

# T2 - Teste unitário

**Grupo:** Gustavo Lopes Rodrigues, Henrique Moraes Cota, Rafael Amauri Diniz Augusto

[Clique aqui para acessar o repositório do projeto](#)

## Classe Venda

### Funções

```
public interface VendaInterface {  
    public NotaFiscal gerarNotaFiscal(ArrayList<String> codigosDeBarra);  
    public NotaFiscal aplicarDesconto(NotaFiscal nf, double desconto);  
    public Item identificarItem(String codigoDeBarra);  
}
```

### GerarNotaFiscal

A primeira função é a **gerarNotaFiscal**, que irá receber uma lista dinâmica (ArrayList) com códigos de barra dos produtos e de comandas compradas dentro da padaria. A função então irá identificar os produtos presentes na lista, e irá adicioná-los na nota fiscal. O resultado da função é a nota fiscal com as compras realizadas pelo cliente.

```
public NotaFiscal gerarNotaFiscal(ArrayList<String> codigos_barra) {  
    NotaFiscal totalizacao = new NotaFiscal();  
  
    for (String s : codigos_barra) {  
        Item i = identificarItem(s);  
        totalizacao.adicionarItem(i);  
    }  
  
    return totalizacao;  
}
```

### aplicarDesconto

A segunda função **aplicarDesconto**, recebe uma NotaFiscal e um desconto (em formato double). O resultado dessa função é a Nota Fiscal, com o desconto aplicado na mesma.

```
public NotaFiscal aplicarDesconto(NotaFiscal nf, double desconto) {
    nf.setDesconto(desconto);
    return nf;
}
```

## identificarItem

A terceira função **identificarItem**, recebe uma String que representa um código de barra de um item da padaria. Essa função então irá acessar o banco de dados da classe venda, que é nada mais do que uma estrutura de Mapa , que contém a relação código de barra e item. Ou seja, para cada código de barra, temos um item que o corresponde.

```
private static final HashMap<String, Item> banco_de_dados_items = new HashMap<>();

static {
    banco_de_dados_items.put("14501926", new Item("14501926", "Pão Integral", 15.77));
    banco_de_dados_items.put("39547612", new Item("39547612", "Cerveja", 7.75));
    banco_de_dados_items.put("16437122", new Item("16437122", "Doce", 4.18));
    banco_de_dados_items.put("28629351", new Item("28629351", "Bolo", 16.06));
    banco_de_dados_items.put("74972239", new Item("74972239", "Leite Integral", 18.44));
    banco_de_dados_items.put("49914660", new Item("49914660", "Bala", 1.27));
    banco_de_dados_items.put("59802701", new Item("59802701", "Vinho", 10.32));
    banco_de_dados_items.put("05938213", new Item("05938213", "Requeijão", 5.84));
}
```

O resultado da função **identificarItem**, será o item correspondente ao código de barra inserido.

```
public Item identificarItem(String codigo_barra) {
    return banco_de_dados_items.get(codigo_barra);
}
```

# Classe VendaTest

Para verificar que a classe Venda estava funcionando como deveria. Temos a classe que faz integração com o JUnit, para realizar testes em cada uma das funções.

## gerarNotaFiscalTest

Primeiro temos o test da função **gerarNotaFiscal**, onde o temos como entrada de dados, uma instância da classe Venda e uma lista com o código de barra com dois produtos, dois sacos de pão integral e uma lata de cerveja. O resultado do teste irá verificar se a totalização ( preço total ) da nota fiscal está correta, assim como o número total de itens.

```
@Test
public void gerarNotaFiscalTest() {

    Venda padaria = new Venda();

    ArrayList<String> codigos_barra = new ArrayList<String>();
    codigos_barra.add("14501926");
    codigos_barra.add("14501926");
    codigos_barra.add("39547612");

    NotaFiscal nota_fiscal = padaria.gerarNotaFiscal(codigos_barra);

    Assert.assertEquals(39.29, nota_fiscal.getTotalizacao(),0);
    Assert.assertEquals(3,nota_fiscal.getNumeroItems());
}
```

## aplicarDescontoTest

Neste teste iremos gerar uma nota fiscal e então tentar inserir um desconto nela, e verificar que o preço total foi diminuído devido ao desconto

```
@Test
public void aplicarDescontoTest() {

    Venda padaria = new Venda();

    ArrayList<String> codigos_barra = new ArrayList<String>();
    codigos_barra.add("14501926");
    codigos_barra.add("14501926");
    codigos_barra.add("39547612");

    NotaFiscal nota_fiscal = padaria.gerarNotaFiscal(codigos_barra);

    padaria.aplicarDesconto(nota_fiscal, 15);

    Assert.assertEquals(15.0, nota_fiscal.getDesconto(),0);
    Assert.assertEquals(33.3965, nota_fiscal.getPrecoTotal(),0);
}
```

