UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Marayah Sabelle Carvalho Meneses - 493486
Thiago Lins da Silva - 494270
Elias Dias de Araújo - 497445
Igor Gabriel da Silva Castro - 496458
Francisco Ulisses Alves de Lima - 496549

Verificação e Validação

Trabalho de TDD + Mockito

Projeto: Lista de Compras

Profª Paulyne Matthews Juca

Quixadá

2024

Introdução
Este documento detalha a documentação das principais classes, funções e testes presentes no projeto Lista de Compras.

Classes Principais

Classe CadastroProduto
Pacote: CadastroProduto

Descrição: Esta interface é responsável por definir métodos relacionados ao cadastro de produtos no sistema. Ele gerencia a adição, remoção e consulta de produtos na lista de compras.
Métodos:

- **cadastrarProduto(Produto p)**: Adiciona um novo produto à lista de compras. Lança *ElementoRepetidoException* e *ArgumentoInvalidoException*.
- **removerProduto(String id):** Remove um produto com base no ID. Lança *ArgumentoInvalidoException*.
- **exists(String id)**: Verifica se um produto com o ID fornecido existe.
- **getPreco(String id)**: Retorna o preço do produto. Lança *ArgumentoInvalidoException*.
- **getQuantidade(String id)**: Este método retorna um valor inteiro (int) que representa a quantidade de itens relacionados ao id fornecido. Por exemplo, se o id corresponde a um item em estoque, o método pode retornar a quantidade desse item disponível no estoque. Lança *ArgumentoInvalidoException* se o id fornecido for inválido
- **getPeso(String id)**: Este método retorna um valor de ponto flutuante (float) que representa o peso de um item identificado pelo id. O peso é provavelmente em uma unidade padrão como quilogramas ou gramas, dependendo do contexto da aplicação. Lança *ArgumentoInvalidoException* se o id fornecido for inválido.
- **float getMl(String id)**: Este método retorna um valor de ponto flutuante (float) que representa o volume em mililitros (ml) de um item identificado pelo id. Este método é útil para itens que têm um volume mensurável, como líquidos. Lança *ArgumentoInvalidoException* se o id fornecido for inválido.
- **getNome(String id)**: Este método retorna uma String que representa o nome do item associado ao id fornecido. Isso pode ser usado para obter a descrição ou o nome de exibição do item. Lança *ArgumentoInvalidoException* se o id fornecido for inválido.

Classe EditaProduto:
Pacote: EditaProduto

Descrição: Esta interface lida com a modificação da quantidade dos produtos na lista de compras, permitindo aumentar ou diminuir a quantidade de um produto existente.
Métodos:

- **addQuantidade(String id, int quantidade, ArrayList<Produto> produtos)**: Adiciona uma quantidade ao produto. Lança *ArgumentoInvalidoException*.

- **rmQuantidade(String id, int quantidade, ArrayList<Produto> produtos)**: Remove uma quantidade do produto. Lança *ArgumentoInvalidoException*.

Classe Produto
Pacote: Produto

Descrição: Classe abstrata que representa um produto genérico, servindo como base para diferentes tipos de produtos, como bebidas e alimentos.
Métodos:
- **addQuantidade(int quantidade)**: Adiciona uma quantidade ao produto.
- **rmQuantidade(int quantidade)**: Remove uma quantidade do produto.

Classe ProdutoBebida
Pacote: Produto.ProdutoBebida

Descrição: Extensão da classe Produto, representa produtos do tipo bebida.
Métodos:
- **getType()**: Retorna o tipo do produto ('Bebida').
- **getMl()**: Retorna o volume da bebida em mililitros.
- **getPeso()**: Retorna 0, já que bebidas não possuem peso.

Classe ProdutoComida
Pacote: Produto.ProdutoComida

Descrição: Extensão da classe Produto, representa produtos do tipo comida.
Métodos:
- **getType()**: Retorna o tipo do produto ('Comida').
- **getPeso()**: Retorna o peso do produto.
- **getMl()**: Retorna 0, já que comidas não possuem volume.

Classe Utils
Pacote: Utils

Descrição: A interface Utils no pacote Utils fornece um conjunto de métodos utilitários que operam em listas de objetos do tipo Produto. A interface Utils utiliza também uma exceção customizada chamada *ArgumentoInvalidoException*, importada do pacote Exceptions.

