VEM SER

Módulo 4.1 - Testes unitários

Aula 01 - JUnit

Conteúdo da aula - pt 1

- TDD;
- Os passos do TDD;
- Testes para quê?;
- Pirâmide de testes;
- Casos de teste (*Test cases*);
- Asserções ou assertivas;
- Princípio FIRST;
- Maven;
- XML;
- JUnit.

TDD

TDD - Test-Driven Development

O “desenvolvimento orientado a testes” (em tradução livre) é uma prática defendida pela metodologia ágil eXtreme Programming (XP).

Ela defende que o teste deve existir antes mesmo da funcionalidade e que ele deve orientar o desenvolvimento do código.

Ainda que saibamos que o teste vai falhar (pois a funcionalidade não existe ainda), o propósito é esse mesmo. Escrever um teste que falhe, para que a funcionalidade desenvolvida corrija a falha apresentada no teste e faça ele funcionar.

A vantagem

Todo o código desenvolvido já estará testado.

A filosofia

O TDD se baseia nos testes unitários, que consistem, basicamente, em validar (através do teste) uma menor parte testável do sistema (ou da funcionalidade em questão).

E ainda: caso seja encontrado algum *bug*, em vez de debugar ou corrigir, deve-se antes de tudo **criar um teste** para verificar o comportamento e só depois corrigir o *bug*, de modo que o teste oriente a correção.

Os passos do TDD

Os passos do TDD

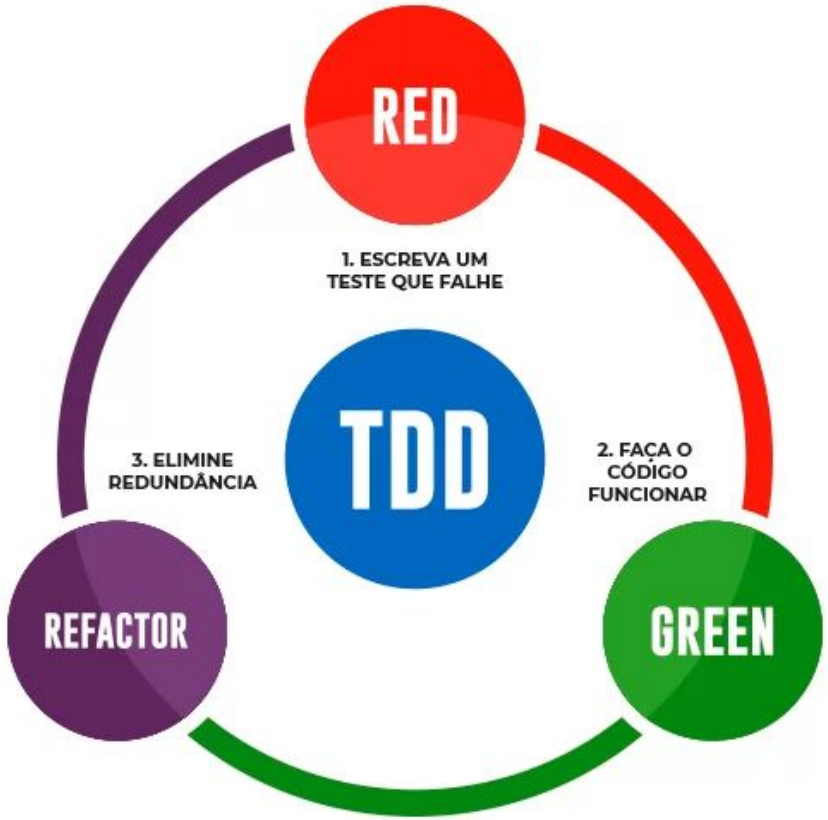


Imagem: https://diegofranca.dev/2021/03/05/tdd-analizando-feedbacks/

Resumindo...

- **RED**: Faça;
- **GREEN**: Faça **certo**;
- **REFACTOR**: Faça **melhor**.

Por quê?

Com isso, eliminamos a otimização prematura e garantimos a qualidade do código.

Outras filosofias

- **DDD** (*Domain-Driven Design*) - filosofia arquitetural que baseou o TDD;
- **BDD** (*Behavior-Driven Development*) - surgiu a partir do TDD.

