frameworks de teste mais utilizados. [Team 2021b]

2. Test Driven Development e Boas praticas Testes

Test Driven Development ou Desenvolvimento guiado por testes, define que todo novo codigo gerado deve também gerar um teste, o TDD se concentra em 3 principais atividades nomeadas de Red, Green e Refactor.[Warcholinski 2021]

- **1º Red:** É escrito um teste que não vai funcionar. (Testar as possiveis inconsistencias do software)
- **2º Green:** Faça correções até que o codigo esteja funcionando. (Geralmente o aqui pode ser executado um teste de verificação acima)
- **3º Refactor:** Refatoramos o codigo removendo duplicidades e melhorando a qualidade e manutenibilidade do Software.

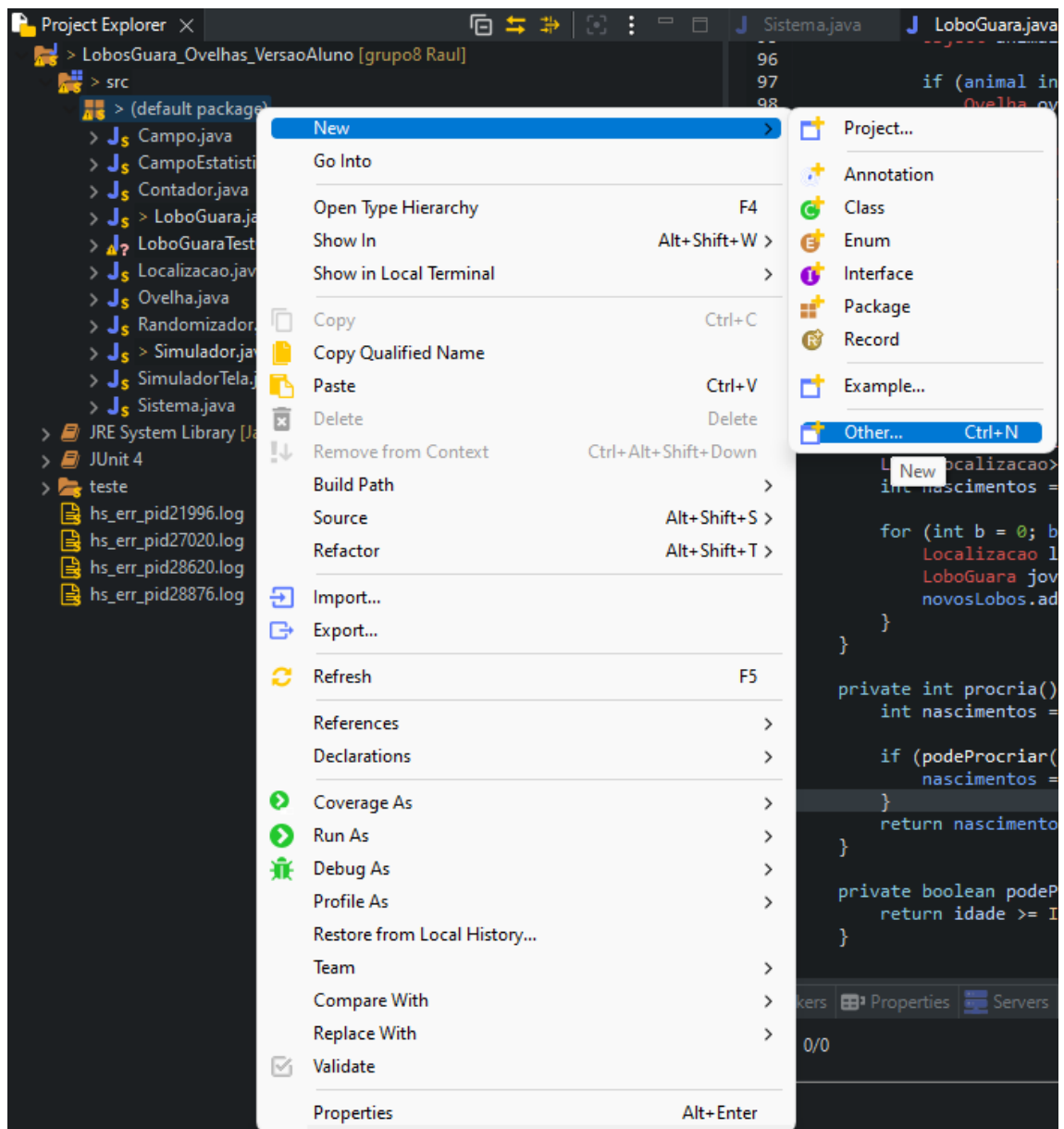
Praticas que podem ser tomadas ao utilizar o TDD na criação dos testes:

