**UNIVERSIDADE ESTADUAL DE MARINGÁ**

**DEPARTAMENTO DE INFORMÁTICA**

**CIÊNCIA DA COMPUTAÇÃO**

**PROJETO DE SISTEMA DE SOFTWARE**

**PROFESSOR: DONIZETE CARLOS BRUZAROSCO**

**TRABALHO DE PROJETO DE IMPLEMENTAÇÃO DE SISTEMAS DE SOFTWARE (ISS)**

JOÃO PEDRO BALESTRA ZACARKIN RA: 83967

MATHEUS ALBERTO VAZ CECILIO RA: 85582

THIAGO DE OLIVEIRA LAVEZZO RA: 84753

THIAGO VIDAL DE MORAES RA: 82349

**MARINGÁ**

**01 DE FEVEREIRO DE 2016**

**ÍNDICE**

**1 ª SPRINT:**

Gerenciamento de Usuários e Autenticar Usuário (Thiago Moraes)

Gerenciamento de Clientes(Matheus Alberto)

Gerenciamento de Produtos (Thiago Lavezzo)

Gerenciamento de Fornecedor(João Pedro)

**2ª SPRINT:**

Gerenciamento de Pagamento (Thiago Moraes)

Gerar Relatórios (Thiago Lavezzo)

Gerenciamento de Vendas - Pedidos (Matheus Alberto)

Simular Manipulação (João Pedro)

**Thiago Vidal de Moraes:**

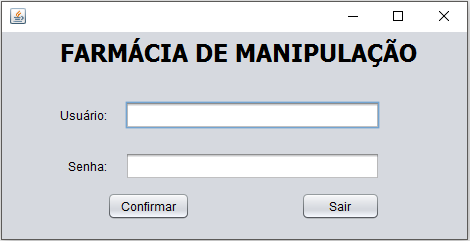
1. Sprint 1: RF002 Gerenciamento de Usuários

* 20h
* Telas:

1. Sprint 1: RF003 Autenticar Usuário

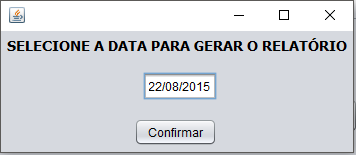
* 4h
* Telas:



1. Sprint 3: RF008 Gerar Relatórios

* 30h
* Telas:

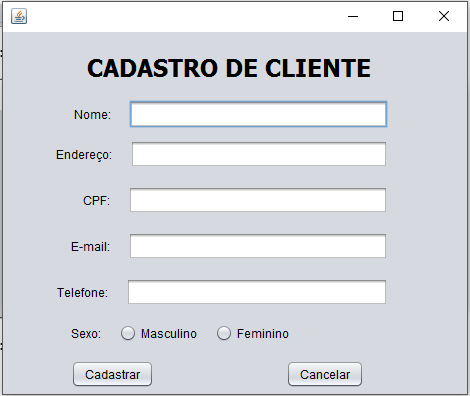
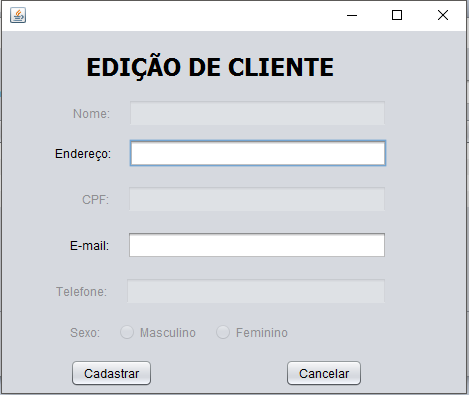
Obs.: Está faltando os filtros, mas o mesmo será implementado.