Testes para quê?

Por que testar?

O software vai ter *bugs* de qualquer modo. Testar não é perda de tempo?

Por que testar?

Esse é um pensamento bastante comum, muitos desenvolvedores pensam que os testes "diminuem" a produtividade da equipe.

Isso é um mito, pois os testes garantem que o código e as funcionalidades que vão para a produção estão validados. Isso também reduz a quantidade de *bugs* e erros em produção - ou seja reduz a quantidade de *bugs* e erros que podem chegar ao usuário final da aplicação.

Caso uma funcionalidade futura seja adicionada, nada irá quebrar no código, pois o código anterior já estava testado e validado.

Testes e qualidade

Por mais que a equipe deixe de focar só em desenvolvimento e passe a focar em **desenvolvimento** e **testes unitários** (uma atividade a mais), a produtividade aumenta.

Como isso é possível?

Testes e qualidade

Um código testável e testado é um código muito mais **durável**, consequentemente, no futuro surgem menos *bugs* e erros e a equipe pode gastar mais tempo desenvolvendo do que corrigindo código antigo.

O impacto dos testes unitários

Estágio	Equipe sem testes	Equipe com testes
Implementação	7 dias	14 dias
Integração	7 dias	2 dias
Testes e correções	12 dias	9 dias
Tempo de lançamento da feature	26 dias	24 dias
Tempo total	**52 dias**	**49 dias**
Bugs encontrados em produção	**71**	**11**

Fonte: A arte dos testes unitários - 2ª edição

Quais os riscos de subir código **não testado** para produção?

