**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:**

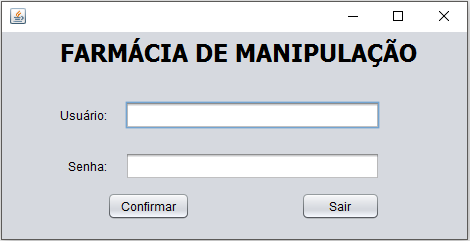
1. Sprint 1: RF002 Gerenciamento de Usuários

* 20h
* Telas:

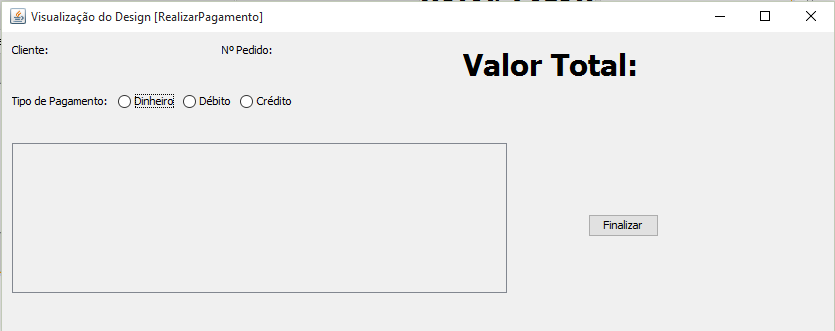
1. Sprint 1: RF003 Autenticar Usuário

* 4h
* Telas:



1. Sprint 3: RF009 Gerenciamento de Pagamento

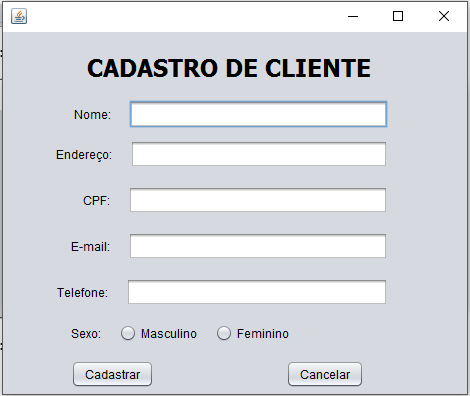
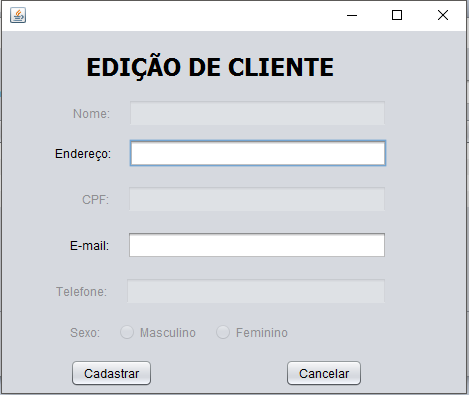
* 30h
* Telas:



**Matheus Alberto Vaz Cecilio:**

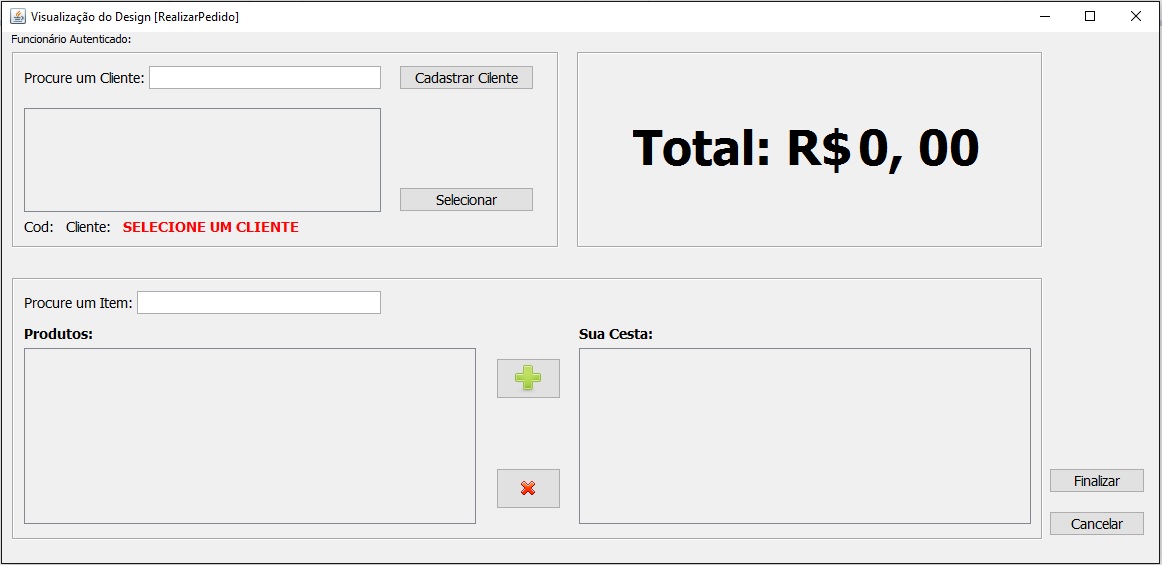
1. Sprint 1: RF001 Gerenciamento de Clientes