**Matheus Alberto Vaz Cecilio:**

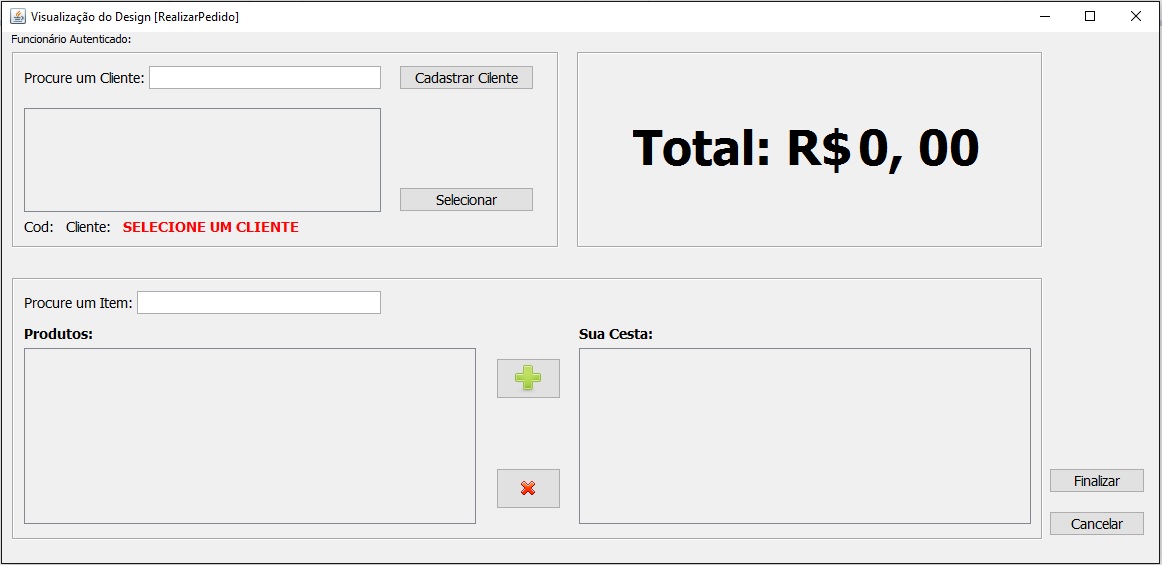
1. Sprint 1: RF001 Gerenciamento de Clientes

* 20h
* Telas:

1. Sprint 2: RF006 Gerenciamento de Vendas

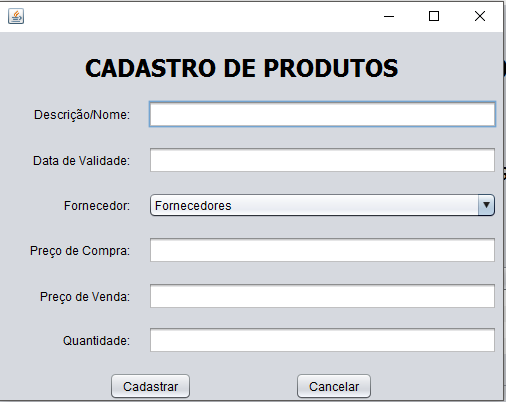
* 35h
* Telas:



**Thiago de Oliveira Lavezzo:**

1. Sprint 1: RF004 Gerenciar Estoque

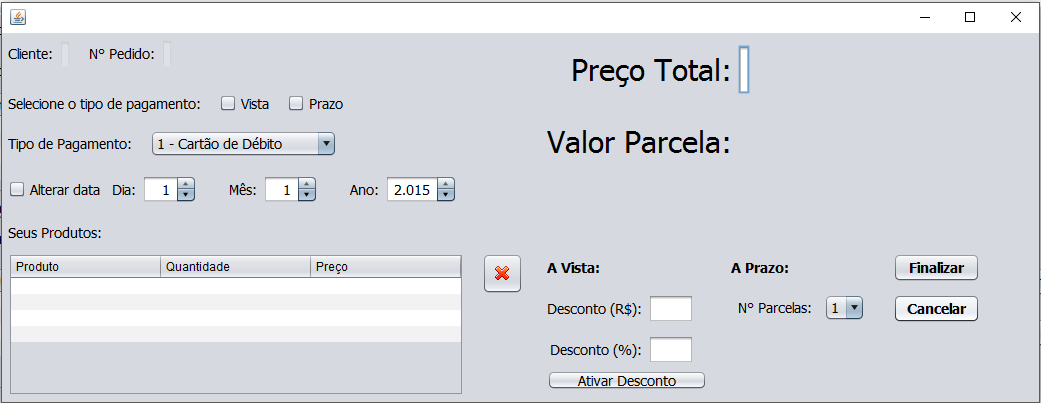
* 16h
* Telas:

1. Sprint 2: RF009 Gerenciamento de Pagamento

* 30h
* Telas:





**João Pedro Balestra Zacarkin:**

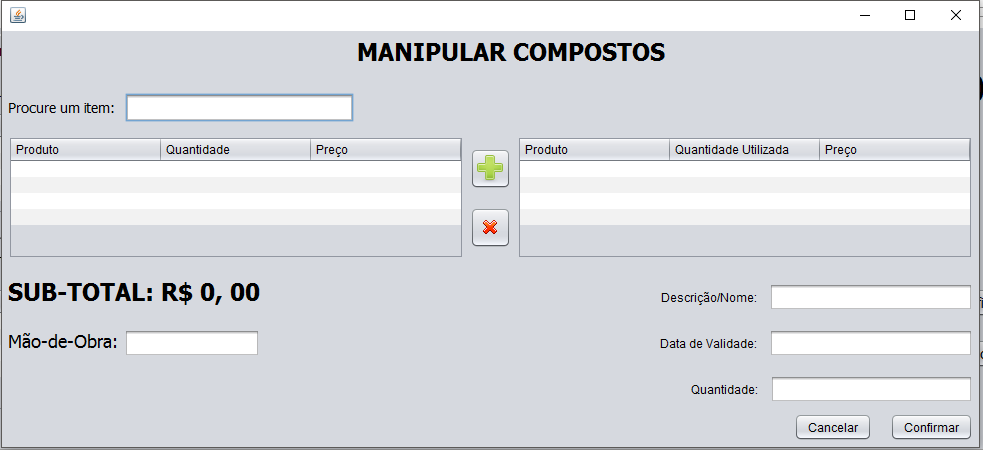
1. Sprint 1: RF007 Gerenciamento de Fornecedor

* 20h
* Telas:

1. Sprint 2: RF005 Simular Manipulação:

* 30h
* Telas:

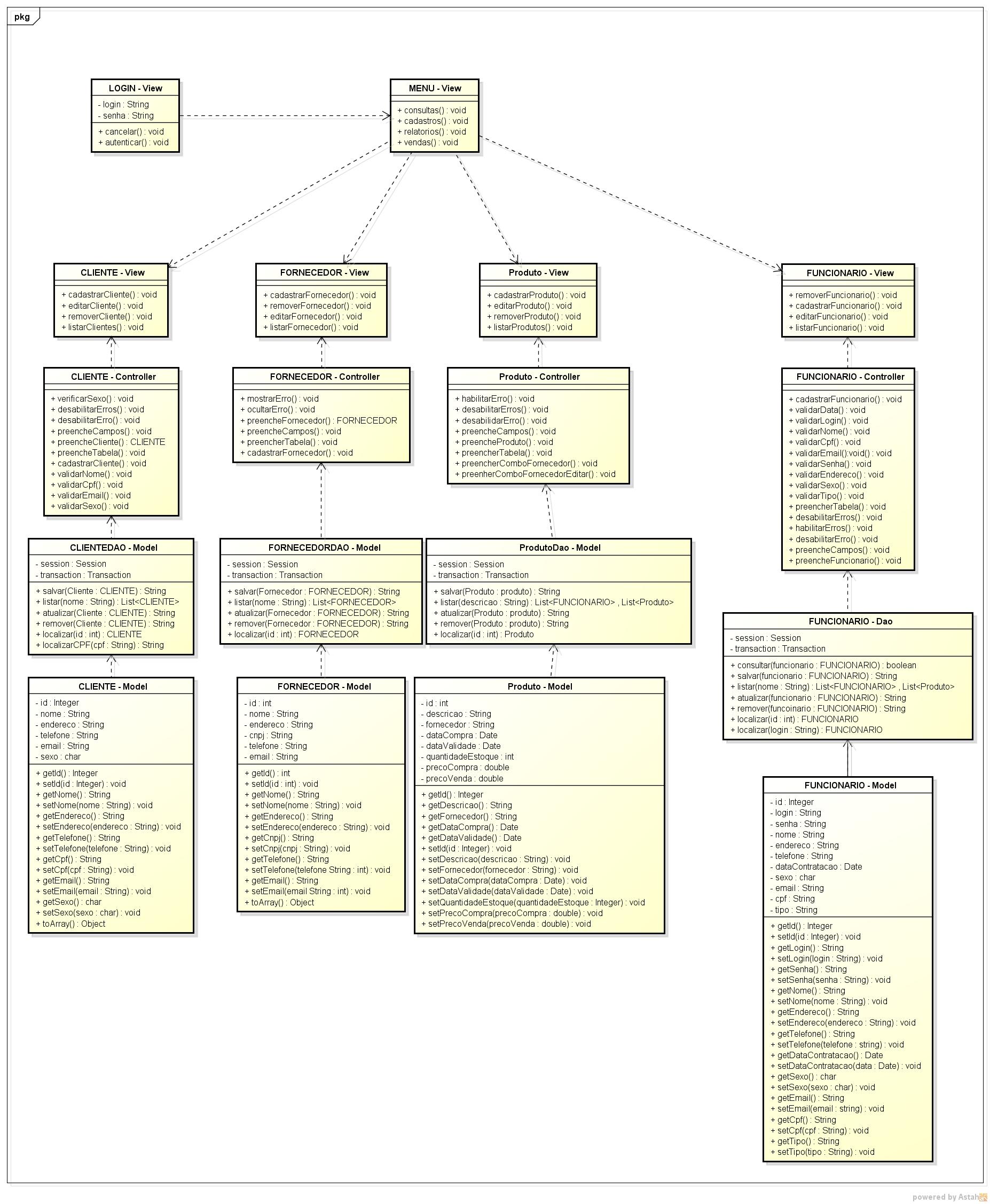


**Frameworks:**

- Hibernate: “É um software livre de código aberto distribuído com licença LGPL. O objetivo do Hibernate é diminuir a complexidade entre os programas Java, baseado no modelo orientado a objeto, que precisam trabalhar com um banco de dados do modelo relacional (presente na maioria dos SGBDs). ”

- Swing Java: “**Swing** é um *[widget toolkit](https://pt.wikipedia.org/wiki/Toolkit" \o "Toolkit)* para uso com o [Java](https://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o)). Ele é compatível com o [Abstract Window Toolkit (AWT)](https://pt.wikipedia.org/wiki/Abstract_Window_Toolkit), mas trabalha de uma maneira totalmente diferente. A API Swing procura renderizar/desenhar por conta própria todos os componentes, ao invés de delegar essa tarefa ao sistema operacional, como a maioria das outras APIs de interface gráfica trabalham. ”