Métodos:
- **searchProduto(String pattern, ArrayList<Produto> produtos)**: Este método recebe uma string (pattern) e uma lista de produtos (produtos) e retorna uma lista de produtos que correspondem ao padrão fornecido. Por exemplo, ele pode buscar produtos pelo

nome ou descrição que contenha o padrão de string especificado. Lança *ArgumentoInvalidoException* se o padrão ou a lista de produtos for inválida.
- **valorTotal(ArrayList<Produto> produtos)**: Calcula o valor total de todos os produtos na lista fornecida.
- **limparLista(ArrayList<Produto> produtos)**: Limpa a lista de produtos fornecida, removendo todos os elementos.
- **exists(String id, ArrayList<Produto> produtos)**: Verifica se existe um produto com o id especificado na lista de produtos.
- **show(ArrayList<Produto> produtos)**: Exibe e imprime detalhes dos produtos na lista fornecida.
- **comparaTipo(Produto produto)**: Compara o tipo do produto fornecido.
- **toString(ArrayList<Produto> produtos)**: Converte a lista de produtos em uma representação de string.

Casos de Uso e Aplicação dos Testes

Os testes foram projetados para garantir que as funcionalidades cruciais sejam validadas e que possíveis exceções e erros sejam tratados corretamente. A seguir, apresentamos os principais casos de uso testados, explicando como cada um foi implementado e validado.

Cadastro de Produtos

Os testes de cadastro de produtos têm como objetivo garantir que os produtos sejam corretamente inseridos na lista de compras, e que produtos duplicados ou com dados inválidos sejam rejeitados. Para isso, o framework Mockito é utilizado para simular comportamentos e garantir que o processo de validação funcione conforme o esperado.

Por exemplo, ao tentar cadastrar um produto duplicado, o teste simula a tentativa de inserção do mesmo produto duas vezes. A validação deve detectar a duplicidade e lançar uma exceção do tipo *ElementoRepetidoException*. O comportamento correto é verificado por meio da captura de exceções e verificação de interações.

Edição de Produtos

Os testes de edição de produtos verificam se a quantidade de itens em um produto pode ser aumentada ou diminuída corretamente. Neste caso, mocks são utilizados para simular o comportamento da lista de produtos e garantir que os métodos de edição, como addQuantidade() e rmQuantidade(), funcionam conforme o esperado. Além disso, o uso de ArgumentCaptor permite capturar os valores passados para esses métodos e verificar se eles correspondem ao esperado.

Cenários testados incluem a adição de uma quantidade válida a um produto e a tentativa de remoção de uma quantidade superior à disponível, o que deve lançar uma exceção de argumento inválido (*ArgumentoInvalidoException*).

Utilitários

A classe Utils facilita a manipulação e o gerenciamento de listas de produtos, fornecendo métodos para operações comuns que envolvem pesquisa, cálculo de valores, exibição e formatação de dados. Ela oferece métodos utilitários para manipular listas de produtos, como searchProduto() para buscar produtos que correspondem a um padrão de string, valorTotal() para calcular o valor total dos produtos na lista, e limparLista() para esvaziar a lista de produtos. O método exists() verifica se um produto com um determinado ID existe na lista, enquanto show exibe detalhes dos produtos no console. Além disso, métodos como comparaTipo() e toString() fornecem representações de string de tipos de produtos e da lista completa, respectivamente.

Validação e Exceções

O sistema possui validação rigorosa para garantir a integridade dos dados inseridos. Os testes de validação verificam se produtos com parâmetros inválidos, como quantidades negativas ou preços fora do esperado, são corretamente rejeitados. Nesses casos, são lançadas exceções específicas como *ArgumentoInvalidoException*.

A simulação com Mockito ajuda a isolar o comportamento de validação, permitindo que erros sejam simulados e testados de forma controlada. Isso garante que as regras de negócio sejam respeitadas e que o sistema mantenha a consistência dos dados.