Pirâmide de testes

Pirâmide de testes

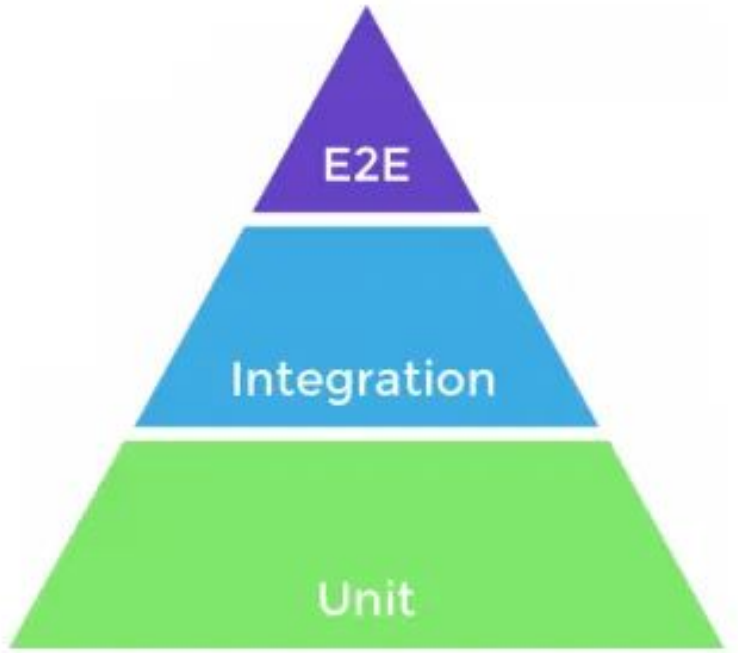


Imagem: https://blog.onedaytesting.com.br/piramide-de-teses/

Pirâmide de testes

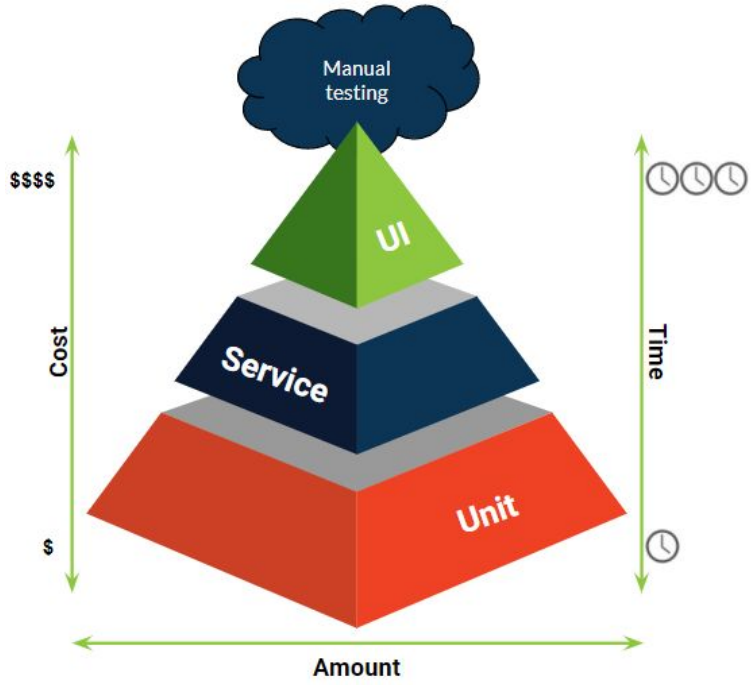


Imagem: https://www.zup.com.br/blog/fases-de-teste-tipos-de-teste

Explicação

- **A base - Testes de unidade ou testes unitários:** são os testes realizados na menor parte testável de uma aplicação ou funcionalidade;

- **O meio - Testes de integração ou testes de serviço:** devem testar as partes da aplicação interagindo entre si;

- **O topo - Testes de ponta a ponta (E2E) ou testes de UI:** têm como objetivo simular um usuário final utilizado a aplicação (testar um fluxo).

Material extra

https://fireworkweb.com.br/testes-de-software-quais-os-tipos-e-por-que-implementar/

https://blog.onedaytesting.com.br/piramide-de-teses/

https://medium.com/creditas-tech/a-pir%C3%A2mide-de-testes-a0faec465cc2

Casos de teste (Test cases)

- Um caso de teste é um conjunto de condições ou dados que são usados para testar um método ou unidade de código;
- Eles são escritos para garantir que o código esteja funcionando conforme o esperado e para identificar possíveis erros;
- Devem cobrir todos os caminhos possíveis pelo código e devem verificar todos os resultados possíveis;
- Também devem ser escritos de forma clara e concisa para que possam ser facilmente entendidos e executados.

Casos de teste (Test cases)

Ao escrever casos de teste, é importante considerar o seguinte:

- A regra do negócio;
- O objetivo do método ou unidade de código que está sendo testado;
- Os diferentes caminhos possíveis pelo código;
- Os diferentes resultados possíveis;
- As condições excepcionais.

Asserções ou assertivas

- Uma asserção ou assertiva é uma declaração que deve ser verdadeira;
- Nos testes unitários, as asserções são usadas para verificar se o código está funcionando corretamente;
- Elas são escritas como um método que recebe dois parâmetros: o valor esperado e o valor real;
- Se os dois valores forem iguais, a asserção é verdadeira; // PASS
- Se os dois valores forem diferentes, a asserção é falsa. // FAIL

Princípio FIRST

- ***F**ast*: Rápidos;
- ***I**ndependent*: Independentes;
- ***R**epeatable*: Repetíveis;
- ***S**elf-validated*: Autovalidáveis;
- ***T**imely / Thorough*: Previamente escritos / Minuciosos.

Vamos para a prática, mas antes...

Maven

Maven é um gerenciador de projetos de código aberto e uma estrutura de construção para projetos Java. Ele ajuda a gerenciar os projetos Java, incluindo a compilação, teste, empacotamento e implantação do código.

Ele também é um **gerenciador de dependências**. Isso pode ajudar a garantir que o projeto em questão esteja usando as versões mais recentes das dependências e pode ajudar a evitar conflitos de dependências.

pom.xml

O arquivo pom.xml é um arquivo de configuração do projeto Maven. Ele contém informações sobre o projeto, como seu nome, versão, dependências e plugins. Ele vai ser usado pelo Maven para construir, testar, empacotar e implantar seu projeto.

pom.xml

O arquivo pom.xml é um arquivo XML e contém as seguintes seções:

- **<project>**: Esta seção contém informações gerais sobre o projeto, como seu nome, versão, nome do grupo e descrição.
- **<modelVersion>**: Esta seção especifica a versão do modelo de projeto Maven que está sendo usado.
- **<groupId>**: Esta seção especifica o nome do grupo do projeto.
- **<artifactId>**: Esta seção especifica o nome do artefato do projeto.
- **<version>**: Esta seção especifica a versão do projeto.
- **<packaging>**: Esta seção especifica o tipo de artefato do projeto, como JAR, WAR ou EAR.
- **<dependencies>**: Esta seção especifica as dependências do projeto.
- **<plugins>**: Esta seção especifica os plugins do projeto.

