

PRÁTICA LABORATORIAL JUNIT5

Objetivos:

- Conceitos de Software Quality Assurance
- Testes em Java
- Gradle
- Junit5

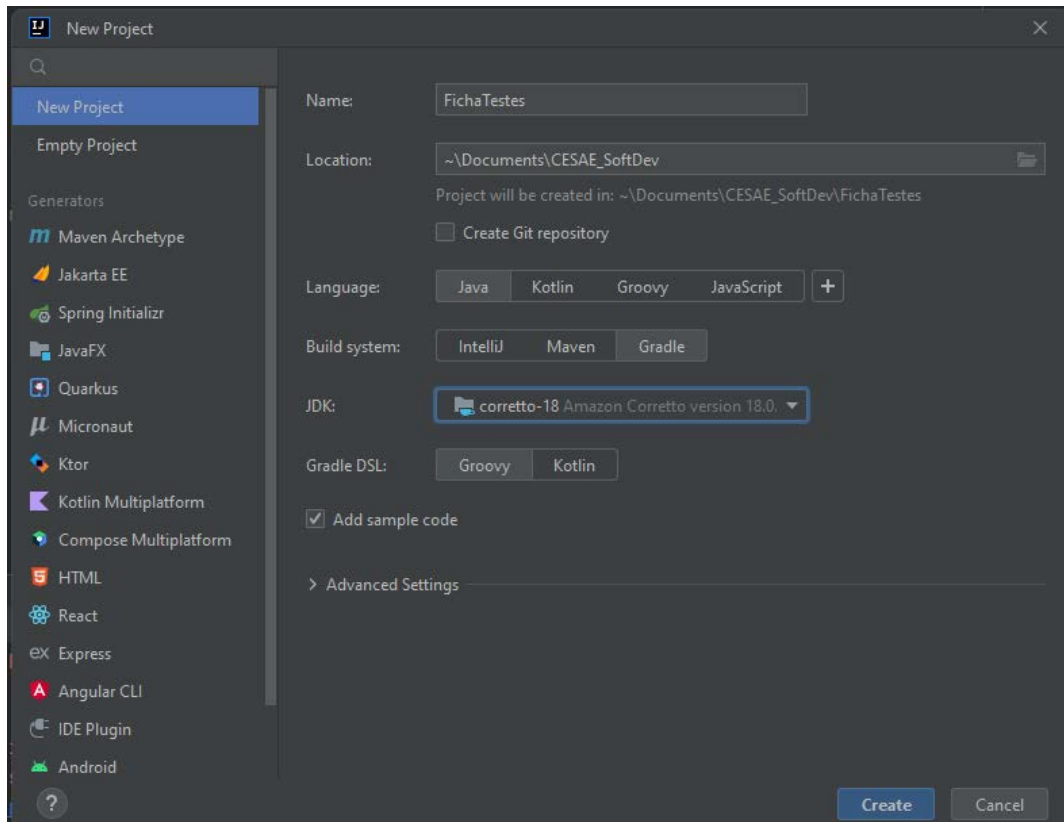
EXERCÍCIOS

Parte 1

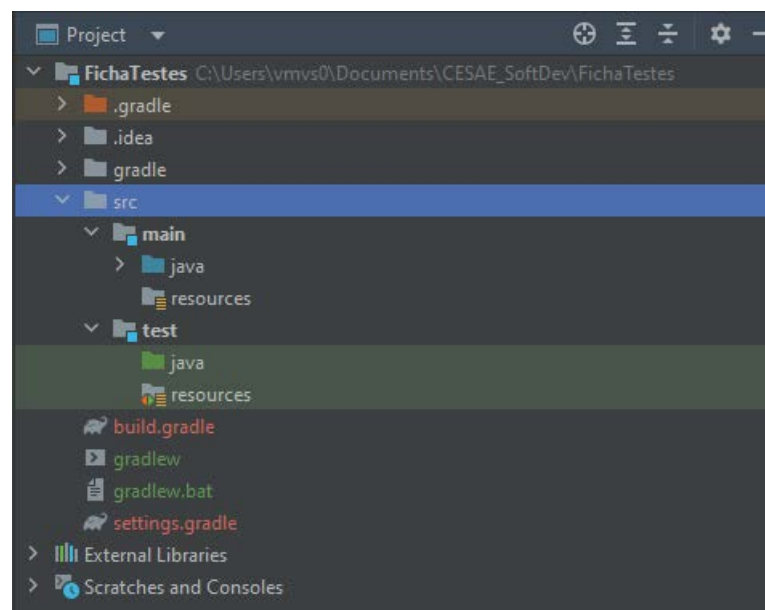
Tenha em consideração o seguinte excerto de código que representa uma calculadora básica em Java. Contém funções de soma, subtração, multiplicação e divisão (atente que a divisão nunca pode ser por 0, daí o tratamento de exceções). Deve ser criada uma classe de testes para cada uma destas funções.

```
public class Calculator {  
    public int add(int a, int b) {  
        return a + b;  
    }  
  
    public int subtract(int a, int b) {  
        return a - b;  
    }  
  
    public int multiply(int a, int b) {  
        return a * b;  
    }  
  
    public int divide(int a, int b) {  
        if (b == 0) {  
            throw new IllegalArgumentException("Impossível dividir por 0");  
        }  
        return a / b;  
    }  
}
```