Cenários testados

- **Cadastrar Produto comida**
    1. Cadastrar Produto de comida com sucesso
    2. Cadastrar Produto de comida com campos faltando
    3. Cadastrar Produto de comida com nome sem ser tipo de texto
    4. Cadastrar Produto de comida com preço sem ser tipo numérico
    5. Cadastrar Produto de comida com quantidade sem ser tipo inteiro
    6. Cadastrar Produto de comida com peso sem ser tipo numérico
- **Cadastrar Produto bebida**
    1. Cadastrar Produto de bebida com sucesso
    2. Cadastrar Produto de bebida com campos faltando
    3. Cadastrar Produto de bebida com nome sem ser tipo de texto
    4. Cadastrar Produto de bebida com preço sem ser tipo numérico
    5. Cadastrar Produto de bebida com quantidade sem ser tipo inteiro
    6. Cadastrar Produto de bebida com peso sem ser tipo numérico
- **Remover Produto**
    1. Remover Produto com sucesso
    2. Remover Produto com nome vazio
    3. Remover Produto com parâmetro de nome do tipo sem ser texto
    4. Remover Produto com nome que não existe na lista
- **Adicionar Quantidade a um Produto**
    1. Adicionar Quantidade a um Produto com sucesso
    2. Adicionar Quantidade a um Produto com parâmetros vazios

3. Adicionar Quantidade a um Produto com parâmetro de nome do tipo sem ser texto
4. Adicionar Quantidade a um Produto com parâmetro de quantidade do tipo sem ser inteiro
5. Adicionar Quantidade a um Produto com nome que não existe na lista
6. Adicionar Quantidade a um Produto com quantidade negativa
7. Adicionar Quantidade a um Produto com quantidade extremamente grande

- **Remover Quantidade de um Produto**
  1. Remover Quantidade de um Produto com sucesso
  2. Remover Quantidade de um Produto com parâmetros vazios
  3. Remover Quantidade de um Produto com parâmetro de nome do tipo sem ser texto
  4. Remover Quantidade de um Produto com parâmetro de quantidade do tipo sem ser inteiro
  5. Remover Quantidade de um Produto com nome que não existe na lista
  6. Remover Quantidade de um Produto com quantidade negativa
  7. Remover Quantidade de um Produto com quantidade maior do que a disponível
- **Visualizar o Valor Total dos Produtos**
  1. Visualizar o Valor Total dos Produtos com sucesso
  2. Visualizar o Valor Total dos Produtos com parâmetros inexistentes
  3. Visualizar o Valor Total dos Produtos com lista vazia
- **Buscar Produtos pela chave (filtro)**
  1. Buscar Produtos pela chave com sucesso
  2. Buscar Produtos pela chave com chave vazia
  3. Buscar Produtos pela chave com parâmetro de chave do tipo sem ser texto
  4. Buscar Produtos pela chave que não existe na lista
  5. Buscar Produtos pela chave com lista vazia
- **Limpar Lista de Compras**
  1. Limpar Lista de Compras com sucesso
  2. Limpar Lista de Compras com parâmetros inexistentes
  3. Limpar Lista de Compras com lista vazia
- **Visualizar Lista de Compras**
  1. Visualizar Lista de Compras com sucesso
  2. Visualizar Lista de Compras com parâmetros inexistentes
  3. Visualizar Lista de Compras com lista vazia
- **Encerrar o Programa**
  1. Encerrar o Programa com sucesso