Artefato?

- Em Maven, um artefato é o produto final de um projeto de construção;
- Um artefato pode ser um JAR, WAR, EAR, etc;
- O nome do artefato é o nome do projeto seguido do tipo de artefato. Por exemplo, o nome do artefato para um projeto chamado "meu-projeto" que está empacotado como um JAR seria "**meu-projeto.jar**".

jar?

- Um arquivo JAR (Java Archive) é um formato de arquivo de arquivo comumente usado para empacotar classes compiladas, recursos e metadados associados em um único arquivo;
- Os arquivos JAR são usados para distribuir aplicativos Java e bibliotecas;
- Resumindo: é um projeto Java, só que empacotado, como o “rar”, por exemplo.

.xml?

- XML significa Extensible Markup Language (Linguagem de Marcação Extensível);
- É uma linguagem de marcação que é usada para representar dados estruturados em um formato legível por humanos e por máquinas;
- Maven usa XML para representar informações sobre projetos Java.

XML vs JSON

XML vs. JSON

```xml
<?xml version="1.0" encoding="UTF-8"?>
<endereco>
    <cep>31270901</cep>
    <city>Belo Horizonte</city>
    <neighborhood>Pampulha</neighborhood>
    <service>correios</service>
    <state>MG</state>
    <street>Av. Presidente Antônio Carlos, 6627</street>
</endereco>
```

```json
{
    "endereco": {
        "cep": "31270901",
        "city": "Belo Horizonte",
        "neighborhood": "Pampulha",
        "service": "correios",
        "state": "MG",
        "street": "Av. Presidente Antônio Carlos, 6627"
    }
}
```