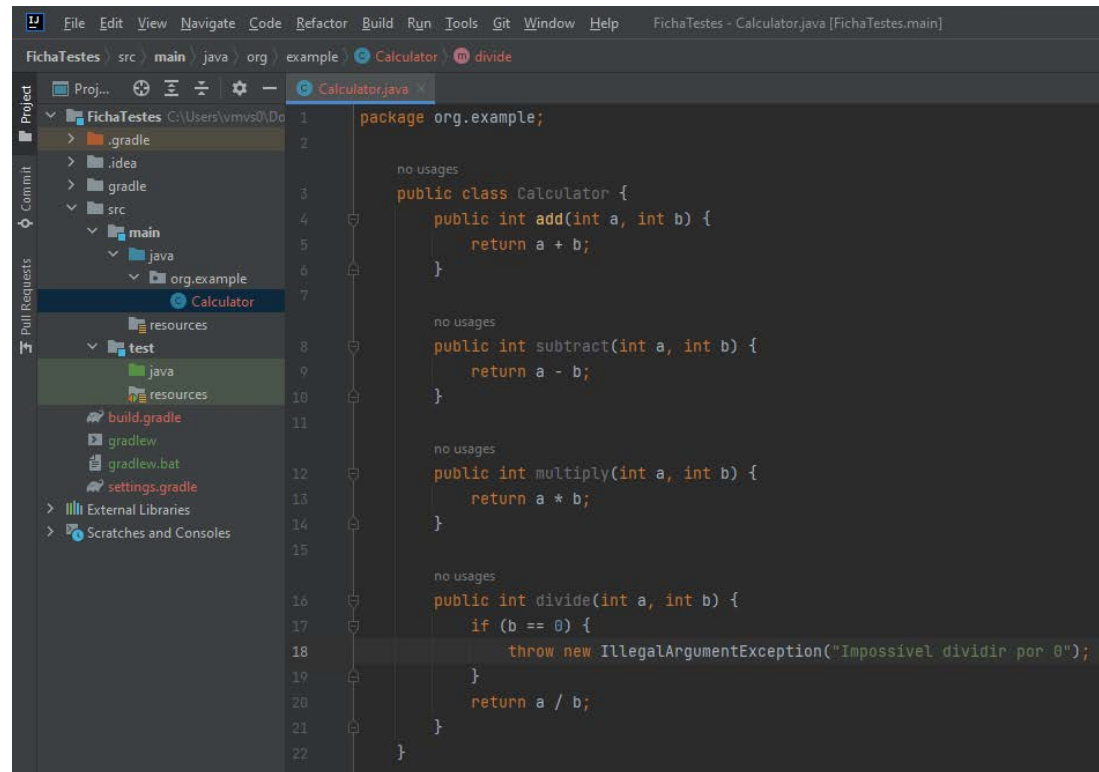
Assim, crie o projeto em IntelliJ tendo como Build System o Gradle (ao contrário de IntelliJ) como estamos habituados.



A estrutura do projeto será, por sua vez, também diferente do habitual, teremos na pasta src dois módulos (main e test) representado o código do programa e o código de testes.



Para tal, devemos criar uma classe Calculator no módulo main (módulo que tem o programa, deve ter os packages, interfaces, classes abstratas, classes e etc... como estamos habituados).



