

Confecção de Um Sistema de Vendas em Java com Swing

Professor Hildeberto Melo - 2011

Índice

Sobre o autor	8
Introdução	9
Configurações	9
Tela principal do NetBeans.....	10
Criando um novo projeto	11
Definição do cenário.....	14
Codificando as Classes Básicas	16
Codificando a Classe Cliente.....	17
Codificando a Classe Produto	20
Codificando a Classe Item de Venda	22
Codificando a Classe Venda	23
Classe RepositorioCliente	27
Implementação do Construtor, métodos obterInstancia e listarTodos.....	27
Implementação do Método verificaExistencia	28
Implementação do Método Inserir Cliente	28
Implementação do Método Remover Cliente	29
Implementação do Método Atualizar Cliente	29
Classe RepositorioProduto	31
Implementação do Construtor e dos Métodos Listar Todos e obterInstancia.....	31
Implementação do Método Inserir Produto	32
Implementação do Método Remover Produto	32
Implementação do Método Atualizar Produto	33
Implementação do Método Verifica Existência.....	33
Classe RepositorioVenda	35
Implementação do Construtor e dos Métodos listarTodos e obterInstancia.....	35
Implementação do Método Inserir Venda	36

Implementação do Método Remover Venda	36
Implementação do Método Atualizar Venda	37
Implementação Método verificaExistência	37
Trabalhando com componentes Swing	39
Configurando a Classe Principal do Projeto.....	41
Manipulando Componentes Swing (básico).....	44
JTextField	44
JComboBox	45
JTable	47
Codificando a Parte Visual do Sistema de Vendas	50
Tela Principal do Sistema.....	50
Telas de Consultas	51
Tela de Consulta de Clientes.....	51
Tela de Consulta de Produtos.....	51
Tela de Consulta de Vendas.....	52
Telas de Cadastro e Alteração	53
Tela de Cadastro e Alteração de Cliente.....	53
Tela de Cadastro e Alteração de Produto.....	54
Tela de Cadastro de Venda.....	55
Código fonte das telas	57
Métodos auxiliares	57
Centralizar frame	57
Tela principal	57
Telas do Cliente	58
Consulta de Clientes	58
Cadastro de Clientes.....	58
Alteração de Cliente	59

Telas do Produto.....	59
Consulta de produtos	59
Cadastro de produtos	60
Alteração de produto.....	60
Telas do Venda	62
Consulta de vendas.....	62
Cadastro de Venda.....	62
Trabalhando com Banco de Dados	64
Estrutura do Banco De Dados.....	65
Configurando uma Conexão	66
Criando as Classes para Trabalhar com a Conexão Criada	68
Classe de Conexão	68
Classe de Repositório.....	69
Estrutura da Classe RepositorioClienteOdbc	70
Método remover	71
Método inserir	72
Método atualizar	73
Método listar todos	74
Modificando as Telas para Trabalhar Com Banco de Dados	75
Considerações Finais.....	75

Figuras

Figura 1 - Tela principal do NetBeans	10
Figura 2 - Criando novo projeto – parte 1	11
Figura 3 - Criando novo projeto – parte 2	12
Figura 4 - Criando novo projeto – parte 3	12
Figura 5 - Estrutura de pastas do projeto	13
Figura 6 - Diagrama de Classes de Domínio.....	14
Figura 7 - Diagrama com as classes referentes à manipulação dos dados do sistema	14
Figura 8 - Criando uma classe no NetBeans parte 1.....	16
Figura 9 - Criando uma classe no NetBeans parte 2.....	17
Figura 10 - Configurando a classe Cliente parte 1	18
Figura 11 - Configurando a classe Cliente parte 2	18
Figura 12 - Configurando a classe Cliente parte 3	19
Figura 13 - Configurando a classe Cliente parte 4	19
Figura 14 - Configurando a classe Cliente parte 5	20
Figura 15 - Configurando a classe Cliente parte 6	20
Figura 16 - Configurando a classe Produto parte 1	21
Figura 17 - Configurando a classe Produto parte 2	21
Figura 18 - Configurando a classe ItemVenda parte 1.....	22
Figura 19 - Configurando a classe ItemVenda parte 2.....	23
Figura 20 - Configurando a classe Venda parte 1	24
Figura 21 - Configurando a classe Venda parte 2	24
Figura 22 - Método para calcular o total da venda	25
Figura 23 – Classe RepositorioCliente	27
Figura 24 - Métodos listarTodos, obterIntancia e construtor do RepositórioCliente	28
Figura 25 - Método verificaExistencia Cliente	28
Figura 26 - Método inserir Cliente.....	29

Figura 27 - Método remover Cliente	29
Figura 28 - Método atualizar Cliente	30
Figura 29 - Estrutura da classe RepositórioProduto	31
Figura 30 - Métodos listarTodos, obterInstancia e construtor do RepositórioProduto	32
Figura 31 - Método inserir produto	32
Figura 32 - Método remover produto	33
Figura 33 - Método atualizar produto	33
Figura 34 - Método verifica existência produto	34
Figura 35 - Estrutura da classe RepositórioVenda	35
Figura 36 - Construtor e métodos listar todos e obterIntancia	36
Figura 37 - Método inserir venda	36
Figura 38 - Método remover venda	37
Figura 39 - Método atualizar venda	37
Figura 40 - Método verifica existência venda	38
Figura 41 - Tela principal do Sistema de Vendas	50
Figura 42 - Tela de Consulta de Clientes	51
Figura 43 - Tela de Consulta de Produtos	52
Figura 44 - Tela de Consulta de Vendas	53
Figura 45 - Tela de Cadastro de Cliente	54
Figura 46 - Tela de Alteração de Cliente	54
Figura 47 - Tela de Cadastro de Produto	55
Figura 48 - Tela de Alteração de Produto	55
Figura 49 - Tela de Cadastro de Venda	56
Figura 50 - Método centralizar frame	57
Figura 51 - Código Fonte tela principal	57
Figura 52 - Código fonte tela consulta de clientes	58
Figura 53 - Código fonte tela novo cliente	59

Figura 54 - Código fonte tela alterar cliente.....	59
Figura 55 - Código fonte tela consulta de produtos	60
Figura 56 - Código fonte tela novo produto	60
Figura 57 - Código fonte tela alterar produto	61
Figura 58 - Código fonte tela consulta de vendas 1.....	62
Figura 59 - Código fonte tela consulta de vendas 2	62
Figura 60 - Código fonte tela cadastro de vendas 1	63
Figura 61 - Código fonte tela cadastro de vendas 2	63
Figura 62 - Código fonte tela cadastro de vendas 3	63
Figura 63 - Código fonte tela cadastro de vendas 4	64
Figura 64 - Configurando a conexão com o banco de dados - 1.....	66
Figura 65 - Configurando a conexão com o banco de dados - 2.....	67
Figura 66 - Configurando a conexão com o banco de dados - 3.....	67
Figura 67 - Configurando a conexão com o banco de dados - 4.....	68
Figura 68 - Classe de Conexão com o banco de dados.....	69
Figura 69 - Estrutura da classe RepositorioClienteOdbc	70
Figura 70 - Método remover do RepositorioClienteOdbc.....	71
Figura 71 - Método inserir do RepositorioClienteOdbc	72
Figura 72 - Método atualizar do RepositorioClienteOdbc.....	73
Figura 73 - Método listarTodos do RepositorioClienteOdbc.....	74
Figura 74 - Mudança na GUI para utilizar banco de dados.....	75

Sobre o autor

O professor Hildeberto Melo atua nas áreas acadêmica, desenvolvimento e consultorias em sistemas de informação, desenvolvimento e consultorias em banco de dados.

Formação acadêmica:

- Técnico em Desenvolvimento de Sistemas – Ibratec Instituto Brasileiro de Tecnologia;
- Bacharel em Sistemas de Informação – FIR Faculdade Integrada do Recife;
- Especialista em Docência no Ensino Superior – FMN Faculdade Maurício de Nassau;
- Mestre em Ciência da Computação – UFPE Universidade Federal de Pernambuco.

