**JUnit**

JUnit é uma API de código aberto para a criação de testes unitários em Java, bem como outras linguagens compatíveis com a JVM. Um dos seus principais projetistas, o cientista Erich Gamma, conhecido pelo livro Design Patterns: Elements of Reusable Object-Oriented Software, citou como a motivação por trás do JUnit permitir a criação de testes que são fáceis de escrever e executar.

**Escrevendo testes**

Escrever uma classe de teste consiste em utilizar alguns métodos e anotações da API do JUnit, criando assim pequenos cenários nos quais identificamos se o comportamento de uma unidade de trabalho corresponde ao esperado.

#### **@Test**

A primeira anotação a ser utilizada para criar testes unitários é @Test, que informa ao JUnit quais são os métodos de teste de uma classe. Para o JUnit o nome do método não importa, pois desde que ele seja anotado dessa maneira ele será identificado como um método de teste. Um método anotado com @Test não deve retornar um valor.

**@ParameterizedTest e @ValueSource**

Para escrever um teste parametrizado devemos utilizar a anotação @ParameterizedTest em conjunto de @ValueSource.

**@RepeatedTest**

JUnit provê uma forma de executarmos um teste diversas vezes anotando-o com a anotação @RepeatedTest, a qual recebe o número repetições desejado. Cada execução de um método anotado dessa maneira é performada como se estivéssemos utilizando a anotação @Test.

**@TestFactory**

Uma novidade no JUnit 5 são os testes dinâmicos, que nos permitem criar casos de teste em tempo de execução, compostos por um nome e um Executable, dentro de um método de fabricação de testes. Com esse recurso podemos usar os dados em uma coleção como fonte para a criação de diferentes contextos de falha, criando um teste para cada um de seus itens dinamicamente, de forma que eles sejam executados pelo JUnit através de Iterator. Um método criado dessa maneira deve ser anotado com @TestFactory e retornar uma coleção de objetos do tipo DynamicTest, que serão um a um executados pelo JUnit.

**@BeforeEach**

@BeforeEach, o método da classe que é executado antes de cada teste, nos permite inicializar objetos ou realizar qualquer tarefa relacionada a preparação de um ambiente adequado para que os objetos reunidos em um contexto executem as suas ações de forma consistente e isolada.

**@AfterEach**

@AfterEach desfaz esse ambiente, removendo da memória caches, fechando streams ou quaisquer outros dados que poderiam comprometer a confiabilidade do próximo teste.

### Assertions

A classe Assertions é uma coleção de métodos que afirmam se certas condições comuns em testes correspondem ao esperado.

#### **assertTrue** **([message,] boolean condition)**

Requer que a condição booleana seja verdadeira.

#### **assertFalse** **([message,] boolean condition)**

Requer que a condição booleana seja falsa.

#### **assertNull** **([message,] object)**

Requer que o valor informado como parâmetro seja nulo.

#### **assertNotNull** **([message,] object)**

Requer que o valor informado como parâmetro não seja nulo.

#### **assertEquals** **([message,] expected, actual, [tolerance])**

Requer que ambos os valores passados como parâmetro sejam verdadeiros.

#### **assertArrayEquals ([message,] expected, actual)**

Requer que ambos os vetores sejam iguais.

#### **assertIterableEquals ([message,] expected, actual)**

Requer que todos os objetos do tipo Iterable passados como parâmetro sejam iguais.

#### **assertLinesMatch ([message,] expected, actual)**

Requer que ambas as listas de strings sejam iguais.

#### **assertNotEquals ([message,] unexpected, actual)**

Requer que ambos os valores passados como parâmetro sejam iguais.

#### **assertSame** **([message,] expected, actual)**

Requer que ambos os parâmetros se refiram ao mesmo objeto.

#### **assertNotSame** **([message,] expected, actual)**

Requer que ambos os parâmetros se refiram a objetos diferentes.

#### **assertAll ([message,] executables)**

Requer que todos os objetos do tipo Executable passados como parâmetro não lançam exceções.

#### **assertThrows ([message,] expectedType, executable)**

Requer que a execução do objeto do tipo Executable passado como parâmetro lance uma exceção do tipo esperado.

#### **assertTimeout ([message,] timeout, executable)**

Requer que uma execução se complete antes do tempo estimado.

#### **assertTimeoutPreemptively ([message,] timeout, executable)**

Requer que uma execução se complete após do tempo estimado.

**fail([message])**

Pode ser usado para verificar se uma determinada parte do código não é atingida ou para ter um teste com falha antes que o código de teste seja implementado.

**Curiosidades**

* É o framework mais famoso e utilizado para a realização de testes;
* Seu estudo e aperfeiçoamento é um dos principais pilares para o conceito TDD;
* É orientado a objetos;
* Testes que podem ser utilizados ao longo de todo o tempo de vida do projeto;
* Usado fortemente em XP, TDD e outros métodos ágeis.

**Referências**

**JUnit 5 User Guide**. Disponível em: <https://junit.org/junit5/docs/current/user-guide/>.

‌HTTPS://WWW.FACEBOOK.COM/DEVMEDIA.COM.BR. **Teste unitário com JUnit**. Disponível em: <https://www.devmedia.com.br/teste-unitario-com-junit/41235>.

‌LARS VOGEL (C) 2007, 2019 VOGELLA GMBH. **Unit Testing with JUnit - Tutorial**. Disponível em: <https://www.vogella.com/tutorials/JUnit/article.html>.

‌**JUnit**. Disponível em: <https://prezi.com/p/jcenulsaicql/junit/>. Acesso em: 1 nov. 2023.

‌AZEVEDO, M. **Qualidade e JUnit: introduzindo automatização de testes unitários do seu software Java no dia-a-dia**. Disponível em: <https://mari-azevedo.medium.com/qualidade-e-junit-introduzindo-automatiza%C3%A7%C3%A3o-de-testes-unit%C3%A1rios-do-seu-software-java-no-dia-a-dia-849611de5574>.