* 20h
* Telas:

1. Sprint 2: RF006 Gerenciamento de Vendas

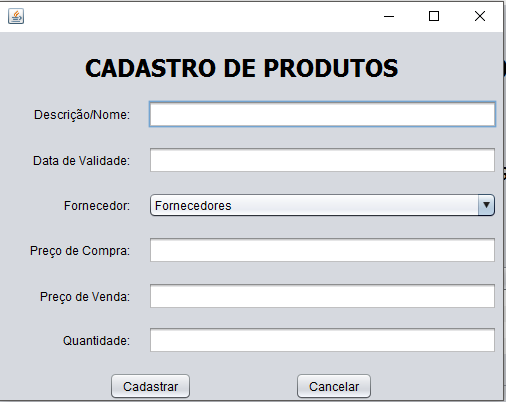
* 35h
* Telas:



**Thiago de Oliveira Lavezzo:**

1. Sprint 1: RF004 Gerenciar Estoque

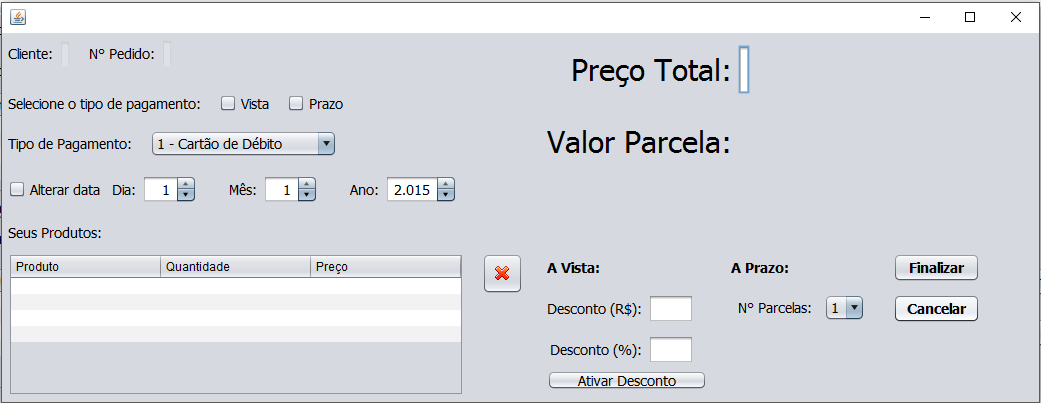
* 16h
* Telas:

1. Sprint 2: RF009 Gerenciamento de Pagamento

* 30h
* Telas:





**João Pedro Balestra Zacarkin:**

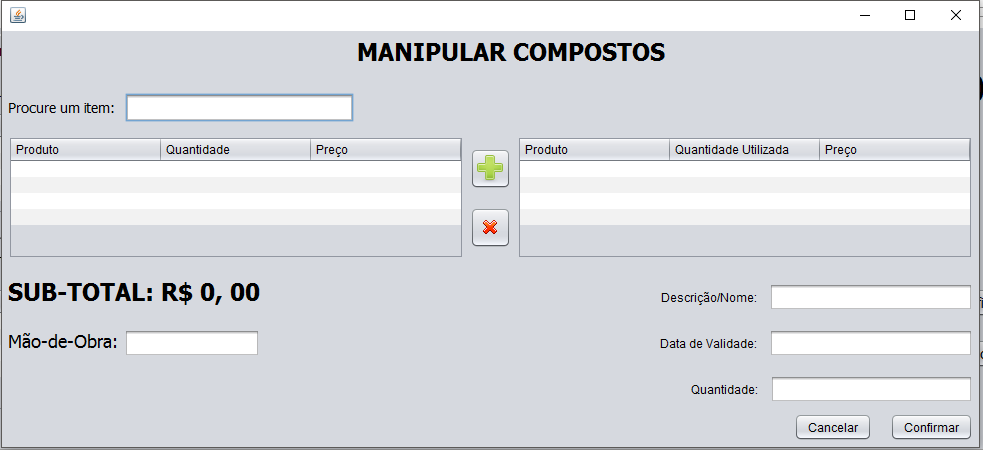
1. Sprint 1: RF007 Gerenciamento de Fornecedor

* 20h
* Telas:

1. Sprint 2: RF005 Simular Manipulação:

* 30h
* Telas:

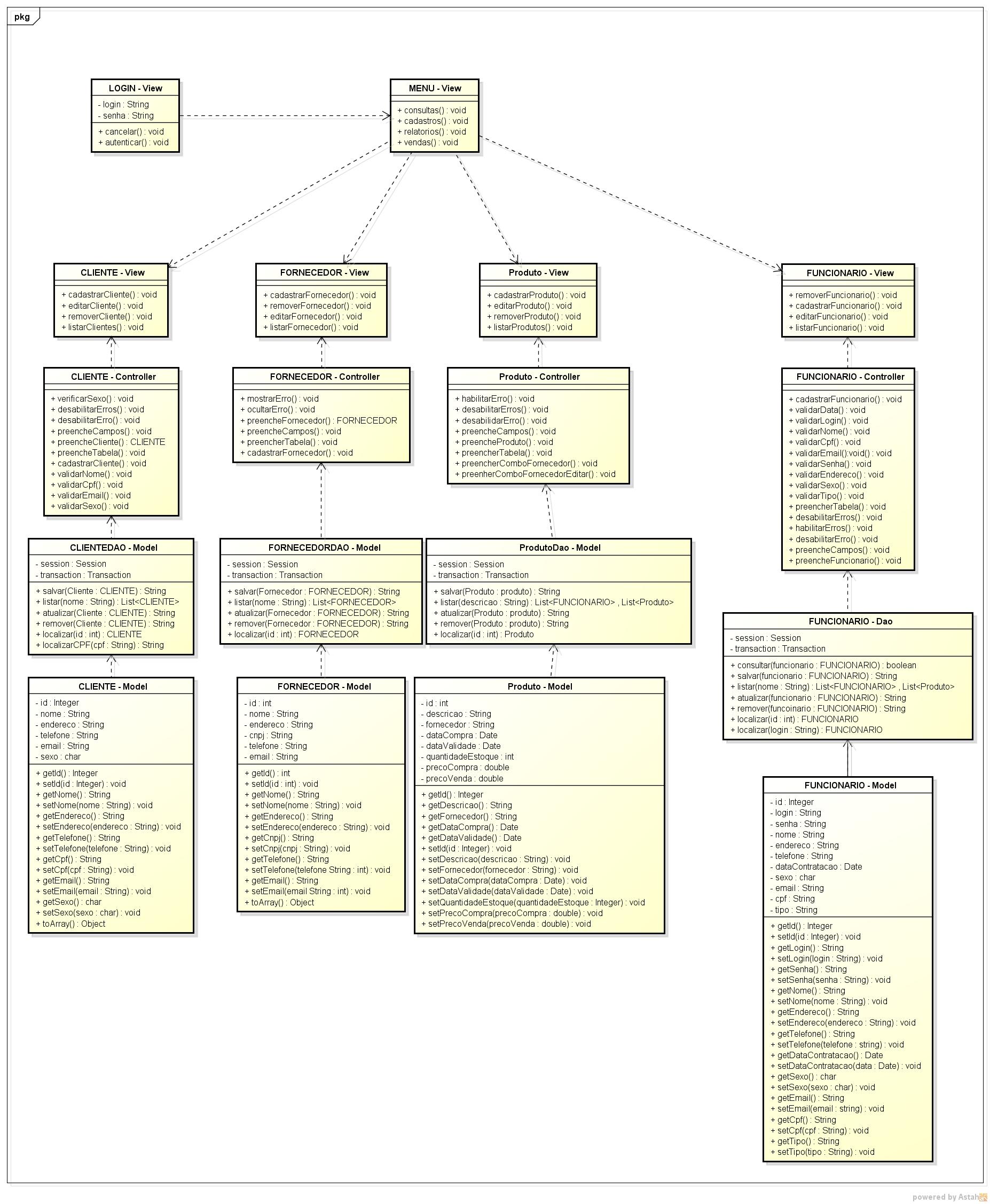


**Frameworks:**

- 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 |

* Thiago Vidal de Moraes
  + **PARTICIONAMENTO DE EQUIVALÊNCIA**
  + Funcionalidade: Finalizar Pagamento
  + Variáveis de Entrada: Desconto e radio button da escolha do tipo de pagamento
  + Variáveis internas: não possui.
  + Variáveis de saída: Mensagens “Selecione um tipo de pagamento”(1), “Pagamento efetuado com sucesso”(2), “Tente Novamente”(3) e “Valor de desconto inválido”(4).
  + Classes de equivalência:
    - **Variável Desconto**
    - Classe 1: txtDesconto.getText() < 0 (Classe inválida)
    - Classe 2: txtDesconto.getText() >= 0 e < txtTotal.getText() (Classe válida)
    - Classe 3: txtDesconto.getText() > txtTotal.getText() (Classe inválida)
    - **Variável Radio Button**
    - Classe 3: nenhuma radio button selecionada (Classe inválida)
    - Classe 4: qualquer radio button selecionada (Classe válida)
  + Definição dos casos de teste:
    - Caso de teste 1:

|  |  |
| --- | --- |
| **VARIÁVEL (CLASSE COBERTA = 1)** | **RESULTADO ESPERADO** |
| txtDesconto.getText() = -1.2 | Mensagem 4 |

* + - Caso de teste 2:

|  |  |
| --- | --- |
| **VARIÁVEL (CLASSE COBERTA = 2)** | **RESULTADO ESPERADO** |
| txtDesconto.getText() = 0 | Valor válido |

* + - Caso de teste 3:

|  |  |
| --- | --- |
| **VARIÁVEL (CLASSE COBERTA 3)** | **RESULTADO ESPERADO** |
| txtDesconto.getText() > txtTotal.getText() | Mensagem 4 |

* + - Caso de teste 4:

|  |  |
| --- | --- |
| **VARIÁVEL (CLASSE COBERTA)** | **RESULTADO ESPERADO** |
| Nenhum radio button selecionado | Mensagem 1 |

* + **ANÁLISE DE VALOR LIMITE**
  + Observação: Será realizado a análise no valor de desconto.
  + Variável: txtDesconto.getText().

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

* + Definição dos casos de teste:

|  |  |  |
| --- | --- | --- |
| **CASO DE TESTE** | **txtDesconto.getText()** | **Resultado esperado** |
| 1 | -1 | Mensagem 4 |
| 2 | 0 | Mensagem 1 |
| 3 | 1 | Mensagem 1 |
| 4 | = txtTotal.getText() – 1 | Mensagem 1 |
| 5 | = txtTotal.getText() | Mensagem 1 |
| 6 | = txtTotal.getText() + 1 | Mensagem 4 |

* João Pedro Balestra Zacarkin
  + **PARTICIONAMENTO DE EQUIVALÊNCIA**
  + **Funcionalidade**: Adicionar composto para manipulação
  + **Variável de entrada**: quantidadeRequerida , unidadeSelecionada (combobox onde o usuário seleciona a unidade de medida para a quantidade requerida)
  + **Variáveis internas**: quantidadeEstoque,unidadeEstoque (unidade de medida que foi cadastrada para a quantidade do produto no estoque).
  + **Variáveis de saída**:

Mensagens :“Quantidade Inválida”, “Quantidade insuficiente em estoque”, “Unidade de medida inválida” e o composto é adicionado à lista de manipulação (mostrado graficamente na tela pela passagem de uma tabela para outra).