Classes de Teste

Classe CadastroProdutoTest

Pacote: CadastroProduto

- **testCadastrarProdutoComidaComSucesso():**
  Verifica o cadastro bem-sucedido de um produto do tipo comida. Simula o cadastro de um produto válido e confirma se todas as propriedades do produto foram corretamente armazenadas.
- **testCadastrarProdutoComidaCamposFaltando():**
  Testa o cadastro de um produto de comida com campos obrigatórios faltando (ex. nome vazio). Lança *ArgumentoInvalidoException* com uma mensagem específica sobre a ausência de campos.
- **testCadastrarProdutoComidaNomeInvalido():**
  Verifica o comportamento ao tentar cadastrar um produto de comida com um nome inválido (ex. numérico). Lança *ArgumentoInvalidoException* com uma mensagem sobre nome inválido.
- **testCadastrarProdutoComidaPrecoInvalido():**
  Testa o cadastro de um produto de comida com um preço inválido (ex. negativo). Lança *ArgumentoInvalidoException* com uma mensagem sobre preço inválido.
- **testCadastrarProdutoComidaQuantidadeInvalida():**
  Verifica o comportamento ao tentar cadastrar um produto de comida com quantidade inválida (ex. negativa). Lança *ArgumentoInvalidoException* com uma mensagem sobre quantidade inválida.
- **testCadastrarProdutoComidaPesoInvalido():**
  Testa o cadastro de um produto de comida com peso inválido (ex. negativo). Lança *ArgumentoInvalidoException* com uma mensagem sobre peso inválido.
- **testCadastrarProdutoBebidaComSucesso():**
  Verifica o cadastro bem-sucedido de um produto do tipo bebida. Simula o cadastro de um produto válido e confirma se todas as propriedades do produto foram corretamente armazenadas.
- **testCadastrarProdutoBebidaCamposFaltando():**
  Testa o cadastro de um produto de bebida com campos obrigatórios faltando (ex. nome vazio). Lança *ArgumentoInvalidoException* com uma mensagem específica sobre a ausência de campos.
- **testCadastrarProdutoBebidaNomeInvalido():**
  Verifica o comportamento ao tentar cadastrar um produto de bebida com um nome inválido (ex. numérico). Lança *ArgumentoInvalidoException* com uma mensagem sobre nome inválido.
- **testCadastrarProdutoBebidaPrecoInvalido():**
  Testa o cadastro de um produto de bebida com um preço inválido (ex. negativo). Lança *ArgumentoInvalidoException* com uma mensagem sobre preço inválido.
- **testCadastrarProdutoBebidaQuantidadeInvalida():**
  Verifica o comportamento ao tentar cadastrar um produto de bebida com quantidade inválida (ex. negativa). Lança *ArgumentoInvalidoException* com uma mensagem sobre quantidade inválida.
- **testCadastrarProdutoBebidaMlInvalido():**
  Testa o cadastro de um produto de bebida com volume inválido (ex. negativo). Lança *ArgumentoInvalidoException* com uma mensagem sobre volume inválido.

- **testRemoverProdutoComSucesso():**
  Verifica a remoção bem-sucedida de um produto cadastrado. Simula a remoção de um produto existente e confirma se ele foi removido da lista.
- **testRemoverProdutoNomeVazio():**
  Testa a remoção de um produto com nome vazio. Lança *ArgumentoInvalidoException* com uma mensagem sobre nome vazio.
- **testRemoverProdutoNomeInvalido():**
  Verifica a remoção de um produto com nome inválido (ex. numérico). Lança *ArgumentoInvalidoException* com uma mensagem sobre nome inválido.
- **testRemoverProdutoNaoExistente():**
  Testa a remoção de um produto que não existe na lista. Lança *ArgumentoInvalidoException* com uma mensagem sobre o produto não existente.

Classe EditaProdutoTest
Pacote: EditaProduto

- **testAddQuantidadeComidaComSucesso():**
  Este teste verifica se a quantidade de um produto de comida (ProdutoComida) é incrementada corretamente quando uma quantidade válida é adicionada.
- **testAddQuantidadeBebidaComSucesso():**
  Este teste verifica se a quantidade de um produto de bebida (ProdutoBebida) é incrementada corretamente quando uma quantidade válida é adicionada.
- **testAddQuantidadeParametrosVazios():**
  Este teste verifica o comportamento do método addQuantidade ao receber parâmetros vazios (null ou strings vazias).
- **testAddQuantidadeNomeNaoTexto():**
  Este teste verifica o comportamento do método addQuantidade quando o nome do produto passado não é uma string de texto válida (por exemplo, um número ou um objeto).
- **testAddQuantidadeNaoInteira():**
  Este teste verifica o comportamento do método addQuantidade quando a quantidade passada não é um número inteiro (por exemplo, uma string ou um número decimal).
- **testAddQuantidadeProdutoNaoExistente()**:
  Este teste verifica o comportamento do método addQuantidade quando o nome do produto passado não corresponde a nenhum produto existente no sistema.
- **testAddQuantidadeNegativa():**
  Este teste verifica o comportamento do método addQuantidade quando uma quantidade negativa é adicionada a um produto.
- **testAddQuantidadeExtremamenteGrande():**
  Este teste verifica o comportamento do método addQuantidade quando uma quantidade extremamente grande é adicionada a um produto.
- **testRmQuantidadeComidaComSucesso():**
  Este teste verifica se a quantidade de um produto de comida (ProdutoComida) é reduzida corretamente quando uma quantidade válida é removida.