Disciplinas Ministradas:

- Banco de Dados,
- Projeto de Banco de Dados,
- Desenvolvimento WEB,
- Estrutura de Dados,
- Programação Orientada a Objetos,
- Desenvolvimento de Aplicações Desktop,
- Programação Cliente Servidor,
- Sistemas Distribuídos,
- Sistemas Operacionais,
- Análise de Projetos de Sistemas Orientado a Objetos,
- Linguagens: Java, C#, C, Pascal, Delphi, PHP, ASP.
- Lógica Matemática
- Lógica de Programação
- Informática Básica
- Tecnologia da Informação e Sociedade

Introdução

Neste documento veremos o passo a passo para o desenvolvimento de um sistema básico de gerenciamento de vendas. Foram aplicadas apenas as validações necessárias para confeccionar este sistema.

Configurações

Para a confecção deste aplicativo utilizaremos a IDE NetBeans.

Vamos começar com a instalação do JDK (Java Development Kit), é necessário ter instalado o JDK no computador para podermos começar a trabalhar com desenvolvimento em Java.

- Baixar o JDK
- Instalar o JDK no computador

Baixar o NetBeans do site www.netbeans.org. Após baixar o NetBeans, efetue sua instalação.

Tela principal do NetBeans

Ao ser iniciado o NetBeans apresenta sua interface de desenvolvimento, mostrado na Figura 1. Na parte superior é apresentada à barra de menus. No canto esquerdo temos a guia dos projetos e seus respectivos arquivos (Project Explore). Na parte central serão mostrados os arquivo em utilização pelo programador. Na parte inferior, informações sobre servidores, propriedades, console, etc. Na parte esquerda as propriedades dos arquivos do projeto. Conforme mostra Figura 1.

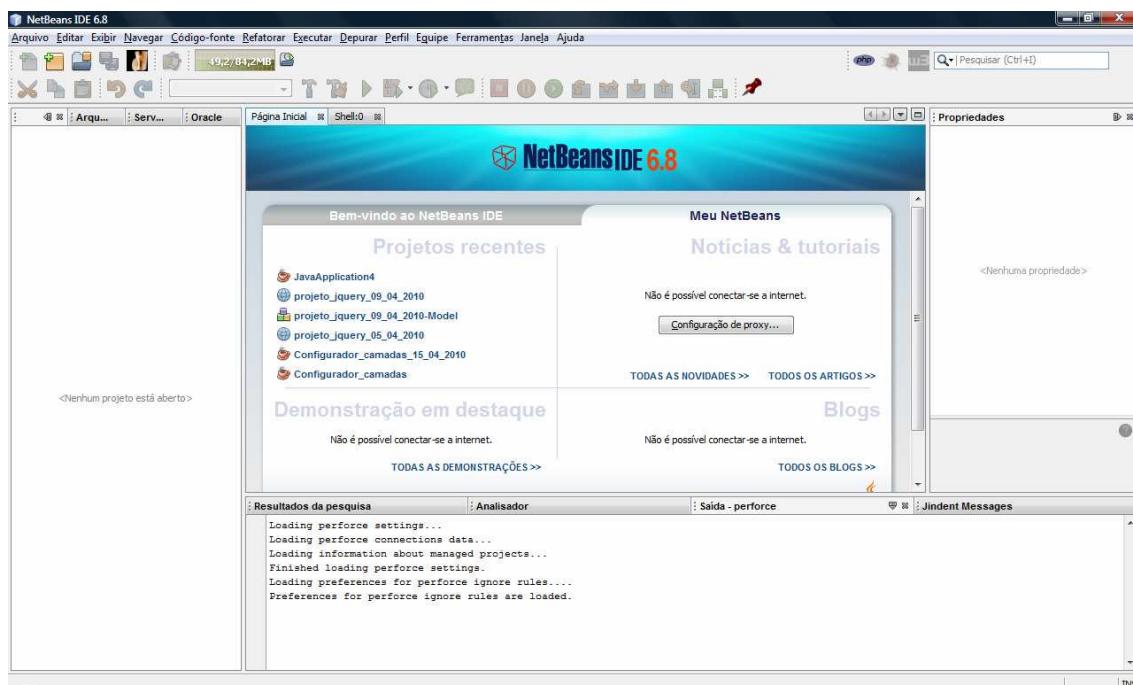


Figura 1 - Tela principal do NetBeans

Criando um novo projeto

Para criarmos uma aplicação Swing, é simples, basta acessar o menu: **arquivo->novo projeto**, conforme mostra a Figura 2.

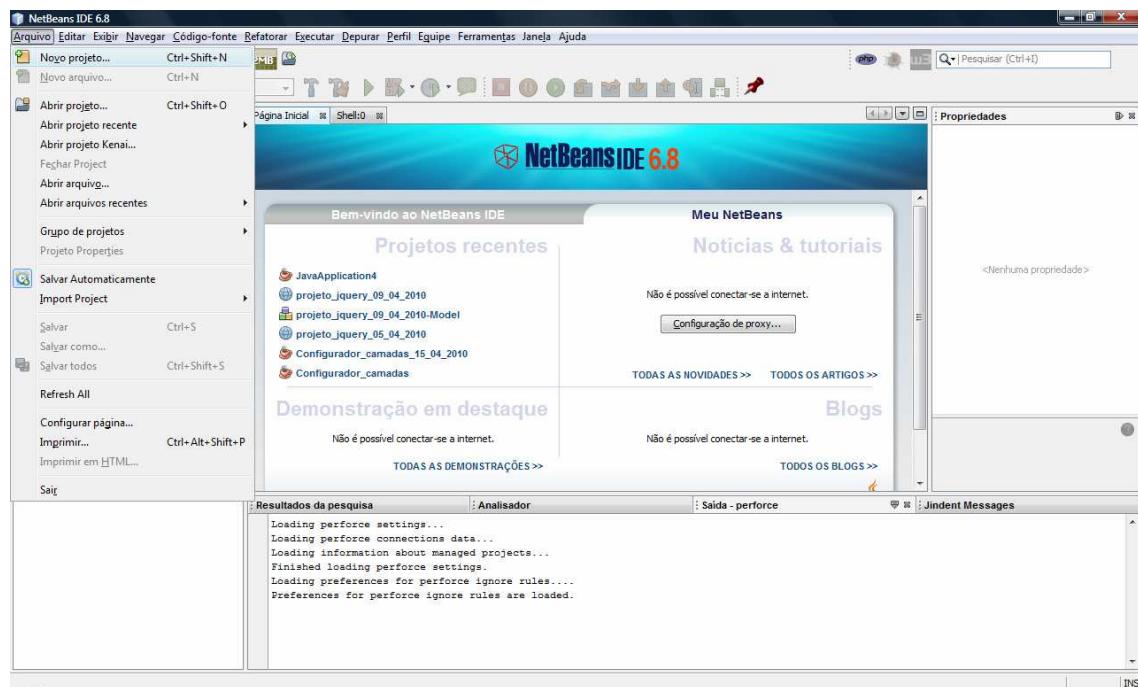


Figura 2 - Criando novo projeto – parte 1

Aparecerá à tela de **Escolha do tipo de projeto**, iremos escolher a categoria **Java** e o projeto **Aplicativo Java**. Após selecionar as opções ditas anteriormente clique no botão **Próximo**. Conforme mostra a Figura 3.