Crédito da imagem: https://pt.m.wikipedia.org/wiki/Ficheiro:JSON_vs._XML.png

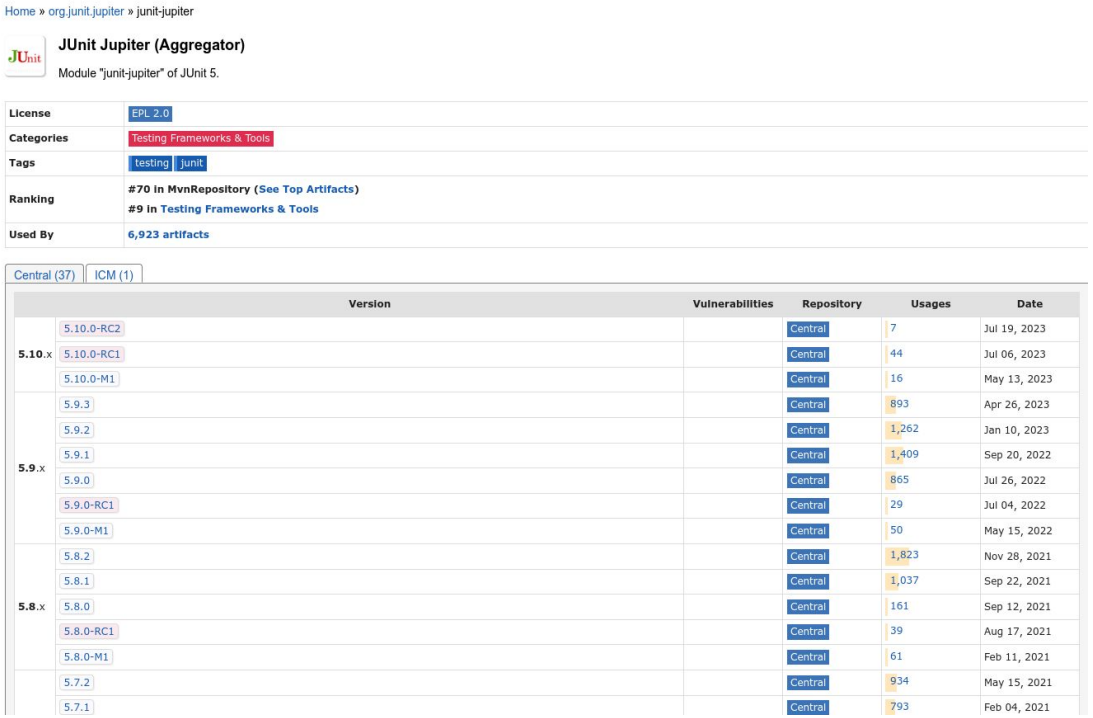
Recomendações de leitura

https://stackify.com/gradle-vs-maven/

JUnit!

JUnit + Maven

https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter



Home » org.junit.jupiter » junit-jupiter

JUnit Jupiter (Aggregator)

Module "junit-jupiter" of JUnit 5.

License	EPL 2.0
Categories	Testing Frameworks & Tools
Tags	testing junit
Ranking	#70 in MvnRepository (See Top Artifacts) #9 in Testing Frameworks & Tools
Used By	6,923 artifacts

Central (37) ICM (1)

	Version	Vulnerabilities	Repository	Usages	Date
5.10.x	5.10.0-RC2		Central	7	Jul 19, 2023
	5.10.0-RC1		Central	44	Jul 06, 2023
	5.10.0-M1		Central	16	May 13, 2023
5.9.x	5.9.3		Central	893	Apr 26, 2023
	5.9.2		Central	1,262	Jan 10, 2023
	5.9.1		Central	1,409	Sep 20, 2022
	5.9.0		Central	865	Jul 26, 2022
	5.9.0-RC1		Central	29	Jul 04, 2022
	5.9.0-M1		Central	50	May 15, 2022
5.8.x	5.8.2		Central	1,823	Nov 28, 2021
	5.8.1		Central	1,037	Sep 22, 2021
	5.8.0		Central	161	Sep 12, 2021
	5.8.0-RC1		Central	39	Aug 17, 2021
	5.8.0-M1		Central	61	Feb 11, 2021
	5.7.2		Central	934	May 15, 2021
	5.7.1		Central	793	Feb 04, 2021

Adicionando no pom.xml

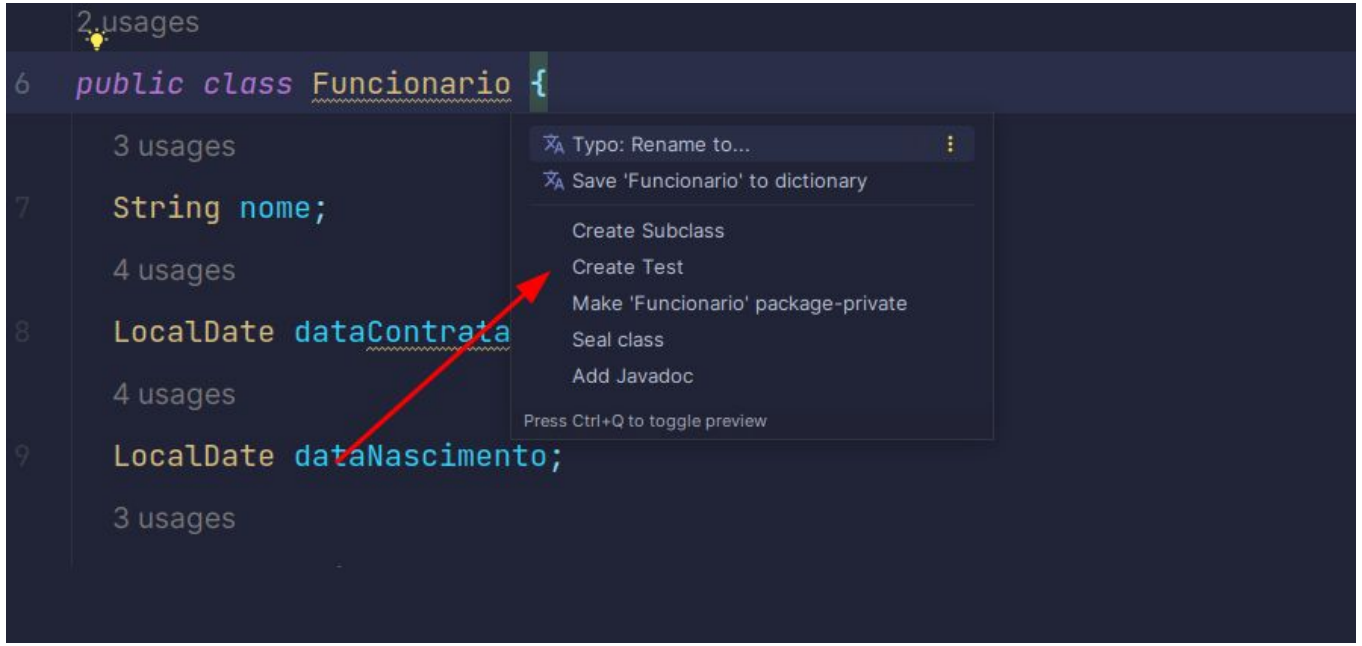
```
<dependencies>
  <dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter</artifactId>
  <version>5.9.2</version>
  <scope>test</scope>
  </dependency>
</dependencies>
```