- **testRmQuantidadeBebidaComSucesso():**
  Este teste verifica se a quantidade de um produto de bebida (ProdutoBebida) é reduzida corretamente quando uma quantidade válida é removida.
- **testRmQuantidadeParametrosVazios():**
  Este teste verifica o comportamento do método rmQuantidade ao receber parâmetros vazios (null ou strings vazias).
- **testRmQuantidadeNomeNaoTexto():**
  Este teste verifica o comportamento do método rmQuantidade quando o nome do produto passado não é uma string de texto válida (por exemplo, um número ou um objeto).
- **testRmQuantidadeNaoInteira():**
  Este teste verifica o comportamento do método rmQuantidade quando a quantidade passada não é um número inteiro (por exemplo, uma string ou um número decimal).
- **testRmQuantidadeProdutoNaoExistente():**
  Este teste verifica o comportamento do método rmQuantidade quando o nome do produto passado não corresponde a nenhum produto existente no sistema.
- **testRmQuantidadeNegativa():**
  Este teste verifica o comportamento do método rmQuantidade quando uma quantidade negativa é removida de um produto.
- **testRmQuantidadeMaiorQueExistente():**
  Este teste verifica o comportamento do método rmQuantidade quando uma quantidade maior do que a disponível é removida de um produto.

Classe ProdutoBebidaTest
Pacote: Produto.ProdutoBebida

- **testGetType()**: Verifica o retorno do tipo de produto ('Bebida').
- **testGetPeso()**: Verifica se o peso é 0 para bebidas.
- **testGetMl()**: Verifica o volume da bebida.

Classe ProdutoComidaTest
Pacote: Produto.ProdutoComida

- **testGetType()**: Verifica o retorno do tipo de produto ('Comida').
- **testGetPeso()**: Verifica o peso do produto.
- **testGetMl()**: Verifica se o volume é 0 para comida.

Classe ProdutoTest
Pacote: Produto

- **testGettersProdutoComida():**

Este teste verifica os métodos "getter" de um objeto ProdutoComida. Ele testa se os métodos de acesso de atributos (getId, getPreco, getQuantidade, getPeso, getMl, getType) retornam os valores corretos para o objeto simulado.

- **testGettersProdutoBebida():**
  Este teste verifica os métodos "getter" de um objeto ProdutoBebida. Assim como no teste anterior, ele valida se os métodos de acesso dos atributos retornam os valores corretos para o objeto simulado.
- **testAddQuantidadeProdutoComida():**
  Testa o comportamento do método addQuantidade da classe ProdutoComida. O teste verifica se a quantidade de um produto de comida é corretamente incrementada ao adicionar uma quantidade válida.
- **testAddQuantidadeProdutoBebida():**
  Testa o comportamento do método addQuantidade da classe ProdutoBebida. O teste verifica se a quantidade de um produto de bebida é corretamente incrementada ao adicionar uma quantidade válida.
- **testRmQuantidadeProdutoComida():**
  Verifica o comportamento do método rmQuantidade de um objeto ProdutoComida. O teste simula a remoção de uma quantidade do produto e verifica se a quantidade total é atualizada corretamente.
- **testRmQuantidadeProdutoBebida():**
  Verifica o comportamento do método rmQuantidade de um objeto ProdutoBebida. O teste simula a remoção de uma quantidade do produto e verifica se a quantidade total é atualizada corretamente.
- **testAddQuantidadeInvalidaProdutoComida():**
  Este teste verifica se o método addQuantidade da classe ProdutoComida lança uma exceção ao tentar adicionar uma quantidade inválida (número negativo).
- **testAddQuantidadeInvalidaProdutoBebida():**
  Similar ao teste anterior, mas aplicado a um objeto ProdutoBebida. Verifica se o método addQuantidade lança uma exceção ao adicionar uma quantidade inválida (número negativo).
- **testRmQuantidadeInvalidaProdutoComida():**
  Este teste verifica se o método rmQuantidade da classe ProdutoComida lança uma exceção ao tentar remover uma quantidade maior do que a disponível.
- **testRmQuantidadeInvalidaProdutoBebida():**
  Similar ao teste anterior, mas aplicado a um objeto ProdutoBebida. Verifica se o método rmQuantidade lança uma exceção ao tentar remover uma quantidade maior do que a disponível.
- **testToStringProdutoComida():**
  Este teste verifica o método toString de um objeto ProdutoComida. Ele testa se o método retorna a representação correta do objeto em formato de string.
- **testToStringProdutoBebida():**
  Este teste verifica o método toString de um objeto ProdutoBebida. Ele testa se o método retorna a representação correta do objeto em formato de string.