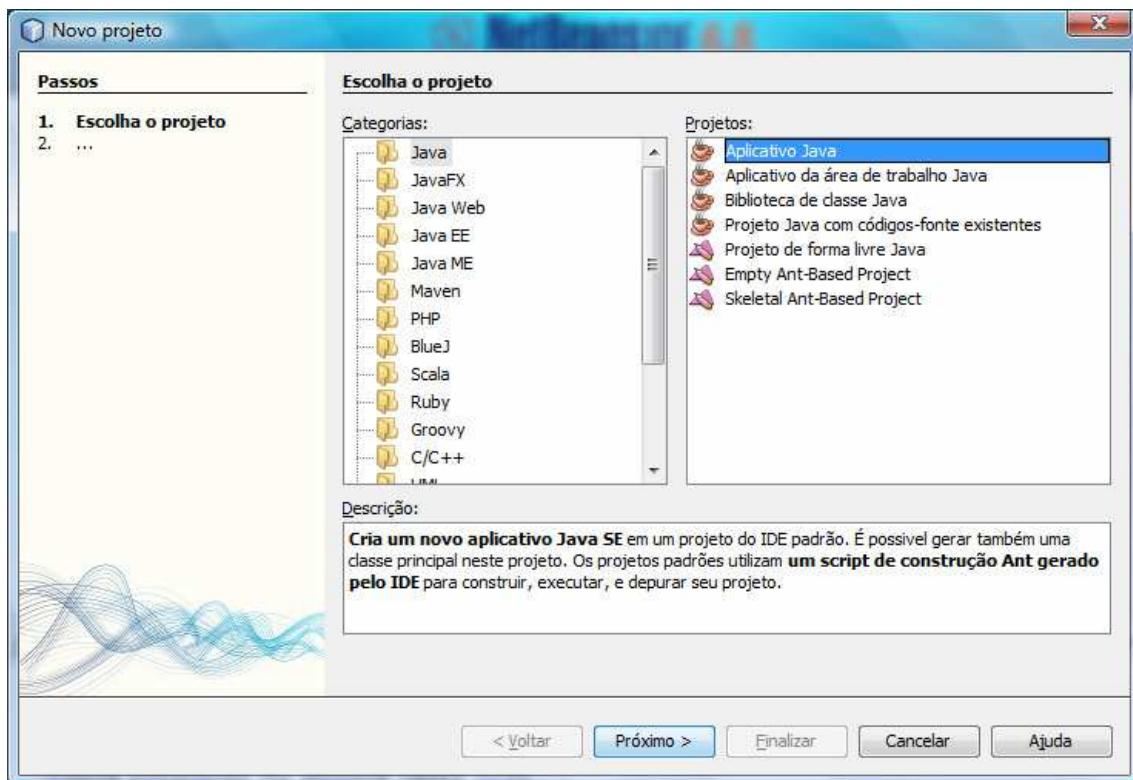


Figura 3 - Criando novo projeto – parte 2

Aparecerá a tela de configuração do Projeto, onde constam as seguintes informações do Nome do projeto e Localização do projeto. Após colocar as informações solicitadas clicar no botão Finalizar. Conforme mostra Figura 4.

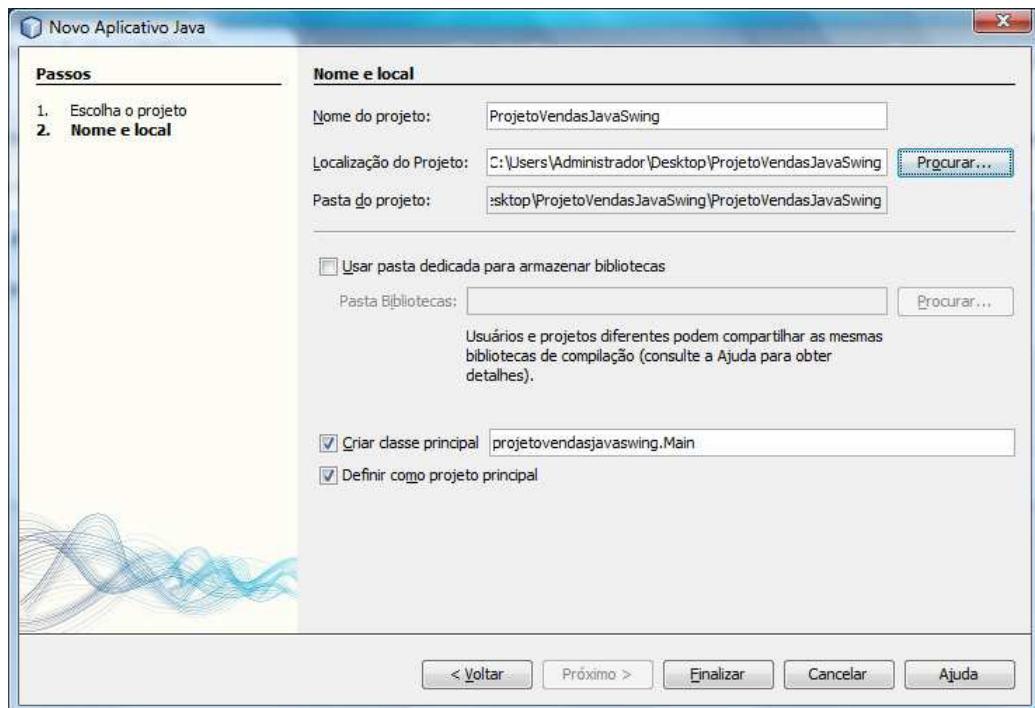


Figura 4 - Criando novo projeto – parte 3

Aparecerá na aba de projetos o projeto recém criado, contendo 5 (cinco) pastas: Pacotes de código-fonte, pacote de testes, Load Generator Scripts, Bibliotecas e Bibliotecas de testes.

Na pasta de Pacote de código-fonte serão colocados os arquivos Java que farão parte do projeto, dentro desta pasta foi criado automaticamente um pacote com o nome correspondente ao nome do projeto. Conforme mostra Figura 5

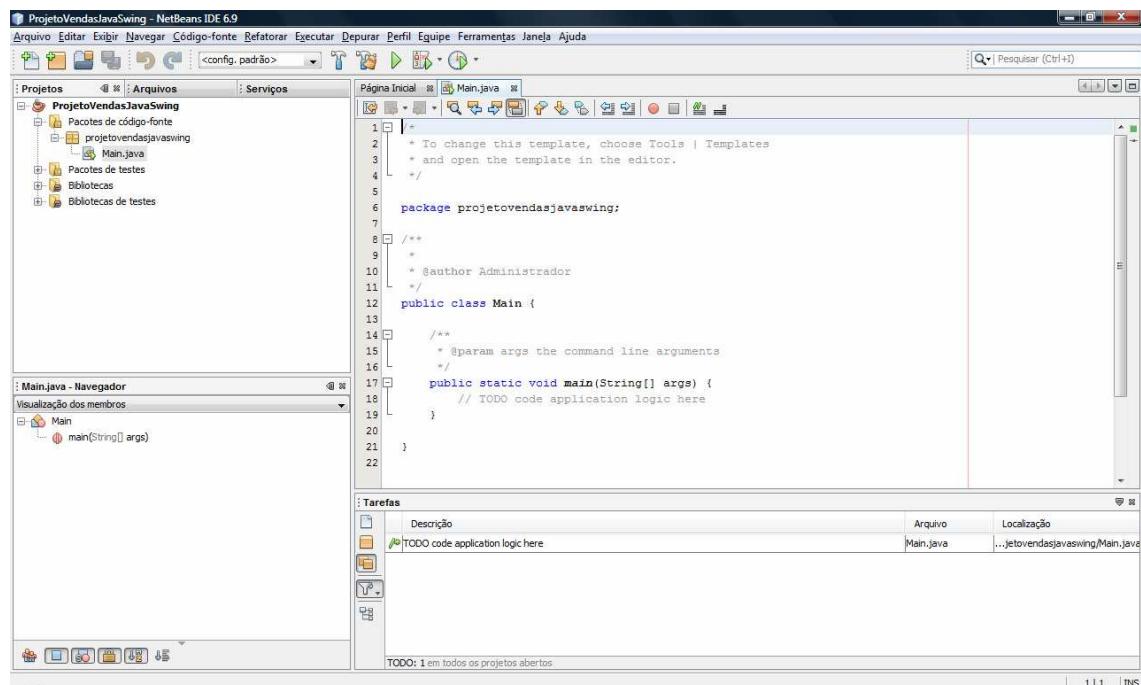


Figura 5 - Estrutura de pastas do projeto

Definição do cenário

O sistema de vendas a ser desenvolvido possui as seguintes entidades: clientes, produtos e as vendas com seus respectivos itens. Tomaremos por base o diagrama abaixo, no qual representará o nosso sistema.

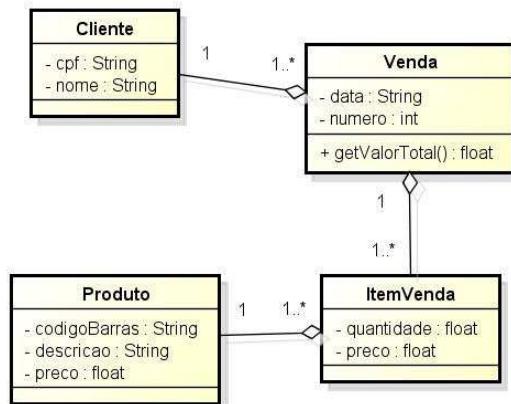


Figura 6 - Diagrama de Classes de Domínio

Para manipularmos as informações do sistema de vendas é necessário incluirmos algumas entidades ao diagrama inicial. Estas entidades têm como objetivo principal implementar as rotinas necessárias para a manipulação das informações do sistema.