* **Classes de equivalência**:
  + **Variável quantidadeRequerida**:
    - Classe 1: quantidadeRequerida >= 1 (Classe Válida)
    - Classe 2: quantidadeRequerida está vazia (Classe Inválida)
    - Classe 3: quantidadeRequerida < 1 (Classe Inválida)
  + **Variável quantidade Estoque**:
    - Classe 4: quantidadeEstoque > quantidadeRequerida (Classe válida)
    - Classe 5: quantidadeEstoque < quantidadeRequerida (Classe inválida)
  + **Variável unidadeSelecionada:**
    - unidadeSelecionada ∈ {Kg, g, mg} :
      * Classe 6: unidadeEstoque ∈ {Kg, g, mg} (Classe Válida)
      * Classe 7: unidadeEstoque ∈ {l, ml} (Classe Inválida)
    - unidadeSelecionada ∈ {l, ml}
      * Classe 8: unidadeEstoque ∈ {l, ml} (Classe Válida)
      * Classe 9: unidadeEstoque ∈ {Kg, g, ml} (Classe Inválida)
  + **Variável Mensagem:**
    - Classe 10: “Quantidade Inválida” (Classe válida)
    - Classe 11: “Quantidade insuficiente em estoque” (Classe Válida)
    - Classe 12: “Unidade de medida inválida” (Classe Válida)
  + **Variável Composto adicionado à lista de manipulação:**
    - Classe 13: O composto é adicionado à tabela de manipulação (Classe Válida)
* **Definição dos casos de Teste**:
  + **Caso de teste 1:**

|  |  |
| --- | --- |
| **Classes Cobertas** | **Resultado Esperado** |
| quantidadeRequerida = 100 (1)  unidadeSelecionada = “g” e unidadeEstoque = “Kg” (6)  quantidadeEstoque = 10 (4) | Composto adicionado à lista de manipulação (13) |

* + **Caso de teste 2:**

|  |  |
| --- | --- |
| **Classes Cobertas** | **Resultado Esperado** |
| quantidadeRequerida = 100 (1)  unidadeSelecionada = “ml” e unidadeEstoque = “l” (8)  quantidadeEstoque = 10 (4) | Composto adicionado à lista de manipulação (13) |

* + **Caso de teste 3:**

|  |  |
| --- | --- |
| **Classes Cobertas** | **Resultado Esperado** |
| quantidadeRequerida = 0 (3) | “Quantidade inválida”(10) |

* + **Caso de teste 4:**

|  |  |
| --- | --- |
| **Classes Cobertas** | **Resultado Esperado** |
| quantidadeRequerida = 11 (2)  unidadeSelecionada = “Kg” e unidadeEstoque = “Kg” (6)  quantidadeEstoque = 10 (5) | “Quantidade insuficiente em estoque”(11) |

* + **Caso de teste 5:**

|  |  |
| --- | --- |
| **Classes Cobertas** | **Resultado Esperado** |
| unidadeSelecionada = “ml” e unidadeEstoque = “Kg” (9) | “Unidade de medida inválida”(12) |

Agora que todas as classes válidas foram cobertas, inicia-se os casos de teste para as classes inválidas ainda não cobertas.

* + **Caso de teste 6:**

|  |  |
| --- | --- |
| **Classes Cobertas** | **Resultado Esperado** |
| unidadeSelecionada = “g” e unidadeEstoque = “ml” (9) | “Unidade de medida inválida”(12) |

* **ANÁLISE DE VALOR LIMITE**
  + **Variável quantidadeRequerida:**

- A variável está no seguinte intervalo:

1 <= quantidadeRequerida <= quantidadeEstoque

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Caso de teste** | **quantidadeRequerida** | **quantidadeEstoque** | **unidadeEstoque** | **unidadeSelecionada** | **Saída Esperada** |
| 1 | 0 | 1000 | g | g | Quantidade inválida. |
| 2 | 1 | 1000 | g | g | Produto Adicionado |
| 3 | 2 | 1000 | g | g | Produto adicionado |

* + **Variável unidadeSelecionada:**

A unidade de medida deve pertencer ao mesmo tipo (Sólido ou Líquido). Assim, o usuário deve selecionar uma unidade compatível a unidade cadastrada no banco de dados.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Caso de teste** | **quantidadeRequerida** | **quantidadeEstoque** | **unidadeEstoque** | **unidadeSelecionada** | **Saída Esperada** |
| 4 | 10 | 1000 | g | mg | Produto adicionado. |
| 5 | 10 | 1000 | g | ml | Unidade Inválida |
| 6 | 10 | 1000 | l | ml | Produto adicionado |
| 7 | 10 | 1000 | L | G | Unidade Inválida. |