- **1º Deixar claro** todas as características dos testes e também tudo que deve ser testado.
- **2º Os casos de testes** devem ser simples e objetivos, com a finalidade de garantir a cobertura do software.
- **3º Os testes** devem ser independentes ou seja, os testes podem ser executados em grupo ou individualmente.
- **4º O processo de teste** deve ser documentado do inicio ao fim, para facilitar a manutenibilidade.
- **5º Respeite o ciclo de testes**, não exclua nem altere testes antigos, eles são parte da cobertura da aplicação.

3. JUnit

Com base nas informações adicionadas anteriormente, agora podemos falar um pouco sobre o JUnit, JUnit é um framework que facilita o desenvolvimento utilizando o junit framework e a execução de testes de software utilizando a test Runner.

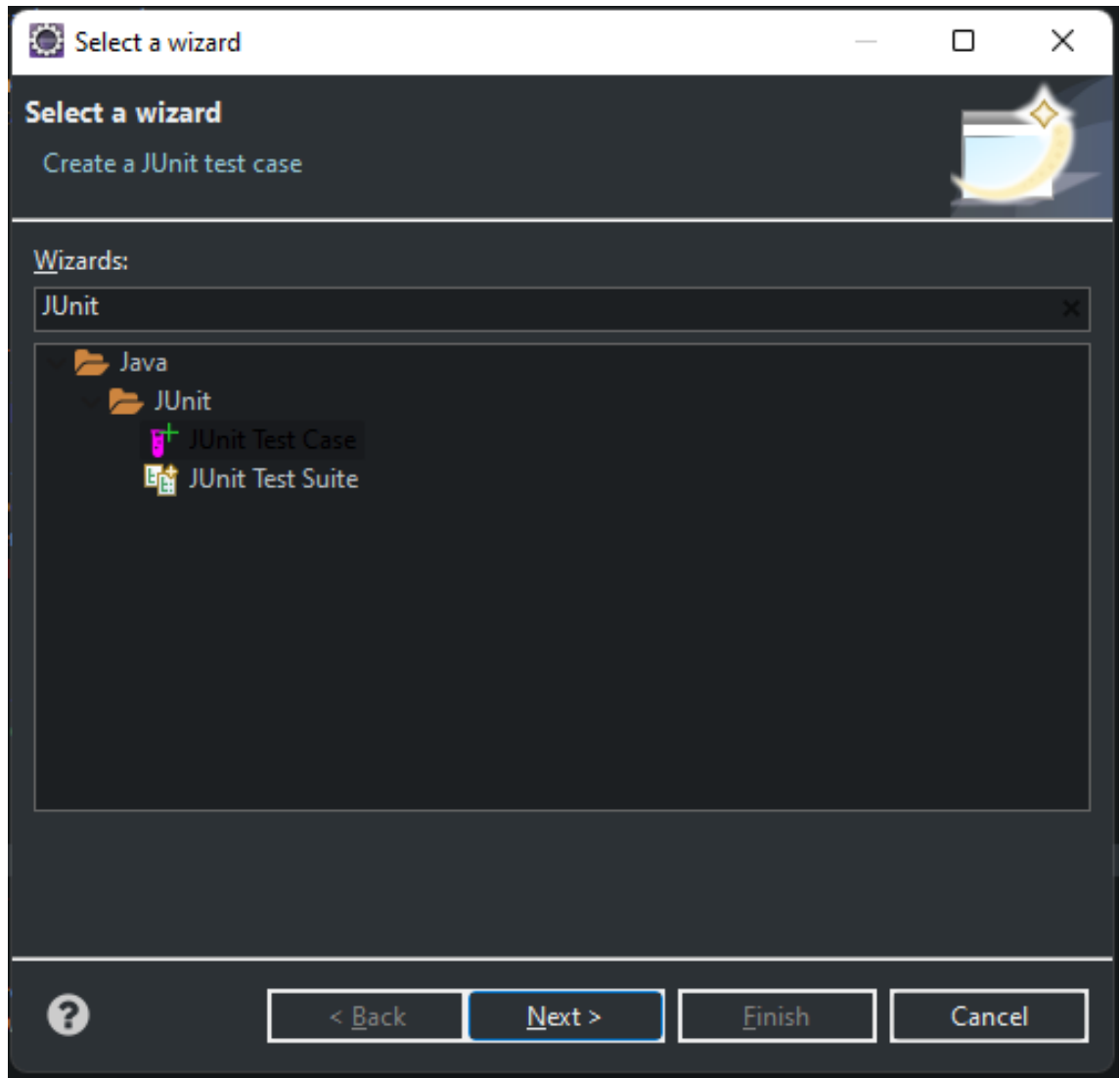
Hoje as principais IDE's do mercado IntelliJ IDEA, Eclipse, NetBeans, e Visual Studio Code, já possuem o JUnit 5 integrados a JUnit na sua 5 versão possui tanto JUnit Jupiter quanto JUnit Vintage, que não iremos abordar nesse documento, mas que podem ser utilizados de acordo com a necessidade de cada projeto. [Team 2021a] No projeto que estamos atuando nesse momento padronizamos a utilização do Eclipse em sua versão mais atual (2021-09 4.2.0), portanto o JUnit já está integrado a IDE, logo para gerar os casos de testes podemos seguir os passos que irei adicionar abaixo:



Uma vez que você esteja com o projeto aberto no Eclipse, clique com o botão direito na package, clique em "New" e Logo após "Other" ou utilize o comando Ctrl + N no

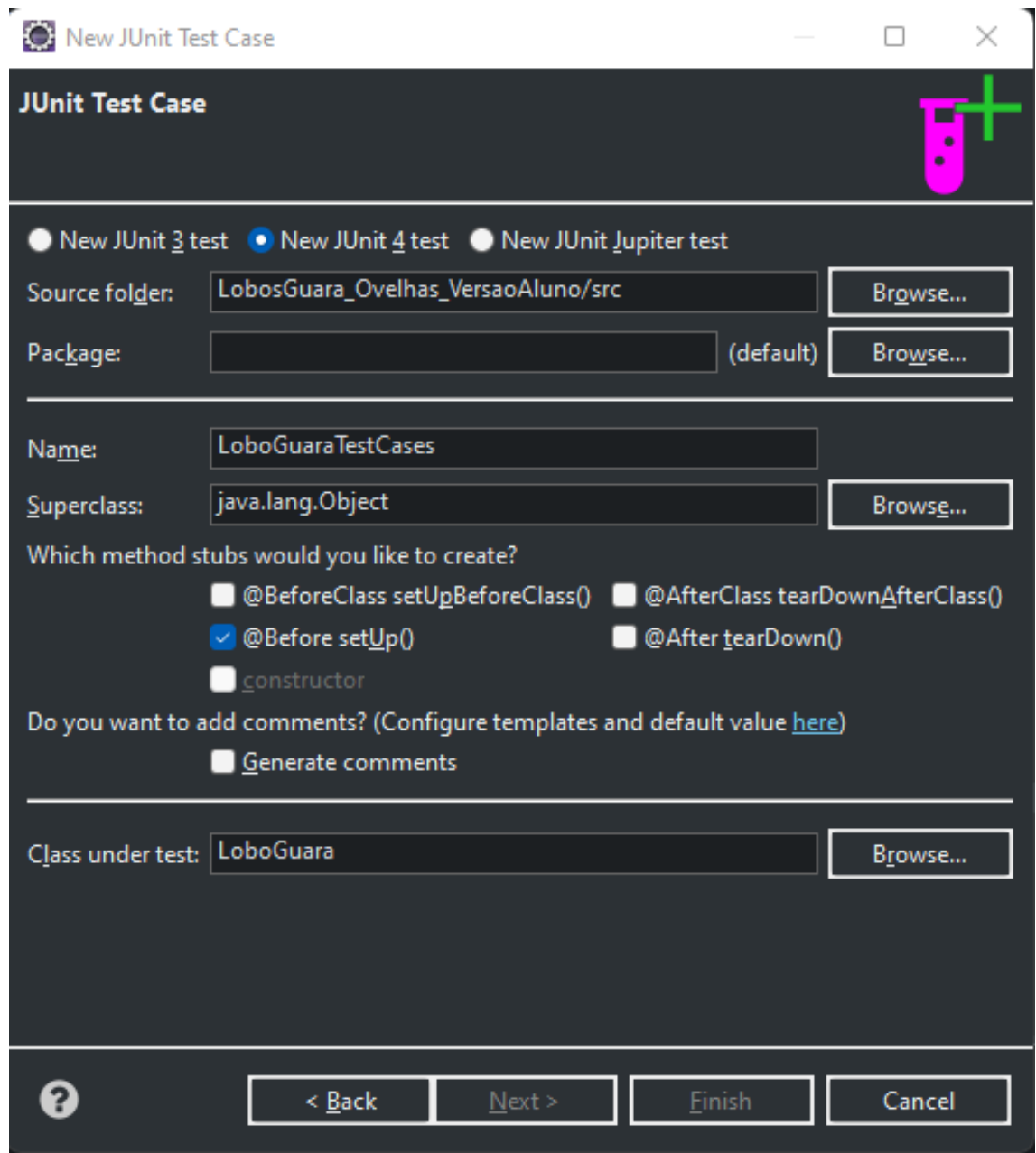
teclado. (Conforme a imagem acima)

O Eclipse irá exibir uma nova Janela, digite o nome "JUnit" e selecione a opção "JUnit Test Case" e clique no botão "Next". (Conforme a imagem abaixo)



Na configuração do JUnit Test Case, siga os seguintes passos:

- 1º Selecione "New JUnit 4 test".
- 2º Adicione o nome do caso de teste que será realizado (Por padrão definido, será: O nome da classe que será testada + TestCases) por exemplo: LoboGuaraTestCases.
- 3º Selecione a opção "@Before setup()".
- 4º No campo "Class under test" selecione a classe que será testada, seguindo o exemplo anterior selecionei a classe "LoboGuara.java".
- 5º Clique no botão "Finish"



New JUnit Test Case

JUnit Test Case

☐ New JUnit 3 test ☒ New JUnit 4 test ☐ New JUnit Jupiter test

Source folder: LobosGuara_Ovelhas_VersaoAluno/src Browse...

Package: (default) Browse...

Name: LoboGuaraTestCases

Superclass: java.lang.Object Browse...

Which method stubs would you like to create?