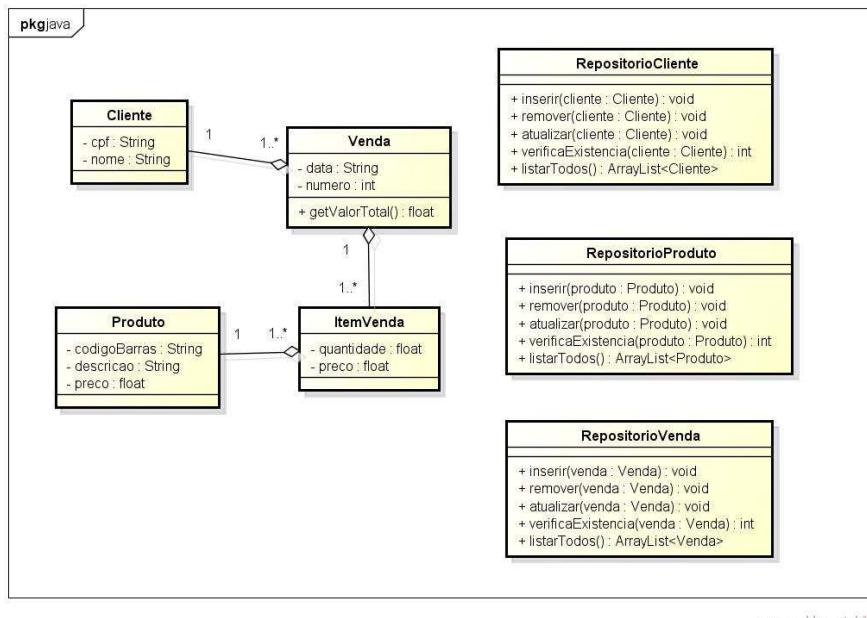


Figura 7 - Diagrama com as classes referentes à manipulação dos dados do sistema

Como podemos observar na Figura acima as classes de repositórios têm os seguintes métodos:

- **Inserir** – utilizado para inserir um determinado objeto no dispositivo de armazenamento.
- **remover** – utilizado para remover um determinado objeto, no qual este objeto deve estar previamente cadastrado no dispositivo de armazenamento.
- **atualizar** – utilizado para atualizar os valores contidos em um objeto previamente cadastrado no dispositivo de armazenamento.
- **verifica existência** – utilizado para verificar se as informações de um determinado objeto estão presentes no conjunto de objetos cadastrados no dispositivo de armazenamento. Este método será utilizado para:
 - Na inclusão: evitar o cadastro de dois objetos com as mesmas informações;
 - Na exclusão: Verificar se um determinado objeto está devidamente cadastrado, para posteriormente removê-lo.
 - Na alteração: Verificar se um determinado objeto está devidamente cadastrado, para posteriormente alterá-lo.
- **Listar** – utilizado para retornar todos os objetos que foram cadastrados no dispositivo de armazenamento. Obs.: as informações contidas em cada objeto deverão ser exibidas conforme o desenvolvedor da aplicação preferir, ex: apresentar as informações em um JComboBox, em uma jTable, etc.

Codificando as Classes Básicas

As classes básicas são **Cliente**, **Produto**, **ItemVenda** e **Venda**, estas classes representam o domínio da aplicação. Cada classe deverá ser um arquivo “. Java” onde cada arquivo deverá possuir todos os atributos e métodos correspondentes.

Para adicionar uma nova Classe ao projeto, deveremos seguir os seguintes passos: clicar com botão direito do mouse no pacote chamado “**projetovendasjavaswing**”, em seguida escolher a opção “**novo**” e depois a opção “**Class Java**”.

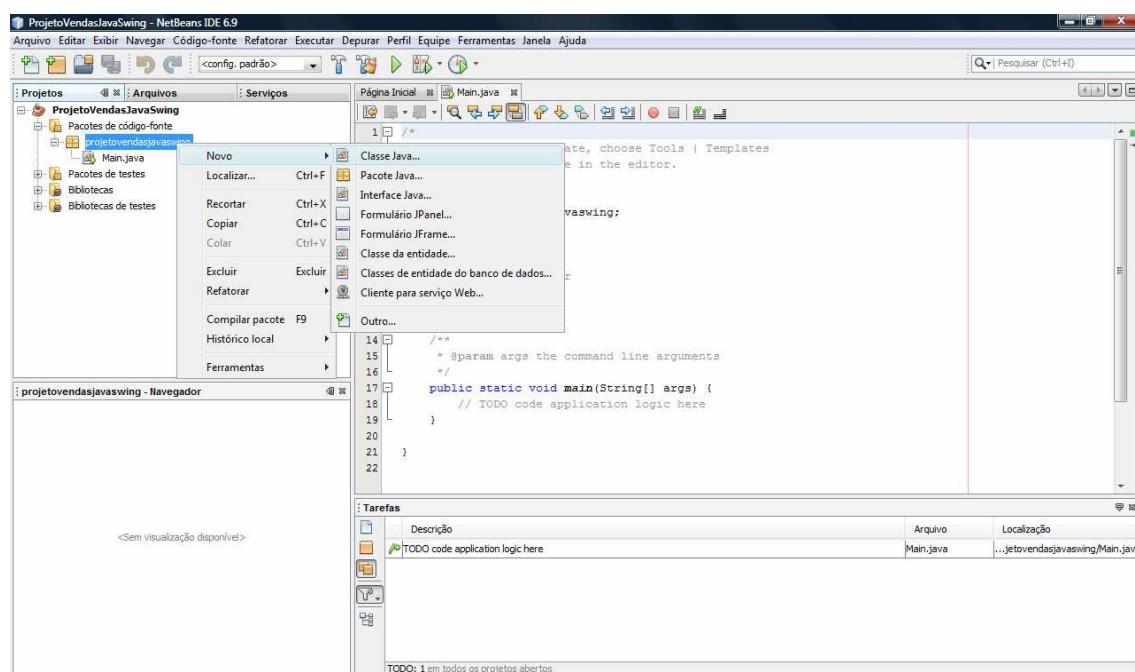


Figura 8 - Criando uma classe no NetBeans parte 1

Em seguida aparecerá à tela para configuração da Classe, neste caso informaremos o nome da Classe no qual codificaremos. Esta seqüência de passos deverá ser seguida para todas as classes que farão parte do sistema.

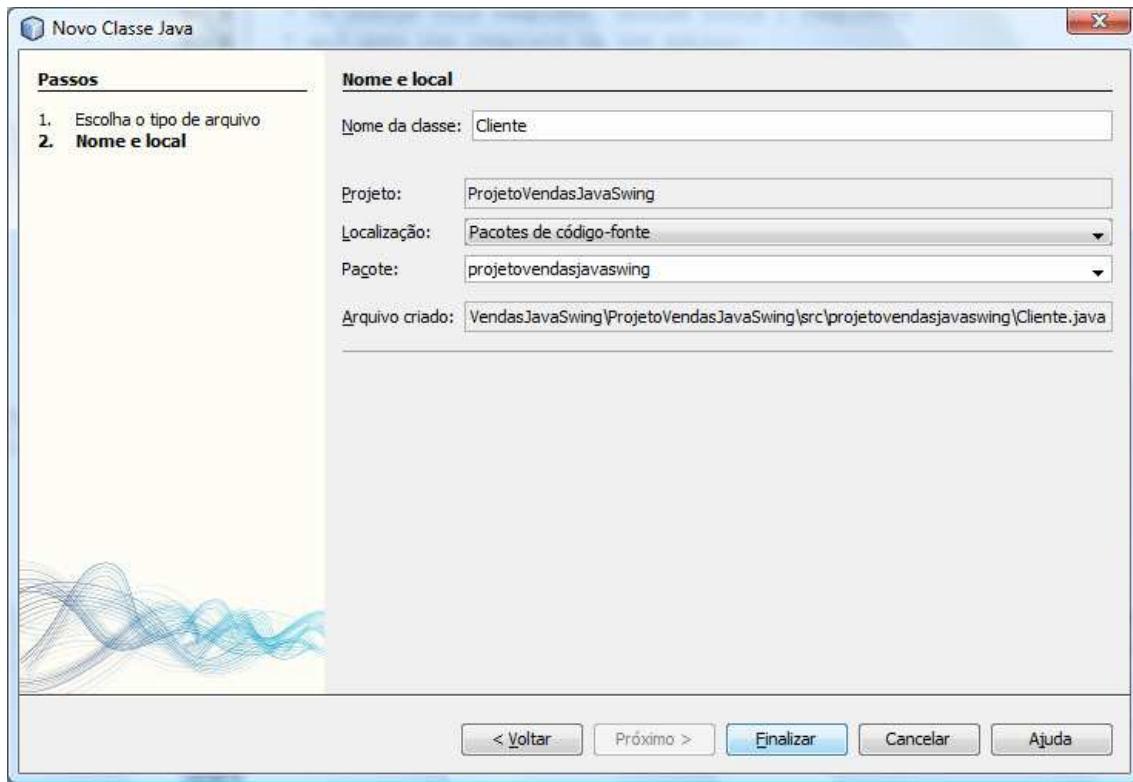


Figura 9 - Criando uma classe no NetBeans parte 2

Codificando a Classe Cliente

Para adicionarmos a Classe Cliente ao sistema, é necessário seguir os passos descritos acima, aparecerá uma tela de configuração inicial da Classe, nesta tela basta informar o nome da Classe, que neste caso, deverá se chamar “**Cliente**”, e clicar no botão “**Finalizar**”.