* + **Variável quantidadeEstoque:**

A variável quantidadeEstoque deve ser maior ou igual a quantidadeRequerida.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Caso de teste** | **quantidadeRequerida** | **quantidadeEstoque** | **unidadeEstoque** | **unidadeSelecionada** | **Saída Esperada** |
| 8 | 1000 | 999 | g | g | Quantidade insuficiente em estoque |
| 9 | 1000 | 1000 | g | g | Produto adicionado. |
| 10 | 1000 | 1001 | g | g | Produto adicionado. |

**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()));

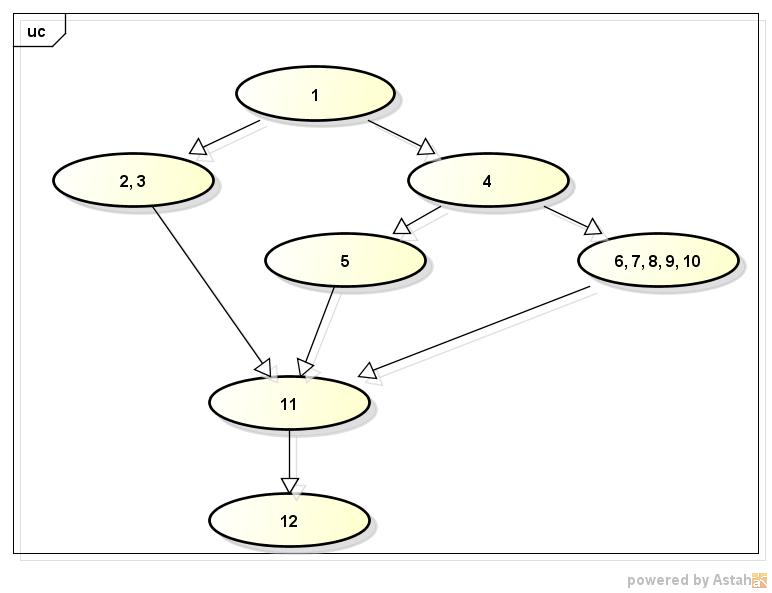
9 Funcionario funcionario = new FuncionarioDao().localizar(idFuncionario);

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)

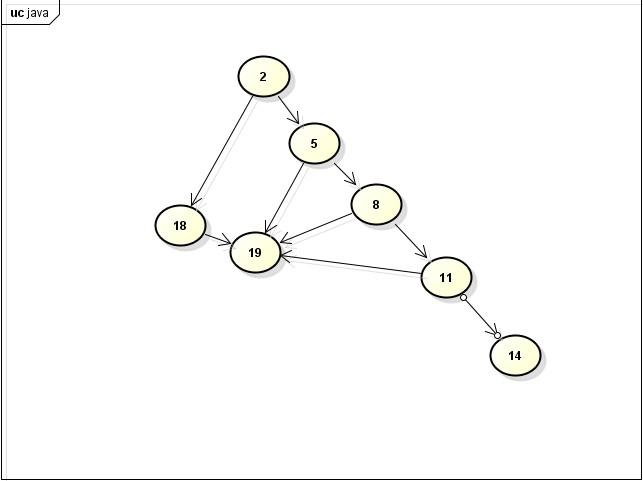
* + Caminhos independentes:
    - 1-2-3-11-12:
      * Casos de teste:
        + txtIdCliente = “”, listaCesta = [Produto2, Produto5, Produto20] .
      * Resultados:
        + Mensagem "Selecione um cliente para continuar!" na tela, o método é encerrado, continua na tela de pedido.
    - 1-4-5-11-12:
      * 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”.