Classe UtilsTest
Pacote: Utils

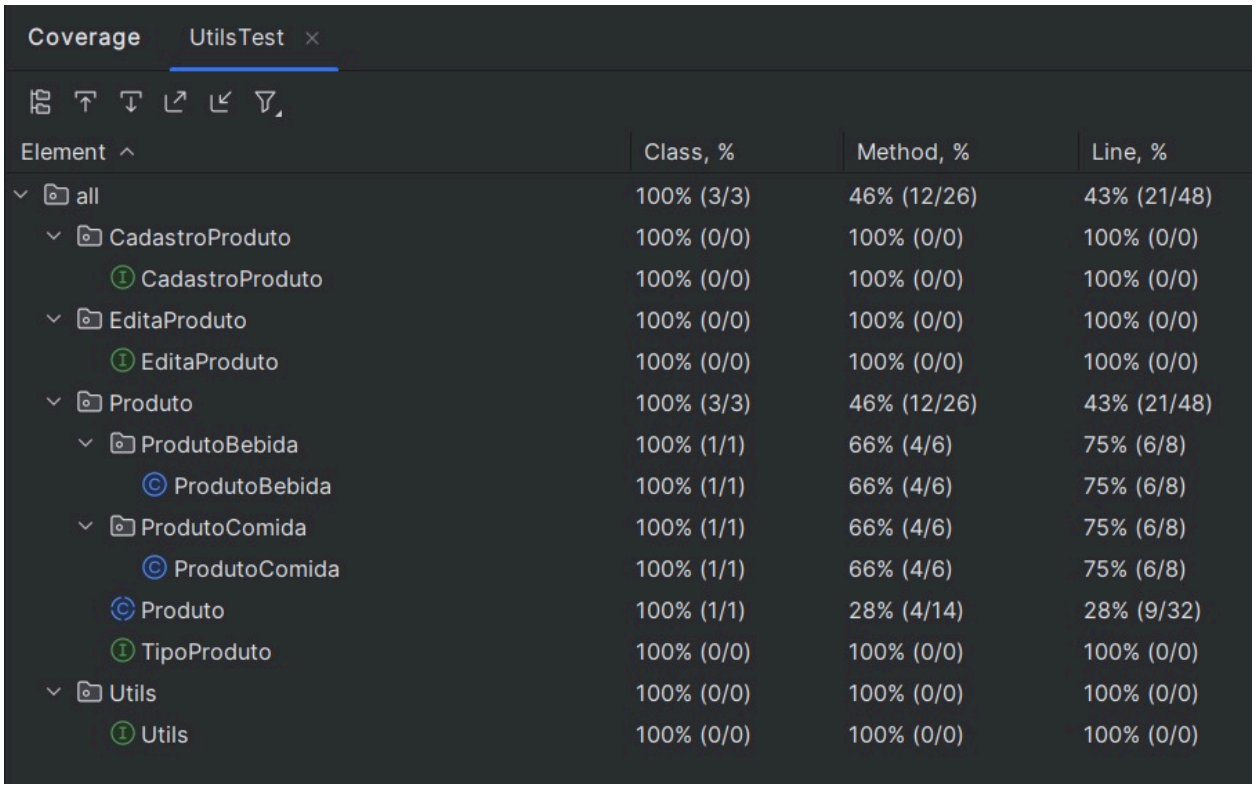
- **testVisualizarValorTotalComSucesso()**
  Este teste verifica se o método valorTotal calcula corretamente o valor total dos produtos na lista. Ele adiciona dois produtos (um de comida e um de bebida) à lista produtos, simula o método valorTotal e verifica se o resultado retornado é o esperado.
- **testVisualizarValorTotalComParametrosInexistentes()**
  Este teste verifica o comportamento do método valorTotal quando o parâmetro passado é null. Ele espera que o método lance uma NullPointerException com uma mensagem específica.
- **testVisualizarValorTotalComListaVazia()**
  Este teste verifica se o método valorTotal retorna o valor correto quando a lista de produtos está vazia. A lista produtos é passada vazia e o método simulado deve retornar um total de "R$ 0,00".
- **testBuscarProdutosChaveComSucesso()**
  Este teste verifica o método searchProduto quando uma chave de busca válida é fornecida. Ele adiciona produtos à lista e verifica se o método retorna o produto correto quando pesquisado por uma palavra-chave.
- **testBuscarProdutosChaveVazia()**
  Este teste verifica o comportamento do método searchProduto quando a chave de busca é uma string vazia. Ele espera que uma *ArgumentoInvalidoException* seja lançada.
- **testBuscarProdutosChaveNaoTexto()**
  Este teste verifica o comportamento do método searchProduto quando a chave de busca não é texto (por exemplo, números). Ele espera que uma *ArgumentoInvalidoException* seja lançada.
- **testBuscarProdutosChaveNaoExistente():**
  Este teste verifica o comportamento do método searchProduto quando a chave de busca não corresponde a nenhum produto na lista. Ele espera que uma ArgumentoInvalidoException seja lançada.
- **testBuscarProdutosListaVazia()**
  Este teste verifica o comportamento do método searchProduto quando a lista de produtos está vazia. Ele espera que uma *ArgumentoInvalidoException* seja lançada.
- **testLimparListaDeComprasComSucesso()**
  Este teste verifica o método limparLista para garantir que ele limpa a lista de produtos com sucesso. Após limpar a lista, o teste verifica se a lista está vazia.
- **testLimparListaDeComprasComParametrosInexistentes()**
  Este teste verifica o comportamento do método limparLista quando o parâmetro fornecido é null. Ele espera que uma *NullPointerException* seja lançada.
- **testLimparListaDeComprasComListaVazia()**
  Este teste verifica o método limparLista quando a lista de produtos já está vazia. O teste certifica-se de que a lista permaneça vazia após a chamada do método.
- **testVisualizarListaDeComprasComSucesso()**