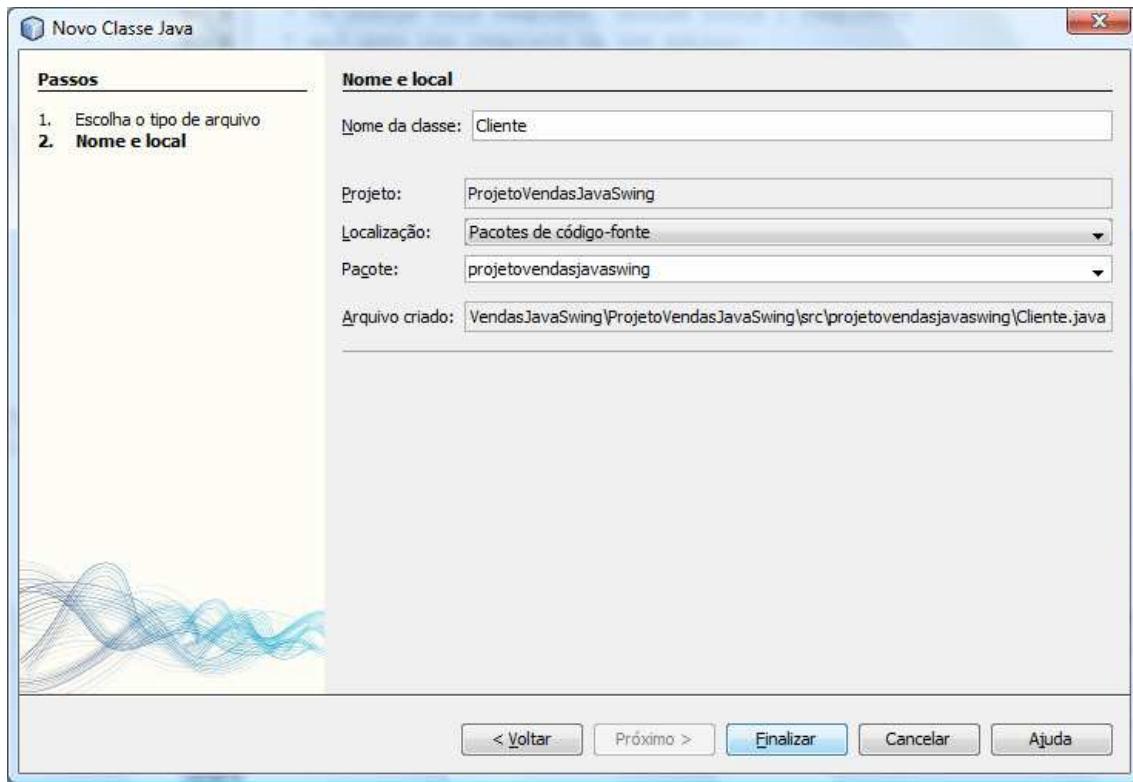


Figura 10 - Configurando a classe Cliente parte 1

Aparecerá a estrutura básica de uma classe em Java, que terá apenas o qualificador de acesso “**public**”, a palavra reservada “**class**” que define o tipo de arquivo e nome da classe “**Cliente**”.

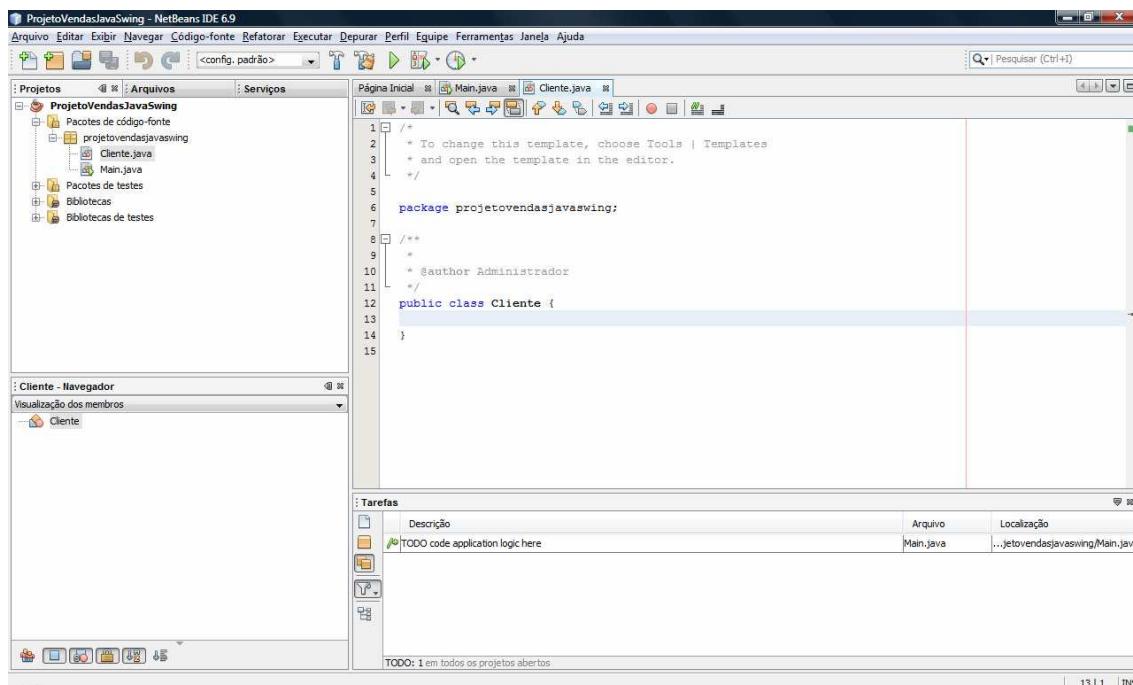


Figura 11 - Configurando a classe Cliente parte 2

Codificaremos os atributos da classe **Cliente**, estes atributos foram definidos no diagrama de classe, neste caso os atributos **nome** e **cpf**, ambos do tipo **String**.