The screenshot shows an IDE with the project 'FichaTestes' open. The left sidebar shows the project structure with the 'main' module selected. The main editor displays the 'Calculator.java' file with the following code:

```
package org.example;

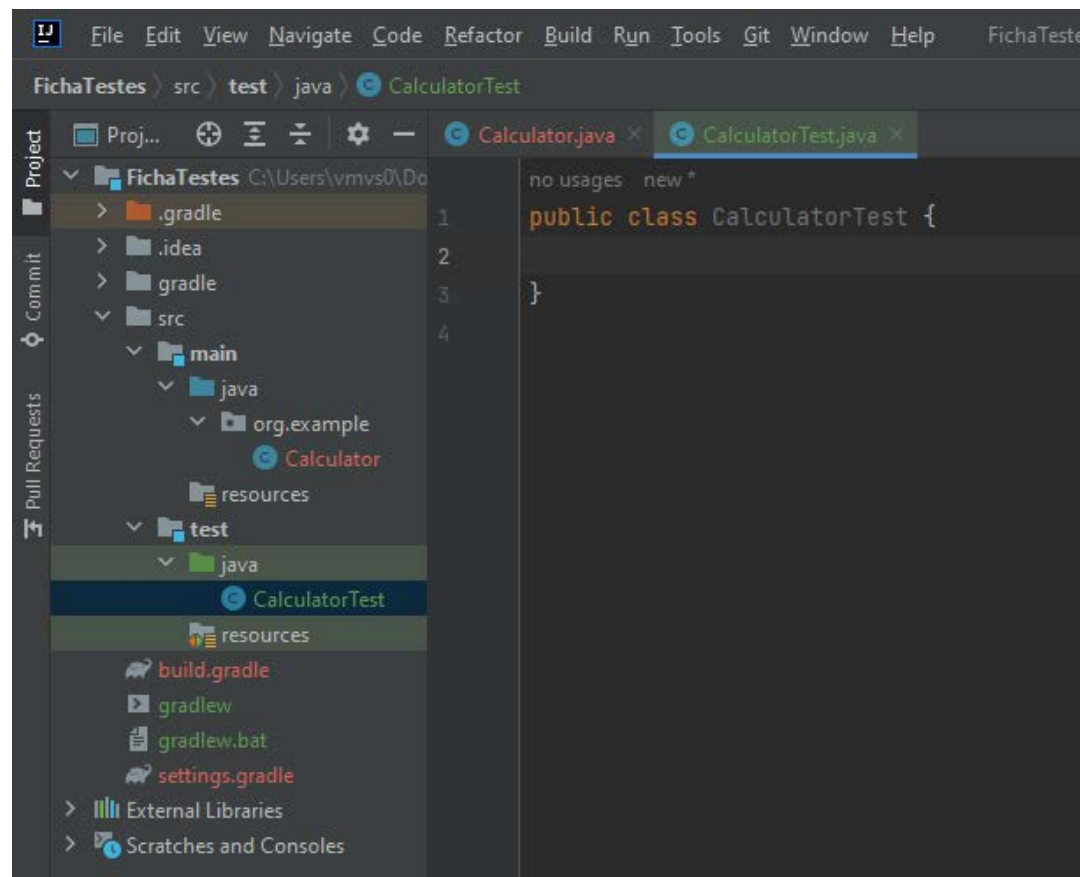
no usages
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    no usages
    public int subtract(int a, int b) {
        return a - b;
    }

    no usages
    public int multiply(int a, int b) {
        return a * b;
    }

    no usages
    public int divide(int a, int b) {
        if (b == 0) {
            throw new IllegalArgumentException("Impossível dividir por 0");
        }
        return a / b;
    }
}
```

Seguidamente, deve ser criada uma classe de testes já no módulo test:



The screenshot shows the same IDE with the project 'FichaTestes' open. The left sidebar shows the project structure with the 'test' module selected. The main editor displays the 'CalculatorTest.java' file with the following code:

```
no usages new *
public class CalculatorTest {
}

no usages
```