## identificarItemTest

Depois temos o test da função **identificarItem**, neste caso, a entrada de dados será três códigos de barra, sendo um deles inválido. O teste irá verificar que dois retornaram o item com sucesso e o outro retornou um objeto nulo.

```
@Test
public void identificarItemTest() {

    Venda padaria = new Venda();

    Item pao = padaria.identificarItem("14501926");
    Assert.assertEquals("Pão Integral", pao.getName());

    Item bala = padaria.identificarItem("49914660");
    Assert.assertEquals("Bala", bala.getName());

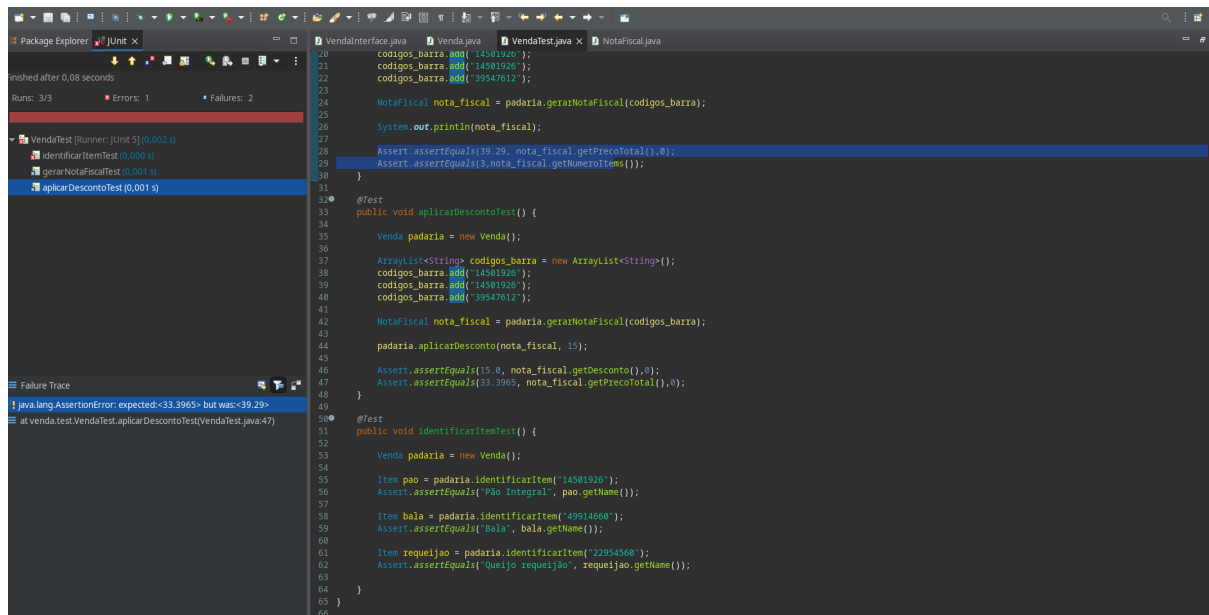
    Item requeijao = padaria.identificarItem("22954560");
    Assert.assertEquals("Queijo requeijão", requeijao.getName());

}
```