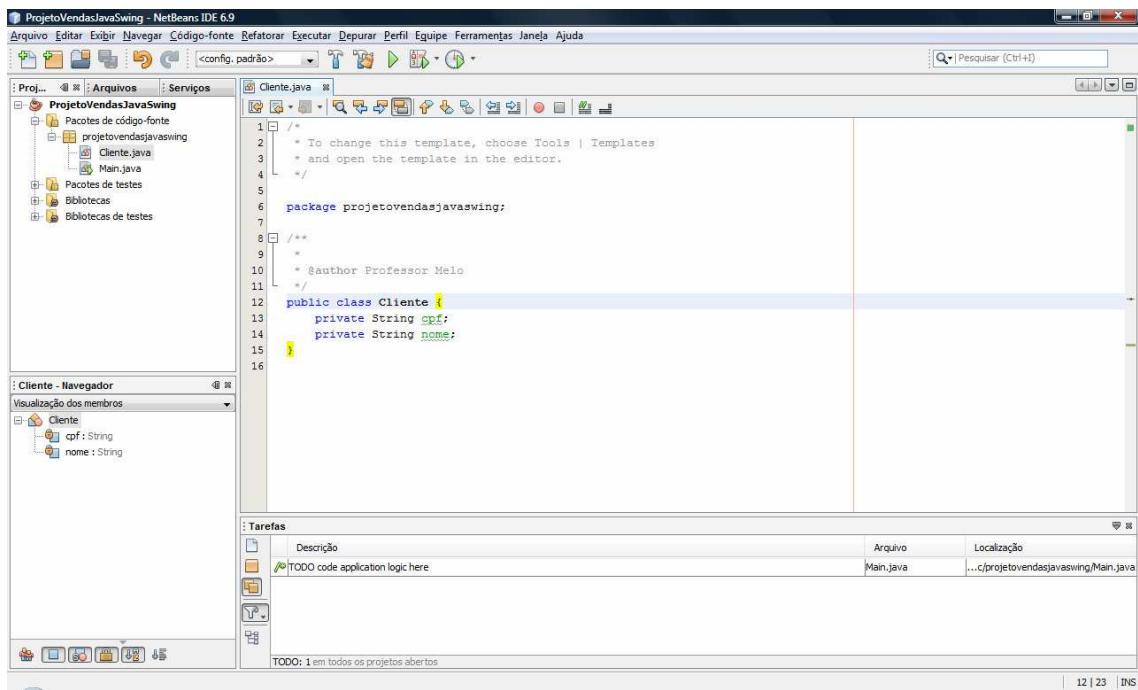


Figura 12 - Configurando a classe Cliente parte 3

Para encapsular automaticamente os atributos da classe Cliente, seguiremos os seguintes passos: Clicar com o botão direito do mouse em cima de qualquer atributo, depois escolher a opção “refatorar”, em seguida “encapsular campos”, conforme ilustra figura abaixo.

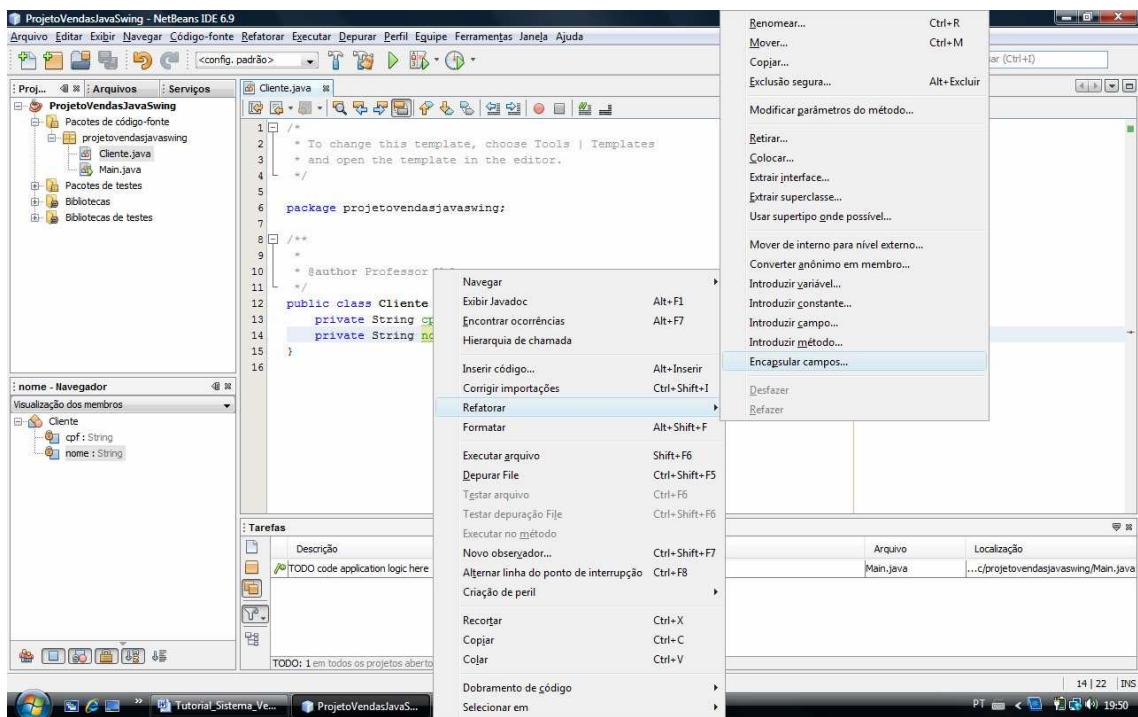


Figura 13 - Configurando a classe Cliente parte 4

Aparecerá a tela de configuração dos “gets” e “sets” correspondentes ao referidos atributos da classe. Clicar no botão “**Selecionar tudo**”, em seguida clicar no botão “**Refatorar**”

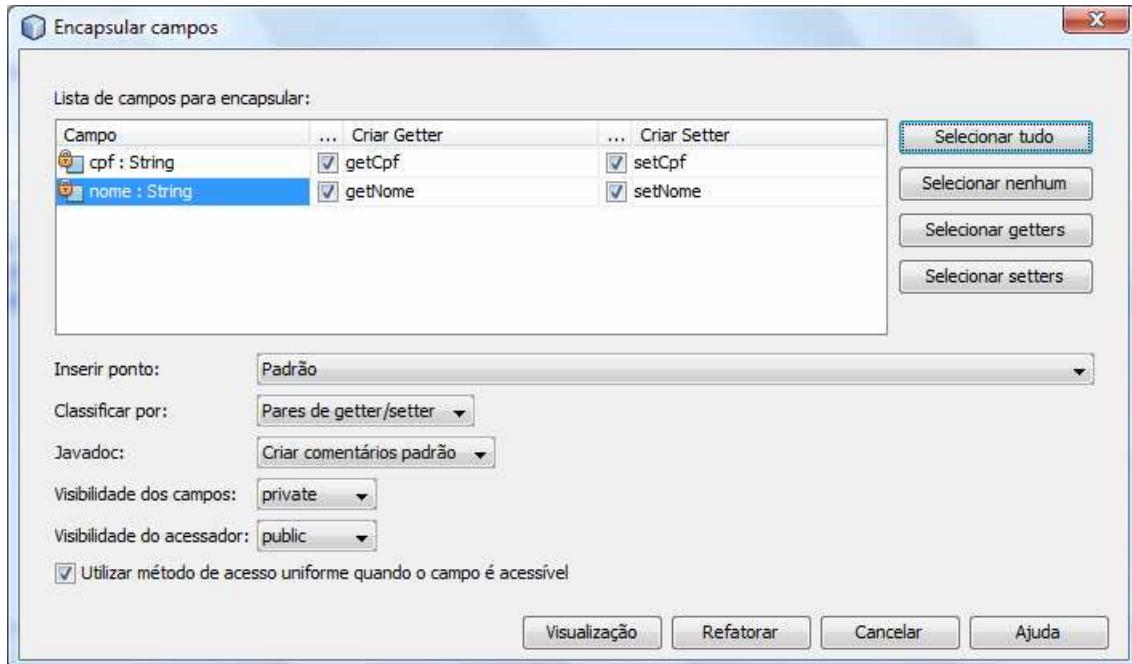


Figura 14 - Configurando a classe Cliente parte 5

Seguindo as etapas citadas acima, foi gerado automaticamente os “gets” e “sets” conforme ilustra a figura abaixo.

