1. Qual é o principal objetivo dos testes unitários?

Resposta: c) Testar unidades isoladas de código para garantir seu funcionamento correto.

2. O que é TDD (Test-Driven Development)?

Resposta: c) Uma abordagem que utiliza testes automatizados para direcionar o processo de desenvolvimento.

3. Qual é a principal vantagem de escrever testes antes de implementar o código (TDD)?

Resposta: c) Melhoria na qualidade do código.

4. O que é um "teste unitário" no contexto do desenvolvimento de software?

Resposta: c) Um teste que verifica o comportamento de um módulo ou função isoladamente.

5. Quais são as características de um bom teste unitário?

Resposta: c) Deve ser repetível e automatizado.

6. Qual é o principal propósito dos testes unitários dentro de um projeto de desenvolvimento de software?

Resposta: c) Assegurar que cada parte do código funcione conforme o esperado.

7. Quais são os possíveis benefícios de adotar TDD (Test-Driven Development) em um projeto de software?

Resposta: c) Melhoria na qualidade do código.

8. Qual é a sequência de passos típica no ciclo do TDD (Test-Driven Development)?

Resposta: b) Escrever testes, implementar o código, fazer refatorações.

9. Qual é o principal objetivo de refatorar o código durante o processo de TDD?

Resposta: c) Melhorar a estrutura e a organização do código.

10. Qual é a diferença entre testes unitários e testes de integração?

Resposta: a) Testes unitários verificam unidades isoladas de código, enquanto testes de integração verificam a interação entre diferentes partes do sistema.

Implementação

1)

public class CalculadoraIMC {

public static double calcularIMC(double peso, double altura) {

if (altura <= 0) {

throw new IllegalArgumentException("Altura deve ser maior que zero.");

}

return peso / (altura \* altura);

}

}

-----------------------------------------

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculadoraIMCTest {

@Test

public void testCalcularIMC() {

double peso = 68.0;

double altura = 1.75;

double imcCalculado = CalculadoraIMC.calcularIMC(peso, altura);

double imcEsperado = 22.2;

assertEquals(imcEsperado, imcCalculado, 0.1);

}

@Test

public void testCalcularIMCAlturaZero() {

double peso = 68.0;

double altura = 0.0;

IllegalArgumentException exception = assertThrows(IllegalArgumentException.class, () -> {

CalculadoraIMC.calcularIMC(peso, altura);

});

// Verifica se a mensagem de exceção está correta

assertEquals("Altura deve ser maior que zero.", exception.getMessage());

}

}

-------------------------------------------

2)

public class ItemVendido {

private String nomeProduto;

private double precoUnitario;

private double desconto;

public ItemVendido(String nomeProduto, double precoUnitario, double desconto) {

this.nomeProduto = nomeProduto;

this.precoUnitario = precoUnitario;

this.desconto = desconto;

}

public boolean verificarVenda() {

if (desconto > 0.03) {

return false; // Desconto não pode exceder 3%

}

if (precoUnitario == 0) {

return false; // Preço unitário não pode ser zero

}

return true;

}

}

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertFalse;

import static org.junit.jupiter.api.Assertions.assertTrue;

public class ItemVendidoTest {

@Test

public void testVerificarVendaDescontoExcedido() {

ItemVendido item = new ItemVendido("Produto A", 10.0, 0.04);

assertFalse(item.verificarVenda()); // Desconto excede 3%

}

@Test

public void testVerificarVendaPrecoZero() {

ItemVendido item = new ItemVendido("Produto B", 0.0, 0.02);

assertFalse(item.verificarVenda()); // Preço unitário é zero

}

@Test

public void testVerificarVendaCorreta() {

ItemVendido item = new ItemVendido("Produto C", 20.0, 0.02);

assertTrue(item.verificarVenda()); // Venda correta, desconto e preço unitário aceitáveis

}

}