Este teste verifica o método toString para garantir que ele retorna a representação correta em string da lista de produtos. Ele adiciona produtos à lista e verifica se a string retornada é formatada corretamente.

- **testVisualizarListaDeComprasComParametrosInexistentes()**
  Este teste verifica o comportamento do método toString quando o parâmetro fornecido é null. Ele espera que uma NullPointerException seja lançada.
- **testVisualizarListaDeComprasComListaVazia()**
  Este teste verifica o método toString para garantir que ele retorna uma representação de string apropriada para uma lista de produtos vazia.
- **testEncerrarProgramaComSucesso()**
  Este teste simplesmente imprime uma mensagem indicando que o programa foi encerrado com sucesso.

Cobertura de Testes

A cobertura de testes é uma métrica importante que reflete o quanto do código de um software é testado por meio de testes automatizados. Ela é essencial para garantir a qualidade e a confiabilidade do software. Testes com alta cobertura garantem que a maioria das funcionalidades do sistema foi verificada quanto ao seu funcionamento correto.´No relatório de cobertura de testes que fizemos, a cobertura foi analisada para todas as classes do sistema.



Element	Class, %	Method, %	Line, %
all	100% (3/3)	46% (12/26)	43% (21/48)
CadastroProduto	100% (0/0)	100% (0/0)	100% (0/0)
CadastroProduto	100% (0/0)	100% (0/0)	100% (0/0)
EditaProduto	100% (0/0)	100% (0/0)	100% (0/0)
EditaProduto	100% (0/0)	100% (0/0)	100% (0/0)
Produto	100% (3/3)	46% (12/26)	43% (21/48)
ProdutoBebida	100% (1/1)	66% (4/6)	75% (6/8)
ProdutoBebida	100% (1/1)	66% (4/6)	75% (6/8)
ProdutoComida	100% (1/1)	66% (4/6)	75% (6/8)
ProdutoComida	100% (1/1)	66% (4/6)	75% (6/8)
Produto	100% (1/1)	28% (4/14)	28% (9/32)
TipoProduto	100% (0/0)	100% (0/0)	100% (0/0)
Utils	100% (0/0)	100% (0/0)	100% (0/0)
Utils	100% (0/0)	100% (0/0)	100% (0/0)

Embora o projeto tenha atingido uma excelente cobertura de classes, o que é fundamental, há espaço para expandir a cobertura de métodos e linhas de código.