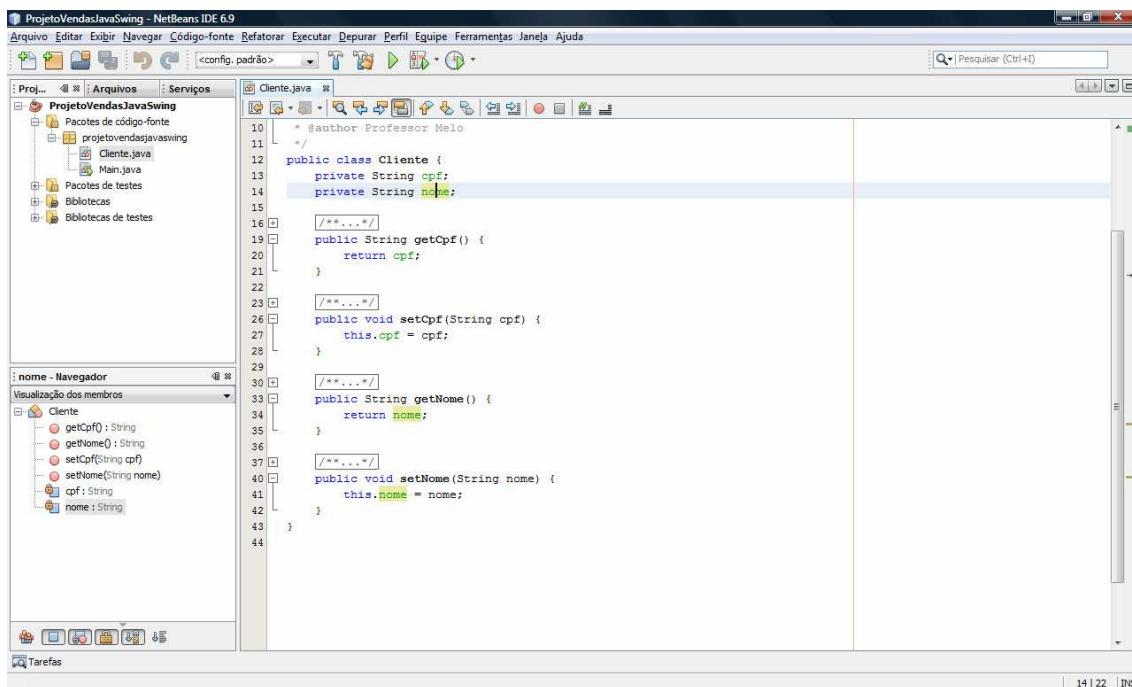


Figura 15 - Configurando a classe Cliente parte 6

Codificando a Classe Produto

Para adicionarmos a Classe **Produto** ao sistema, basta seguir os passos descritos para a classe **Cliente**. Vale salientar que deveremos efetuar as configurações referentes ao **Produto**.

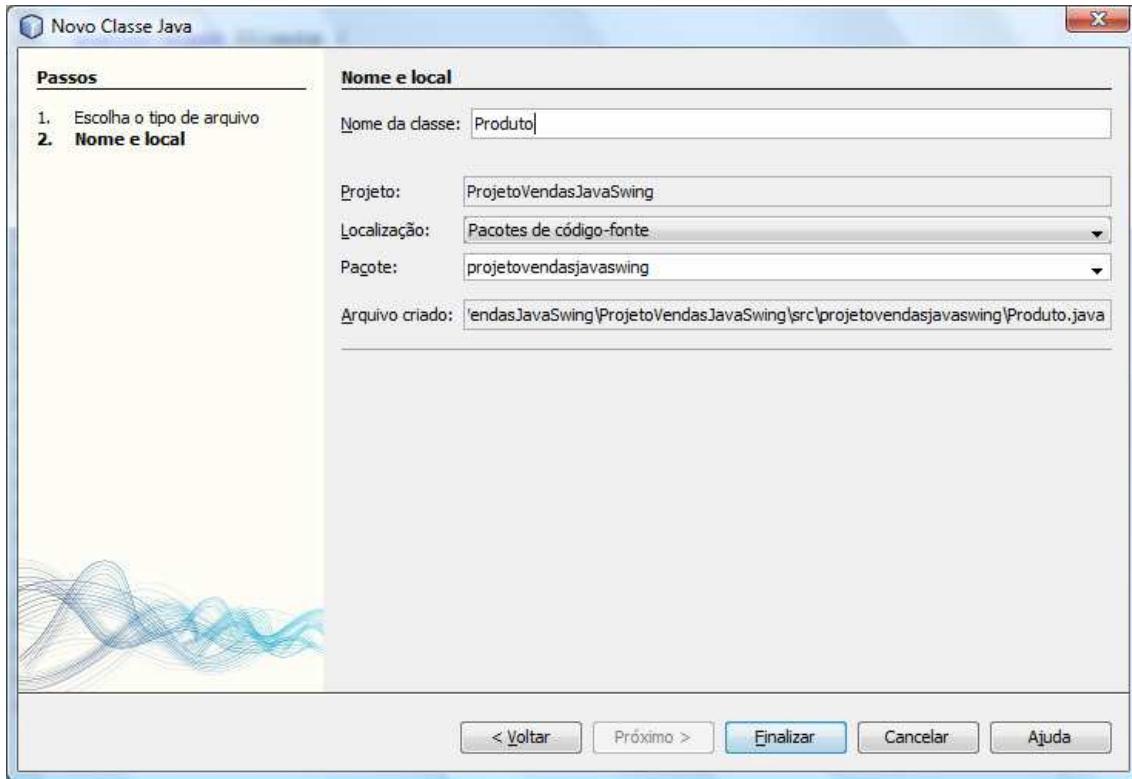


Figura 16 - Configurando a classe Produto parte 1

Depois de codificar os atributos referentes ao Produto e gerar os “gets” e “sets”, a classe Produto ficará assim.

Figura 17 - Configurando a classe Produto parte 2

Codificando a Classe Item de Venda

Para adicionarmos a Classe **ItemVenda** ao sistema, basta seguir os passos descritos para a classe **Cliente**. Vale salientar que deveremos efetuar as configurações referentes ao **ItemVenda**.

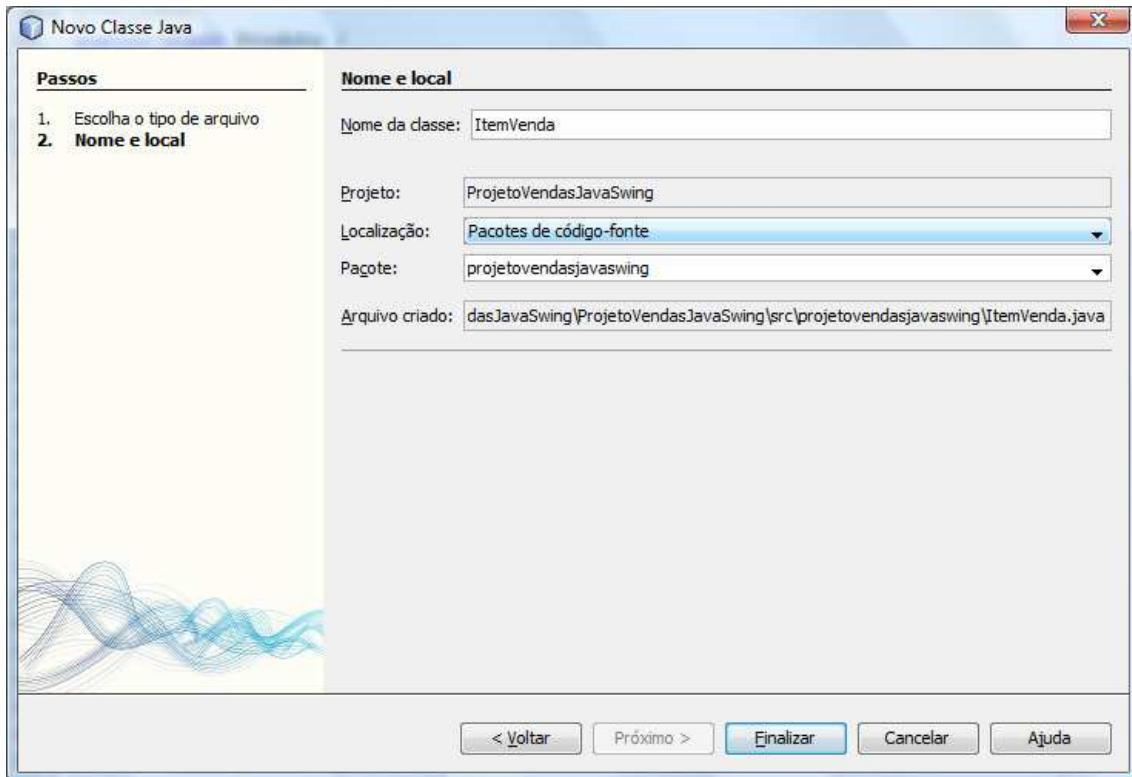


Figura 18 - Configurando a classe ItemVenda parte 1

Além dos atributos quantidade e preço, pode-se observar que a classe **ItemVenda** tem um relacionamento de **agregação** com a classe **Produto**. No diagrama de classes a relação está definida da seguinte forma: Um item de venda possui um Produto. Neste caso o **ItemVenda** terá um atributo do tipo **Produto**.

```

7  /**
8  * 
9  * @author professor Melo
10 */
11 public class ItemVenda {
12     private float quantidade;
13     private float preco;
14     //implementação da agregação do Produto
15     //como um item de venda possui apenas um produto
16     //então o Item de venda terá um atributo do tipo Produto
17     private Produto produto;
18
19     public ItemVenda() {
20         this.produto = new Produto();
21     }
22     /**
23      * @return the quantidade
24     */
25     public float getQuantidade() {...}
26     /**
27     */
28     public void setQuantidade(float quantidade) {...}
29     /**
30     */
31     public float getPreco() {...}
32     /**
33     */
34     public void setPreco(float preco) {...}
35     /**
36     */
37     public Produto getProduto() {...}
38     /**
39     */
40     public void setProduto(Produto produto) {...}
41
42 }

```

Figura 19 - Configurando a classe ItemVenda parte 2

Foi necessário codificar um **construtor** para a classe **ItemVenda**, no construtor será instanciado automaticamente o atributo **Produto**, isto se faz necessário para evitar possíveis erros no manuseio do **ItemVenda**.