O seguinte código representa um exemplo de como devem ser executados os testes às funções da calculadora, terá como exemplo da soma e divisão.

```
import org.example.Calculator;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class CalculatorTest {
    private Calculator calculator;

    @BeforeEach
    public void setUp() {
        calculator = new Calculator();
    }

    @Test
    public void testAdd() {
        int result = calculator.add(2, 3);
        assertEquals(5, result);
    }

    @Test
    public void testDivide() {
        int result = calculator.divide(10, 2);
        assertEquals(5, result);
    }

    @Test
    public void testDivideByZero() {
        assertThrows(IllegalArgumentException.class, () -> {
            calculator.divide(10, 0);
        });
    }
}
```

Os imports serão obrigatórios para termos acesso a funções de testes, devem ser importadas também as classes a testar. Seguidamente devemos ter uma instância da classe que queremos testar. Posteriormente, o `@BeforeEach`, ou seja, o código que será executado automaticamente antes de cada teste, geralmente trata-se de instanciar objetos das classes envolvidas nos testes e/ou declarar variáveis a usar.

A seguir temos os testes unitários, onde testamos individualmente cada função da classe, neste caso em específico será o teste da função da soma e da divisão.

1. Elabore as funções de teste para as funções de subtração e multiplicação da classe Calculator.
 - a. Elabore um main onde tenha um exemplo de operação de divisão como:

```
public static void main(String[] args) {  
  
    Scanner input = new Scanner(System.in);  
    Calculator calculator = new Calculator();  
    int num1, num2;  
  
    System.out.print("Primeiro número: ");  
    num1= input.nextInt();  
  
    System.out.print("Segundo número: ");  
    num2= input.nextInt();  
  
    System.out.println("Divisão: "+ calculator.divide(num1, num2));  
}
```

Assim, percebemos que o facto de o projeto estar construído com Gradle, permite executar uma série de tarefas antes de cada build/run, nomeadamente testes unitários...

- b. Altere a função add e faça de modo que reprove os testes, por exemplo, em vez de return a+b; pode retornar sempre 0 ou return a+a;
 - c. Execute o programa e perceba que, embora o nosso main não invoque a função add, que os seus testes serão sempre executados antes de cada build/run!

2. Atente o seguinte código que contém uma função que retorna se uma String é um palíndromo ou não, a mesma função tira partido da função reverseString que recebe uma String e reverte a mesma.

```
public class StringUtils {  
    public static boolean isPalindrome(String str) {  
        if (str == null) {  
            return false;  
        }  
  
        String reversed = reverseString(str);  
        return str.equalsIgnoreCase(reversed);  
    }  
  
    private static String reverseString(String str) {  
        StringBuilder sb = new StringBuilder();  
        for (int i = str.length() - 1; i >= 0; i--) {  
            sb.append(str.charAt(i));  
        }  
        return sb.toString();  
    }  
}
```

Elabore a classe de testes para a classe StringUtils que contém estes dois métodos. Deverá prever todos os cenários e testar todos os casos, seguidamente verifique se passam em todos os testes.

Exemplos de casos específicos: maiúsculas/minúsculas, null, etc...

Caso reprove algum teste, altere os métodos de forma que passem a 100% dos testes.

3. Considere a seguinte classe `Animal` que tem `nome`, `comFome` e `tipoAlimentacao`. Seguidamente teremos o método `comer` que recebe um tipo de alimento como argumento e, de seguida, verifica se o animal come ou não dito alimento.

```
public enum Alimento {  
    CARNE,  
    PEIXE,  
    FRUTAS,  
    VEGETAIS  
}  
  
public class Animal {  
    private String nome;  
    private boolean comFome;  
    private Alimento tipoAlimentacao;  
  
    public Animal(String nome, Alimento tipoAlimentacao) {  
        this.nome = nome;  
        this.comFome = true;  
        this.tipoAlimentacao = tipoAlimentacao;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public boolean estaComFome() {  
        return comFome;  
    }  
  
    public Alimento getTipoAlimentacao() {  
        return tipoAlimentacao;  
    }  
  
    public void comer(Alimento comida) {  
        if (tipoAlimentacao == comida) {  
            comFome = false;  
            System.out.println(nome + " comeu " + comida + ".");  
        } else {  
            System.out.println(nome + " não come " + comida + ".");  
        }  
    }  
}
```

Elabore as classes de testes necessárias e altere os métodos caso entenda que não passem nos testes parametrizados.