**Diagrama MVC – Primeira Sprint:**



**Diagrama MVC – Segunda Sprint:**

**Planejamento dos testes:**

**Prazo – Cronograma**

**Recursos:**

**Pessoas:** Equipe de desenvolvimento do projeto.

**Ferramentas:** Softwares especializados e computadores da própria equipe.

**Estratégia de testes**

**Teste Caixa Preta:**

* Matheus Alberto Vaz Cecilio
  + **PARTICIONAMENTO DE EQUIVALÊNCIA**
  + Funcionalidade: Adicionar produto na cesta
  + Variável de entrada: Produto (idProduto).
  + Variáveis internas: não possui.
  + Variáveis de saída: Mensagem “Valor Inválido”, “Produto não encontrado” e o produto é aceito na cesta, que na tela se dá com a inserção do produto na tabela que corresponde à cesta de produtos para o pedido.
  + Observação: O produto, que é adquirido com base no id do produto, possui um atributo que é o seu preço de venda. Sendo assim, temos também como variável o preço de venda.
  + Classes de equivalência:
    - **Variável Preço de venda do Produto (preçoVenda).**
    - Classe 1: meuProduto.getPreçoVenda() <= 0 (Classe inválida)
    - Classe 2: meuProduto.getPreçoVenda() > 0 (Classe válida)
    - **Variável ID PRODUTO**
    - Classe 3: idProduto ϵ {Conjunto de todos os produtos existentes no banco de dados} (Classe válida)
    - Classe 4: idProduto ∉ {Conjunto de todos os produtos existentes no banco de dados} (Classe inválida)
    - **Variável Mensagem**
    - Classe 5: Valor inválido (Classe válida)
    - Classe 6: Produto não encontrado (Classe válida)
    - **Variável produto inserido na cesta**
    - Classe 7: Produto inserido na cesta (Classe válida).
  + Definição dos casos de teste:
    - Caso de teste 1:

|  |  |
| --- | --- |
| **VARIÁVEL (CLASSE COBERTA)** | **RESULTADO ESPERADO** |
| meuProduto.getPreçoVenda() = 25 (2) | Produto inserido na cesta |
| idProduto ϵ CONJUNTO DOS PRODUTOS (3) |  |

* + - Caso de teste 2:

|  |  |
| --- | --- |
| **VARIÁVEL (CLASSE COBERTA)** | **RESULTADO ESPERADO** |
| meuProduto.getPreçoVenda() = -1 (1) | Valor inválido |
| idProduto ϵ CONJUNTO DOS PRODUTOS (3) |  |

* + - Caso de teste 3:

|  |  |
| --- | --- |
| **VARIÁVEL (CLASSE COBERTA)** | **RESULTADO ESPERADO** |
| meuProduto.getPreçoVenda() = 25 (2) | Produto não encontrado |
| idProduto ∉ CONJUNTO DOS PRODUTOS (4) |  |

* + - Caso de teste 4:

|  |  |
| --- | --- |
| **VARIÁVEL (CLASSE COBERTA)** | **RESULTADO ESPERADO** |
| meuProduto.getPreçoVenda() = -3 (1) | Produto não encontrado |
| idProduto ∉ CONJUNTO DOS PRODUTOS (4) |  |

* + **ANÁLISE DE VALOR LIMITE**
  + Observação: Será realizado a análise com base no preço de venda de algum produto.
  + Variável: meuProduto.getPreçoVenda(), adotarei neste teste o nome precoVenda.

Classe de equivalência: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_0\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* + Definição dos casos de teste:

|  |  |  |
| --- | --- | --- |
| **CASO DE TESTE** | **precoVenda** | **Resultado esperado** |
| 1 | -1 | “Valor Inválido” |
| 2 | 0 | “Valor Inválido” |
| 3 | 1 | Produto inserido na cesta |

**Teste Caixa Branca:**

* Matheus Alberto Vaz Cecilio
  + Funcionalidade: Clique do botão para finalizar um pedido.
  + Observações: O clique do botão é implementado automaticamente pela IDE Netbeans, não permitindo a passagem de parâmetros, sendo assim as variáveis são globais e estão na própria interface que, as enxergam para realizar algumas funcionalidades.
  + Variáveis globais do código que deverão conter algum valor:

private final List<Produto> listaCesta = new ArrayList<>();

private final int idFuncionario;

private javax.swing.JLabel txtIdCliente;

* + Código:

private void btnFinalizarPedidoActionPerformed(java.awt.event.ActionEvent evt) {

1 if (txtIdCliente.getText().equals("")) { //VAZIO, NÃO SELECIONOU CLIENTE

2 pedidoController.habilitarNaoCliente(labelCod, labelCliente, txtNaoCliente); //HABILITA UMA LABEL QUE DIZ QUE SE DEVE SELECIONAR UM CLIENTE