* Thiago Vidal de Moraes:
  + Funcionalidade: botão para finalizar a compra.
  + Código:

1. String tipo = null;
2. if (Double.parseDouble(txtDesconto.getText().replace(',', '.')) < 0.00 ||
3. Double.parseDouble(txtDesconto.getText().replace(',', '.')) > Double.parseDouble(txtTotal.getText().replace(',', '.'))) {
4. JOptionPane.showMessageDialog(this, "Valor de desconto inválido", "Erro", JOptionPane.ERROR\_MESSAGE);
5. } else if (radioDinheiro.isSelected()) {
6. tipo = "0";
7. pagamentoController.pagamento(this, tipo, pedidoFinal.getValorCompra(), pedidoFinal.getCliente(), pedidoFinal, pedidoFinal.getDataPedido(), produtosFinal);
8. } else if (radioDébito.isSelected()) {
9. tipo = "1";
10. pagamentoController.pagamento(this, tipo, pedidoFinal.getValorCompra(), pedidoFinal.getCliente(), pedidoFinal, pedidoFinal.getDataPedido(), produtosFinal);
11. } else if (radioCrédito.isSelected()) {
12. tipo = "2";
13. pagamentoController.pagamento(this, tipo, pedidoFinal.getValorCompra(), pedidoFinal.getCliente(), pedidoFinal, pedidoFinal.getDataPedido(), produtosFinal);
14. } else {
15. labelPagamento.setVisible(true);
16. labelPagamentoObrigatorio.setVisible(true);
17. JOptionPane.showMessageDialog(this, "Selecione um tipo de pagamento.", "Erro", JOptionPane.ERROR\_MESSAGE);
18. }//FIM IF
19. //FIM IF

* Grafo:



* Observação: O nó “Fim” simboliza que caiu no if e saiu da função.
* Complexidade Aciclomática:
  + 9 (arestas) – 7 (nós) + 2 = 4
* Caminhos Independentes:
  + 2 – 18 – 19
    - Caso de teste : valor do desconto menor que zero ou maior que o valor total
    - Mensagem: “Valor de desconto inválido”
  + 2 – 5 – 19
    - Caso de teste: valor de desconto válido e radioDinheiro selecionado
    - Mensagem: “Pagamento efetuado com sucesso”
  + 2 – 5 – 8 – 19
    - Caso de teste: valor de desconto válido e radioDébito selecionado
    - Mensagem: “Pagamento efetuado com sucesso”
  + 2 – 5 – 8 – 11 – 19
    - Caso de teste: valor de desconto válido e radioCrédito selecionado
    - Mensagem: “Pagamento efetuado com sucesso”
  + 2 – 5 – 8 – 11 – 14
    - Caso de teste: valor de desconto válido e nenhum radioButton selecionado
    - Mensagem: “Selecione um tipo de Pagamento”
* **João Pedro Balestra Zacarkin**
  + **Funcionalidade: Finalizar Manipulação-** Funcionalidade do controller para a finalização da manipulação de compostos.
  + **Variáveis** **principais**: tamListComp(tamanho da lista de compostos para ser calculado o preço da mão-de-obra), salvar(variável em que é retornado o sucesso ou falha ao se salvar no banco de dados)
  + **Código**:

1. **public** Produto finalizarManipulacao(JFrame tela, List<Composto> listaCompostos, **double** total, String nome) {
3. Produto produto = **new** Produto();
4. produto.setDescricao(nome);
6. produto.setCompostos(listaCompostos);
8. //CALCULADNO A MÃƒO DE OBRA
9. **int** tamListComp = 0;
10. tamListComp = listaCompostos.size();
12. **if** (tamListComp <= 5) {
13. total = total + (total \* 0.10); //atÃ© 5 compostos: 10% de mÃ£o de obra
14. } **else** **if** (tamListComp <= 10) {
15. total = total + (total \* 0.15); //de 6 a 10 compostos: 15% de mÃ£o de obra
16. } **else** **if** (tamListComp > 10) {
17. total = total + (total \* 0.20); // mais de 10 compostos: 20% de mÃ£o de obra
18. }
19. produto.setPrecoVenda(total);
20. produto.setQuantidadeEstoque(1);
22. ProdutoDao produtoDao = **new** ProdutoDao();
23. String salvar = produtoDao.salvar(produto);
25. **switch** (salvar) {
26. **case** "SUCESSO":
27. JOptionPane.showMessageDialog(tela, "Compostos manipulados com sucesso!", "Sucesso", JOptionPane.DEFAULT\_OPTION);
28. tela.dispose();
29. **break**;
30. **default**:
31. JOptionPane.showMessageDialog(tela, "Tente novamente", "Algo deu errado", JOptionPane.DEFAULT\_OPTION);
32. **break**;
33. }
35. **return** produto;
37. }

* **Grafo de fluxo:**
* **Complexidade Ciclomática:**

**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