Adicionando no pom.xml

```
<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>3.1.2</version>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
```

Criando arquivo de teste

Vá na classe que deseja testar, pressione **Alt + Enter** e selecione “Create Test”.



Observação!

Lembrando que os arquivos de testes devem seguir o padrão:
NomeDaClasseTest.java

Exemplo:
Nome da classe: Pessoa
Arquivo de teste: PessoaTest

Anotação @Test

```
@Test
void nomeDoTeste(){
}
```

O que deve ser testado?

- Regras de negócio (services);
- Classes principais;
- Códigos com lógica e complexidade.

O que não deve ser testado (nos testes unitários)?

- Conexão com banco de dados;
- Envio de e-mail;
- Conexão com API;
- Get/Set;
- Controllers;
- etc.

Task individual

- Criar pasta "**modulo-04-1**" (ou "**modulo-04-1-junit**") no seu Git e copiar os arquivos do último projeto conta-corrente3 do **módulo de Java e renomeiem o projeto para conta-corrente4**;
- **Não esquecer de selecionar Maven na parte do Build System;**

- <**groupId**>br.com.dbccompany</**groupId**>
- <**artifactId**>conta-corrente4</**artifactId**>

- Criar uma classe ContaTest.java (dentro da pasta **test**) no seu projeto e desenvolver os casos de testes do próximo slide.

Casos de teste

- **deveTestarSaqueContaCorrenteEVerificarSaldoComSucesso**
- **deveTestarSaqueContaCorrenteSemSaldo**: não dar certo o valor do saque (saque > saldo + ce)
- **deveTestarSaqueContaPoupancaEVerificarSaldoComSucesso**: deve creditar taxa antes
- **deveTestarSaqueContaPoupancaSemSaldo**: não dar certo o valor do saque (saque > saldo)
- **deveTestarSaqueContaPagamentoEVerificarSaldoComSucesso**
- **deveTestarSaqueContaPagamentoSemSaldo**: não dar certo o valor do saque (saque > saldo)
- **deveTestarTransferenciaEVerificarSaldoComSucesso**
- **deveTestarTransferenciaSemSaldo**: não dar certo o valor do saque (saque > saldo)
- **deveTestarDepositoEVerificarSaldoComSucesso**
- **deveTestarDepositoNegativo**

Referências

https://junit.org/junit5/docs/current/user-guide/
https://medium.com/@qaschool/conhe%C3%A7a-os-princ%C3%ADpios-f-i-r-s-t-aplicados-em-testes-de-unidade-999c9d2a00d

Referências

https://junit.org/junit5/docs/current/user-guide/
https://medium.com/@qaschool/conhe%C3%A7a-os-princ%C3%ADpios-f-i-r-s-t-aplicados-em-testes-de-unidade-999c9d2a00d


Let's Tech Up Together