3 JOptionPane.showMessageDialog(this, "Selecione um cliente para continuar!", "Algo deu errado.", JOptionPane.ERROR\_MESSAGE);

4 } else if (listaCesta.isEmpty()) {

5 JOptionPane.showMessageDialog(this, "Não é possível continuar, sua cesta está vazia, insira algum item!", "Algo deu errado.", JOptionPane.ERROR\_MESSAGE);

6 } else {

7 pedidoController.desabilitarNaoCliente(labelCod, labelCliente, txtNaoCliente);

8 Cliente cliente = new ClienteDao().localizar(Integer.parseInt(txtIdCliente.getText()));

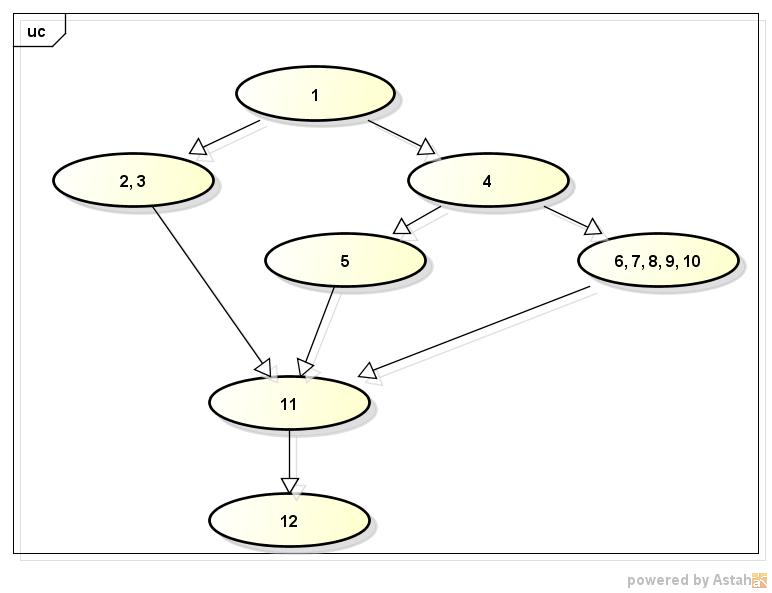
9 Funcionario funcionario = new FuncionarioDao().localizar(idFuncionario);

10 pedidoController.finalizarPedido(this, cliente, funcionario, listaCesta, total);

11 } //FIM SE

12 } //FIM MÉTODO

* + Grafo:



* + Complexidade Ciclomática (CC):

E -> Número de ramos do grafo (arco/aresta);

N -> Número de nós (vértice).

CC = E – N + 2 -> 8 – 7 + 2 = **3** (Três testes)

* + Caminhos independentes:
    - 1-2-3-11-12:
      * Casos de teste:
        + txtIdCliente = “”, listaCesta = [Produto2, Produto5, Produto20] .
      * Resultados:
        + Mensagem "Selecione um cliente para continuar!" na tela, o método é encerrado, continua na tela de pedido.
    - 1-4-5-11-12:
      * Casos de teste:
        + txtIdCliente = “122”, listaCesta = [].
      * Resultados:
        + Mensagem "Não é possível continuar, sua cesta está vazia, insira algum item!" na tela, o método é encerrado, continua na tela de pedido.
    - 1-4-6-7-8-9-10-11-12:
      * Casos de teste:
        + txtIdCliente = “122”, listaCesta = [Produto2, Produto5, Produto20].
      * Resultados:
        + Mensagem "Pedido feito com sucesso!"" na tela, e é feito o pedido e o sistema encaminha o usuário pra outra tela.

Observação: A mensagem neste caso de teste é mostrada através do controller, que antes de exibir a mensagem ele simplesmente faz uma confirmação do pedido. Por isso a mensagem não se encontra no código, mas sim no controller, através do método “finalizarPedido”.

**Padrão de Projeto:**

- Nesta primeira Sprint, foi usado apenas o *Singleton*, outro(s) padrão(s) de projeto serão utilizados no desenvolvimento da segunda sprint, no próximo semestre