☐ @BeforeClass setUpBeforeClass() ☐ @AfterClass tearDownAfterClass()
☒ @Before setUp() ☐ @After tearDown()
☐ constructor

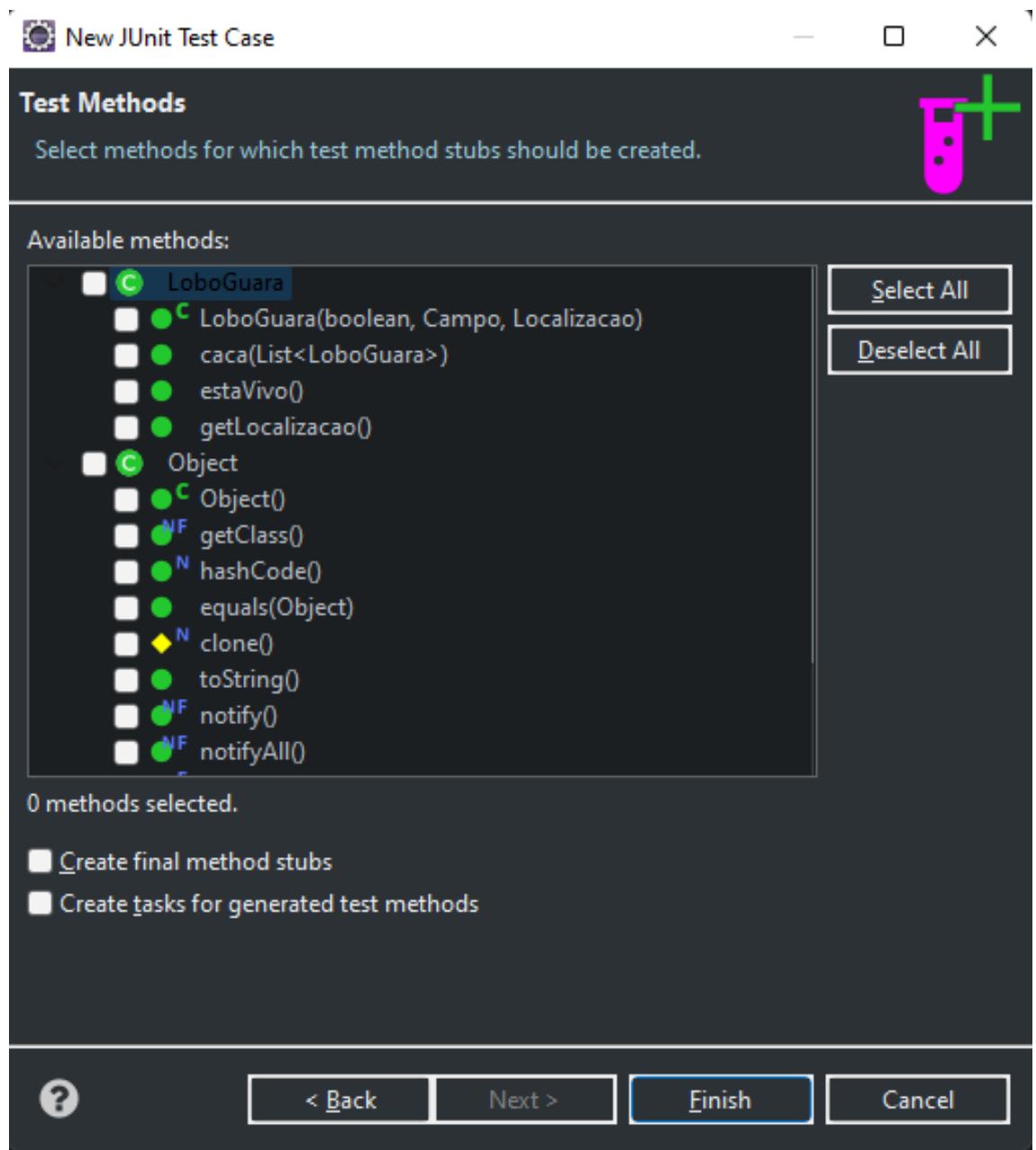
Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Class under test: LoboGuara Browse...

