Cadastro de Produtos - TDD

Funcionalidade: Cadastro de Produtos

Membro do Grupo: Giulia Alcantara

Github: alcantaragiubs

1. Cadastro de Produtos - Teste

```
Produto produto;

@Before
public void setup(){
    produto = new Produto();
}
```

- A classe de teste é declarada como `CadastroProdutoTest`.

- Uma instância de `Produto` é criada antes de cada teste no método `setup()` usando a anotação `@Before`.

```
@Test
public void testeCadastroProdutoCorreto(){
    String nome = "Produto A";
    String codigoBarras = "123456789";
    double precoCompra = 10.0;
    double precoVenda = 20.0;
    int quantidadeEstoque = 50;
    double delta = 0.001;
    Produto produtoCadastrado = produto.cadastroProduto(nome, codigoBarras, precoCompra, precoVenda, quantidadeEstoque);
    assertEquals(nome, produtoCadastrado.getNome());
    assertEquals(codigoBarras, produtoCadastrado.getCodigoBarras());
    assertEquals(precoCompra, produtoCadastrado.getPrecoCompra(), delta);
    assertEquals(precoVenda, produtoCadastrado.getPrecoVenda(), delta);
    assertEquals(quantidadeEstoque, produtoCadastrado.getQuantidadeEstoque());
}
```

- O método `testeCadastroProdutoCorreto` realiza o teste de cadastro de um produto com valores específicos.

- Usa o método `assertEquals` do JUnit para verificar se os valores retornados pelo método `cadastroProduto` são iguais aos valores esperados.

```java
public class CadastroProdutoTest {

    Produto produto;

    @Before
    public void setup(){
        produto = new Produto();
    }
    @Test
    public void testeCadastroProdutoCorreto(){
        String nome = "Produto A";
        String codigoBarras = "123456789";
        double precoCompra = 10.0;
        double precoVenda = 20.0;
        int quantidadeEstoque = 50;
        double delta = 0.001;
        Produto produtoCadastrado = produto.cadastroProduto(nome, codigoBarras, precoCompra, precoVenda, quantidadeEstoque);
        assertEquals(nome, produtoCadastrado.getNome());
        assertEquals(codigoBarras, produtoCadastrado.getCodigoBarras());
        assertEquals(precoCompra, produtoCadastrado.getPrecoCompra(), delta);
        assertEquals(precoVenda, produtoCadastrado.getPrecoVenda(), delta);
        assertEquals(quantidadeEstoque, produtoCadastrado.getQuantidadeEstoque());
    }
}
```

Este teste verifica se o método `cadastroProduto` da classe `Produto` está funcionando corretamente, garantindo que os valores retornados correspondam aos valores esperados.

2. Cadastro de Produtos Exceção - Teste

```java
@Category(TestesExcecoes.class)

public class CadastroProdutoExcecao {

    Produto produto;

    @Before
    public void setup(){
        produto = new Produto();
    }
```

- A classe de teste é declarada como `CadastroProdutoExcecao`.

- Uma instância de `Produto` é criada antes de cada teste no método `setup()` usando a anotação `@Before`.

- A anotação `@Category(TestesExcecoes.class)` indica que esta classe de teste pertence à categoria `TestesExcecoes`.

```java
@Test
public void testeDescricaoEmBranco(){
    DescricaoEmBrancoException exception = assertThrows(expectedThrowable:DescricaoEmBrancoException.class, () -> {
        produto.cadastroProduto(nome:"", codigoBarras:"123456789", precoCompra:0, precoVenda:20.0, quantidadeEstoque:50);
    });
    assertEquals(expected:"Os campos não pode estar em branco", exception.getMessage());
}

@Test
public void testeProdutoInvalido(){
    ValorInvalidoException exception = assertThrows(expectedThrowable:ValorInvalidoException.class, () -> {
        produto.cadastroProduto(nome:"Produto A", codigoBarras:"123456789", precoCompra:0, precoVenda:20.0, quantidadeEstoque:50);
    });
    assertEquals(expected:"Valores de compra, venda e quantidade devem ser maiores que zero", exception.getMessage());
}
```

- `testeDescricaoEmBranco`: Testa se uma `DescricaoEmBrancoException` é lançada quando o campo de descrição está em branco. O método `assertThrows` verifica se a exceção esperada é lançada e compara a mensagem da exceção com a mensagem esperada.