## Tabela de teste

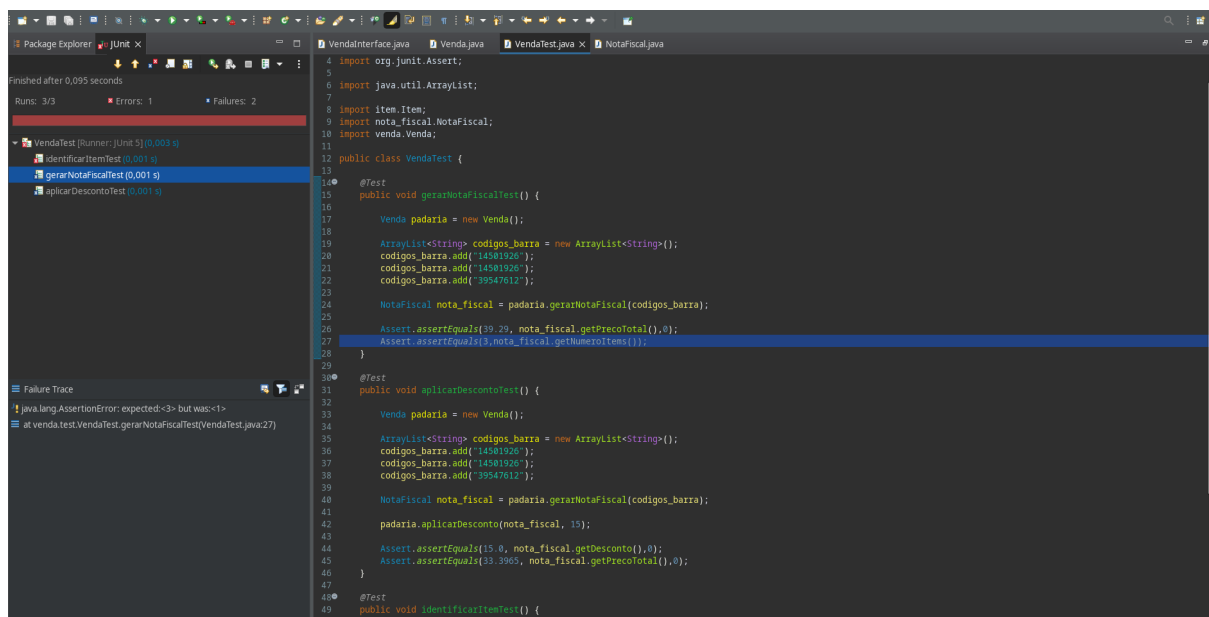
Dados	RE	RO	Avaliação
2x Pão Integral + 1x Cerveja	39,29	39,29	Sucesso
2x Pão Integral + 1x Cerveja	3 Itens	1 Item	Falha
2x Pão Integral + 1x Cerveja	15% de desconto	15% de desconto	Sucesso
2x Pão Integral + 1x Cerveja	33.3965	39,29	Falha
Código de barra: 14501926	Pão Integral	Pão Integral	Sucesso
Código de barra: 49914660	Bala	Bala	Sucesso
Código de barra: 22954560	Queijo requeijão	Null	Erro (Null Pointer Exception)

# Prints do Junit



The screenshot shows an IDE with the following components:

- Package Explorer:** Shows the project structure with a folder named 'JUnit'. Below it, a list of test results for 'VendaTest' is shown, including 'VendaTest [Runner: JUnit 5] (0.002 s)', 'identificarItemTest (0.000 s)', 'gerarNotaFiscalTest (0.001 s)', and 'aplicarDescontoTest (0.001 s)'. The 'aplicarDescontoTest' is highlighted in blue.
- Failure Trace:** Displays the error message: 'java.lang.AssertionError: expected<33.3965> but was<39.29>' at 'venda.test.VendaTest.aplicarDescontoTest(VendaTest.java:47)'. This is also highlighted in blue.
- Source Code:** The 'VendaTest.java' file is open, showing the implementation of the tests. The code includes imports for 'org.junit.Assert', 'java.util.ArrayList', 'item.Item', 'nota\_fiscal.NotaFiscal', and 'venda.Venda'. The 'aplicarDescontoTest' method is highlighted in blue, showing the application of a 15% discount to a 'NotaFiscal' object and assertions for the resulting price and item count.



The screenshot shows an IDE with the following components:

- Package Explorer:** Shows the project structure with a folder named 'JUnit'. Below it, a list of test results for 'VendaTest' is shown, including 'VendaTest [Runner: JUnit 5] (0.003 s)', 'identificarItemTest (0.001 s)', 'gerarNotaFiscalTest (0.001 s)', and 'aplicarDescontoTest (0.001 s)'. The 'gerarNotaFiscalTest' is highlighted in blue.
- Failure Trace:** Displays the error message: 'java.lang.AssertionError: expected<3> but was<1>' at 'venda.test.VendaTest.gerarNotaFiscalTest(VendaTest.java:27)'. This is also highlighted in blue.
- Source Code:** The 'VendaTest.java' file is open, showing the implementation of the tests. The code includes imports for 'org.junit.Assert', 'java.util.ArrayList', 'item.Item', 'nota\_fiscal.NotaFiscal', and 'venda.Venda'. The 'gerarNotaFiscalTest' method is highlighted in blue, showing the generation of a 'NotaFiscal' object from a list of bar codes and assertions for the resulting price and item count.