-
4. Considere o exercício resolvida na Ficha Prática 9 Ex.3 onde temos a classe Retângulo que pode calcular perímetro e área.... Disponível no repositório: https://github.com/Vmvs007/CESAE/tree/main/Ficha_09/src . Elabore a classe de testes para testar todos os métodos da classe Retângulo.

 5. Considere o exercício resolvida na Ficha Prática 9 Ex.10 onde temos a classe Funcionário que pode exibir detalhes e aumentar o salário... Disponível no repositório: https://github.com/Vmvs007/CESAE/tree/main/Ficha_09/src . Elabore a classe de testes para testar todos os métodos da classe Funcionário.

 6. Considere o exercício resolvida na Ficha Prática 10 Ex.1 onde temos a classe Carro que pode ligar, fazer corridas, calcular consumos e etc... Disponível no repositório: https://github.com/Vmvs007/CESAE/tree/main/Ficha_10/src . Elabore a classe de testes para testar todos os métodos da classe Carro.

 7. Considere o exercício resolvida na Ficha Prática 10 Ex.2 onde temos a classe Conta que pode depositar, levantar, transferir e etc... Disponível no repositório: https://github.com/Vmvs007/CESAE/tree/main/Ficha_10/src . Elabore a classe de testes para testar todos os métodos da classe Conta.

8. Atente o seguinte código que contém uma classe ShoppingCart que guardará vários produtos associados a um código de cliente:

```
package org.example;

import java.util.ArrayList;
import java.util.List;

public class ShoppingCart {
    private ArrayList<String> items;
    private String codCliente;

    public ShoppingCart(String codCliente) {
        this.codCliente=codCliente;
        this.items = new ArrayList<>();
    }

    public void addItem(String item) {
        items.add(item);
    }

    public void removeItem(String item) {
        items.remove(item);
    }

    public boolean containsItem(String item) {
        return items.contains(item);
    }

    public int getItemCount() {
        return items.size();
    }

    public void clearCart() {
        items.clear();
    }
}
```

Elabore a classe de testes para ShoppingCart.