? < Back Next > Finish Cancel

Após seguir os passos acima, clique em "Next" e o Eclipse irá solicitar que sejam adicionadas as classes que serão testadas, selecione as classes que já possuem testes já documentados (Wiki do projeto).



Após selecionadas as classes basta clicar no botão "Finish" que o JUnit ira gerar um novo arquivo .Java com o nome que foi informado e será possível validar os metodos que serão testados, na proxima sessão vamos abordar um pouco sobre as tags.

```

import static org.junit.Assert.*;

public class LoboGuaraTestCases {

    static LoboGuara lobo;
    static Simulador s;

    @Before
    public void setUp() throws Exception {

    }

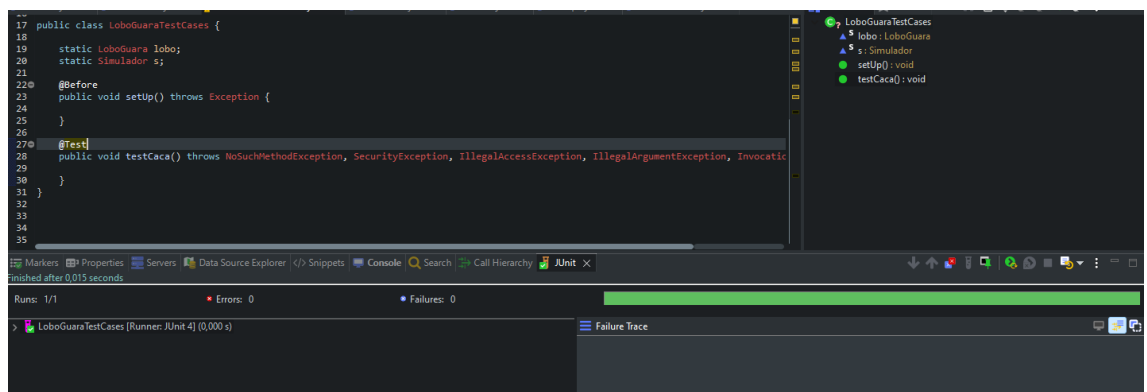
    @Test
    public void testCaca() throws NoSuchMethodException, SecurityException, IllegalAccessException, IllegalArgumentException, InvocationTargetException {

    }

}

```

Exemplo de Execução de Casos de Testes:



4. Tags e informações importantes

O JUnit possui notações específicas para identificar o tipo de teste, repetir o teste e até para adicionar tarefas pré e pós execução, abaixo irei listar algumas das importantes notações que vamos utilizar e também no fim desse artigo o link onde você pode encontrar mais informações.

- **1º @Test:** Essa notação que fica acima do método que foi criado em sua JUnit Test, define que o método abaixo é um caso de teste que será executado.
- **2º @Before:** Essa instrução irá indicar que o método abaixo será executado sempre antes de qualquer teste pré definido, ela é útil para adicionar construtores ou adicionar passos que sempre vão se repetir, para evitar repetição de código e melhorar a manutenibilidade.
- **3º @After:** Assim como a instrução anterior o After também terá uma rotina fixa, que será sempre executada após os testes que foram finalizados, o que é útil para salvar evidências ou até gerar relatórios pós execução de testes.
- **4º @DisplayName:** Essa "TAG" é muito útil, pois ela irá definir o nome do teste que foi executado exibindo essa informação na Switch de Testes que foi executada pelo JUnit isso simplifica a identificação de possíveis erros.

References

- Team, J. (2021a). JUnit 5 user guide. pages <https://junit.org/junit5/docs/current/user-guide/>. JUnit.
- Team, T. P. (2021b). JUnit - quick guide. In Team, J., editor, *JUnit - Overview*, pages <https://junit.org/junit5/docs/current/user-guide/>. Tutorials Point.

Warcholinski, M. (2021). Test-driven development (tdd) – quick guide. In Warcholinski, M., editor, *Test-Driven Development*, pages <https://brainhub.eu/library/test-driven-development-tdd/>. BrainHub.