- `testeProdutoInvalido`: Testa se uma `ValorInvalidoException` é lançada quando valores de compra, venda ou quantidade são iguais a zero. O método `assertThrows` é usado da mesma forma que no teste anterior.

```java
public class CadastroProdutoExcecao {

    Produto produto;

    @Before
    public void setup(){
        produto = new Produto();
    }

    @Test
    public void testeDescricaoEmBranco(){
        DescricaoEmBrancoException exception = assertThrows(expectedThrowable:DescricaoEmBrancoException.class, () -> {
            produto.cadastroProduto(nome:"", codigoBarras:"123456789", precoCompra:0, precoVenda:20.0, quantidadeEstoque:50);
        });
        assertEquals(expected:"Os campos não pode estar em branco", exception.getMessage());
    }

    @Test
    public void testeProdutoInvalido(){
        ValorInvalidoException exception = assertThrows(expectedThrowable:ValorInvalidoException.class, () -> {
            produto.cadastroProduto(nome:"Produto A", codigoBarras:"123456789", precoCompra:0, precoVenda:20.0, quantidadeEstoque:50);
        });
        assertEquals(expected:"Valores de compra, venda e quantidade devem ser maiores que zero", exception.getMessage());
    }

}
```

Estes testes são úteis para garantir que as exceções corretas sejam lançadas quando valores inválidos são passados para o método `cadastroProduto` da classe `Produto`.

```java
public class DescricaoEmBrancoException extends RuntimeException {

    public DescricaoEmBrancoException() {
        super();
    }

    public DescricaoEmBrancoException(String msg) {
        super(msg);
    }

}
```

```java
public class ValorInvalidoException extends RuntimeException {

    public ValorInvalidoException() {
        super();
    }

    public ValorInvalidoException(String msg) {
        super(msg);
    }

}
```

É certificado também de que as exceções `DescricaoEmBrancoException` e `ValorInvalidoException` estejam implementadas corretamente na sua aplicação para que esses testes funcionem conforme esperado.

3. Atualização da classe produto

```java
public String getNome() {
    return nome;
}
public void setNome(String nome){
    this.nome = nome;
}

public String getCodigoBarras() {
    return codigoBarras;
}
public void setCodigoBarras(String codigoBarras){
    this.codigoBarras = codigoBarras;
}

public double getPrecoCompra() {
    return precoCompra;
}
public void setPrecoCompra(double precoCompra){
    this.precoCompra = precoCompra;
}

public double getPrecoVenda() {
    return precoVenda;
}
public void setPrecoVenda(double precoVenda){
    this.precoVenda = precoVenda;
}

public int getQuantidadeEstoque() {
    return quantidadeEstoque;
}
public void setQuantidadeEstoque(int quantidadeEstoque){
    this.quantidadeEstoque = quantidadeEstoque;
}
```

- São utilizados métodos getters e setters para obter e definir os valores dos atributos da classe referente ao cadastro de produtos

```
public Produto cadastroProduto(String nome, String codigoBarras, double precoCompra, double precoVenda, int quantidadeEstoque)
        throws DescricaoEmBrancoException, ValorInvalidoException {
    Produto produto = new Produto();
    validarEntradas(nome, codigoBarras, precoCompra, precoVenda, quantidadeEstoque);
    produto.setNome(nome);
    produto.setCodigoBarras(codigoBarras);
    produto.setPrecoCompra(precoCompra);
    produto.setPrecoVenda(precoVenda);
    produto.setQuantidadeEstoque(quantidadeEstoque);

    System.out.println("Produto cadastrado com sucesso!");
    return produto;
}
```

- Recebe parâmetros como nome, código de barras, preço de compra, preço de venda e quantidade em estoque.

- Chama o método privado validarEntradas para verificar se as entradas são válidas.

- Se as entradas são válidas, cria uma nova instância de Produto, define os atributos e retorna o produto cadastrado.

- Se as entradas não são válidas, lança exceções (DescricaoEmBrancoException ou ValorInvalidoException).

```
private void validarEntradas(String nome, String codigoBarras, Double precoCompra, Double precoVenda, Integer quantidadeEstoque)
        throws DescricaoEmBrancoException, ValorInvalidoException {
    if (nome == null || codigoBarras == null || nome.isEmpty() || codigoBarras.isEmpty() ||
     precoCompra == null || precoVenda == null || quantidadeEstoque == null){
        throw new DescricaoEmBrancoException(msg:"Os campos não pode estar em branco");
    }
    if (precoCompra <= 0 || precoVenda <= 0 || quantidadeEstoque <= 0){
        throw new ValorInvalidoException(msg:"Valores de compra, venda e quantidade devem ser maiores que zero");
    }
}
```

- Recebe os mesmos parâmetros do método cadastroProduto.

- Lança exceções DescricaoEmBrancoException se algum dos campos obrigatórios estiver em branco ou nulo.

- Lança exceção ValorInvalidoException se o preço de compra, preço de venda ou quantidade em estoque for menor ou igual a zero.