9. Considere o seguinte código que representa um programa que dá suporte a uma loja de videojogos, cada jogo tem nome, editora, preço de custo e preço de venda. A classe GameAlert atua como classe de loja e armazena num array list denominado stock um conjunto de videojogos inicializados com nome, editora, preço de custo e preço de venda. Noutro arraylist denominado vendas, armazena jogos que saem diretamente do arraylist de stock e inicializando o preço de venda. Tem o método para adicionar jogos ao stock e para vender um jogo.

```
import java.util.ArrayList;
import java.util.List;

class VideoGame {
    private String name;
    private String publisher;
    private double costPrice;
    private double sellingPrice;

    public VideoGame(String name, String publisher, double costPrice, double sellingPrice) {
        this.name = name;
        this.publisher = publisher;
        this.costPrice = costPrice;
        this.sellingPrice = sellingPrice;
    }

    public String getName() {
        return name;
    }

    public String getPublisher() {
        return publisher;
    }

    public double getCostPrice() {
        return costPrice;
    }

    public double getSellingPrice() {
        return sellingPrice;
    }
}

class GameStore {
    private ArrayList<VideoGame> stock;
    private ArrayList<VideoGame> sales;

    public GameStore() {
        stock = new ArrayList<>();
        sales = new ArrayList<>();
    }

    public void addGameToStock(VideoGame game) {
        stock.add(game);
    }
}
```

```
public void sellGame(int index) {
    VideoGame game = stock.remove(index);
    sales.add(new VideoGame(game.getName(), game.getPublisher(), game.getCostPrice(),
game.getSellingPrice()));
}

public void displayStock() {
    System.out.println("Stock:");
    for (VideoGame game : stock) {
        System.out.println("Name: " + game.getName());
        System.out.println("Publisher: " + game.getPublisher());
        System.out.println("Cost Price: " + game.getCostPrice());
        System.out.println("Selling Price: " + game.getSellingPrice());
        System.out.println("-----");
    }
}

public void displaySales() {
    System.out.println("Sales:");
    for (VideoGame game : sales) {
        System.out.println("Name: " + game.getName());
        System.out.println("Publisher: " + game.getPublisher());
        System.out.println("Cost Price: " + game.getCostPrice());
        System.out.println("Selling Price: " + game.getSellingPrice());
        System.out.println("-----");
    }
}

}

public class Main {
    public static void main(String[] args) {
        GameStore gameStore = new GameStore();

        // Adicionar jogos ao stock
        gameStore.addGameToStock(new VideoGame("Skyrim", "Bethesda", 10.0, 50.0));
        gameStore.addGameToStock(new VideoGame("GTA V", "Rockstar", 15.0, 70.0));
        gameStore.addGameToStock(new VideoGame("The Legend of Zelda: Breath of the Wild", "Nintendo", 30.0,
60.0));
        gameStore.addGameToStock(new VideoGame("Red Dead Redemption 2", "Rockstar Games", 40.0, 50.0));
        gameStore.addGameToStock(new VideoGame("FIFA 22", "Electronic Arts", 20.0, 45.0));
        gameStore.addGameToStock(new VideoGame("Assassin's Creed Valhalla", "Ubisoft", 35.0, 55.0));
        gameStore.addGameToStock(new VideoGame("Marvel's Spider-Man: Miles Morales", "Insomniac Games",
25.0, 40.0));
        gameStore.addGameToStock(new VideoGame("Grand Theft Auto V", "Rockstar Games", 25.0, 40.0));
        gameStore.addGameToStock(new VideoGame("Minecraft", "Mojang Studios", 15.0, 30.0));
        gameStore.addGameToStock(new VideoGame("The Witcher 3: Wild Hunt", "CD Projekt Red", 20.0, 35.0));
        gameStore.addGameToStock(new VideoGame("Call of Duty: Modern Warfare", "Activision", 30.0, 50.0));
        gameStore.addGameToStock(new VideoGame("Super Mario Odyssey", "Nintendo", 40.0, 60.0));
        gameStore.addGameToStock(new VideoGame("The Last of Us Part II", "Naughty Dog", 40.0, 60.0));
        gameStore.addGameToStock(new VideoGame("Final Fantasy VII Remake", "Square Enix", 30.0, 50.0));
        gameStore.addGameToStock(new VideoGame("Resident Evil Village", "Capcom", 35.0, 55.0));
        gameStore.addGameToStock(new VideoGame("Super Smash Bros. Ultimate", "Nintendo", 25.0, 40.0));
        gameStore.addGameToStock(new VideoGame("Ghost of Tsushima", "Sucker Punch Productions", 30.0, 45.0));
        gameStore.addGameToStock(new VideoGame("Horizon Zero Dawn", "Guerrilla Games", 20.0, 35.0));
        gameStore.addGameToStock(new VideoGame("Dark Souls III", "FromSoftware", 25.0, 40.0));
        gameStore.addGameToStock(new VideoGame("God of War", "Santa Monica Studio", 30.0, 50.0));
    }
}
```

```
gameStore.addGameToStock(new VideoGame("The Elder Scrolls V: Skyrim", "Bethesda Game Studios", 15.0, 30.0));
gameStore.addGameToStock(new VideoGame("Persona 5", "Atlus", 30.0, 50.0));
gameStore.addGameToStock(new VideoGame("Mario Kart 8 Deluxe", "Nintendo", 40.0, 60.0));
gameStore.addGameToStock(new VideoGame("Animal Crossing: New Horizons", "Nintendo", 35.0, 55.0));
gameStore.addGameToStock(new VideoGame("Cyberpunk 2077", "CD Projekt", 25.0, 40.0));
gameStore.addGameToStock(new VideoGame("Uncharted 4: A Thief's End", "Naughty Dog", 20.0, 35.0));
gameStore.addGameToStock(new VideoGame("Cyberpunk 2077", "CD Projekt", 25.0, 40.0));
gameStore.addGameToStock(new VideoGame("Uncharted 4: A Thief's End", "Naughty Dog", 20.0, 35.0));
gameStore.addGameToStock(new VideoGame("Hades", "Supergiant Games", 15.0, 30.0));

// Venda de jogos
gameStore.sellGame(0);
gameStore.sellGame(1);

// Exibir stock e vendas
gameStore.displayStock();
gameStore.displaySales();
}
```

- Desenvolva a classe de testes para cada uma das funções das classes acima explicitadas.
- Desenvolva o método `calcularLucro()` que soma o total de vendas e subtrai o total de custos e retorna o total.
- Desenvolva o teste necessário para este novo método.

Bom trabalho! 😊