Codificando a Classe Venda

Para adicionarmos a Classe **Venda** ao sistema, basta seguir os passos descritos para a classe **Cliente**. Vale salientar que deveremos efetuar as configurações referentes à **Venda**.

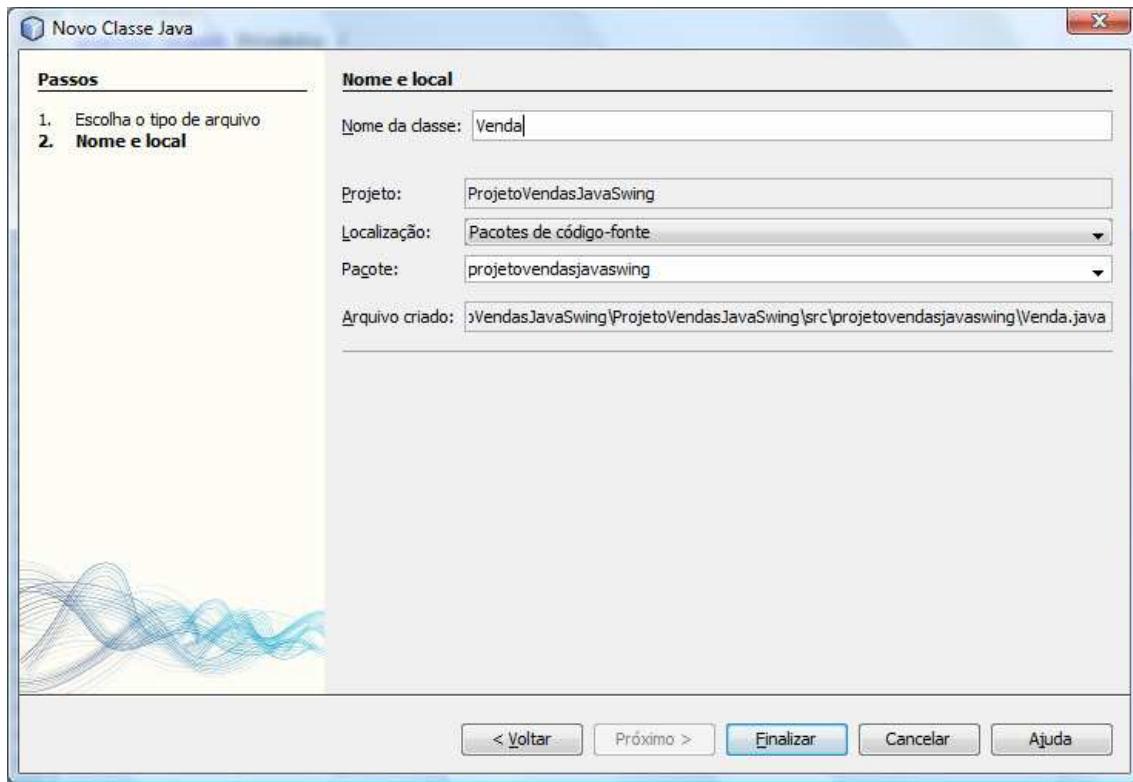


Figura 20 - Configurando a classe Venda parte 1

Além dos atributos número e data, pode-se observar que a classe **Venda** tem um relacionamento de **agregação** com a classe **ItemVenda** e a classe **Cliente**. No diagrama de classes a relação está definida da seguinte forma: Uma **Venda** possui um conjunto de **ItemVenda**. Neste caso a **Venda** terá um atributo do **ArrayList<ItemVenda>**. E para o relacionamento de agregação com o **Cliente** a **Venda** terá um atributo do tipo **Cliente**, seguindo a mesma situação do relacionamento entre o **ItemVenda** e **Produto**.

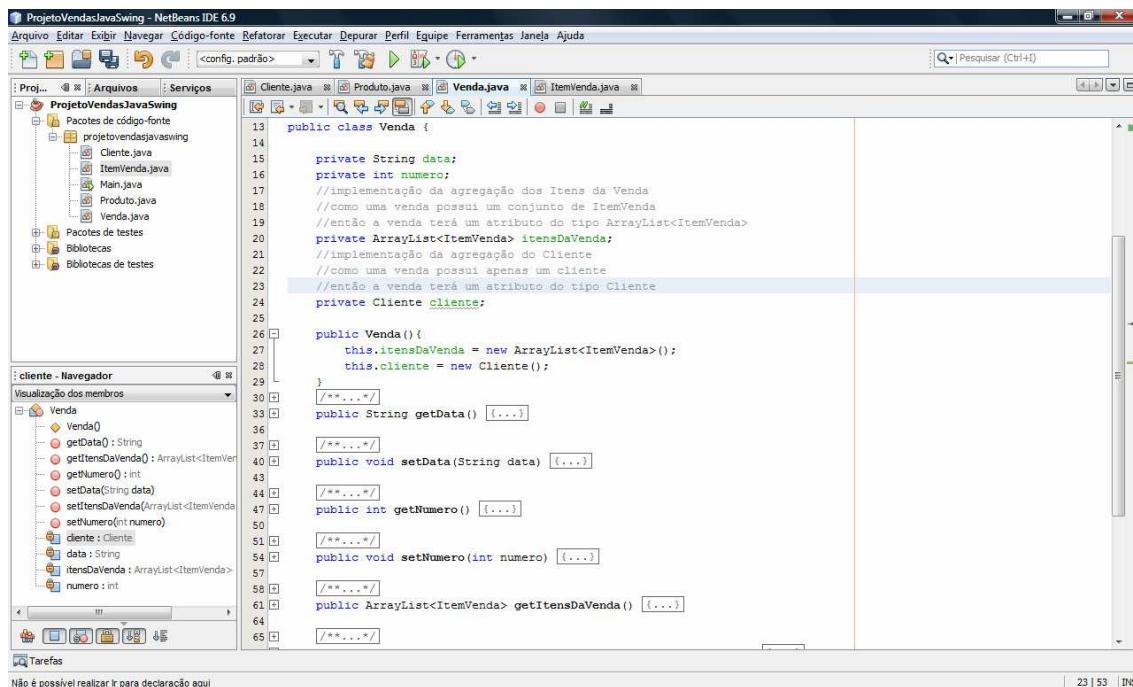


Figura 21 - Configurando a classe Venda parte 2

Foi necessário codificar um **construtor** para a classe **Venda**, no construtor será instanciado automaticamente o atributo `itensDaVenda` (do tipo `ArrayList<ItemVenda>`) e atributo `cliente`, isto se faz necessário para evitar possíveis erros no manuseio da **Venda**.

A classe `Venda` possui um método responsável por calcular o total geral da Venda. A lógica utilizada foi percorrer todos os itens pertencentes à `Venda` e acumular o cálculo da (`quantidade * preço`) de cada item.

```
public float getValorTotal(){
    float retorno = 0;
    for (int i = 0; i < this.itensDaVenda.size(); i++) {
        float valor = (this.itensDaVenda.get(i).getQuantidade() * this.itensDaVenda.get(i).getPreco());
        retorno = retorno + valor;
    }
    return retorno;
}
```

Figura 22 - Método para calcular o total da venda

Codificando as Classes de Repositórios

Serão codificadas três classes de repositórios, `RepositorioCliente`, `RepositorioProduto` e `RepositorioVenda`. Cada classe corresponderá aos clientes, produtos e vendas com seus respectivos itens.

Utilizaremos três `ArrayList` para armazenar as informações referentes aos clientes, produtos e venda. Em cada método será implementada as validações básicas referentes a cada rotina.

Antes de iniciar a codificação destas classes, vamos revisar algumas coisas sobre o `ArrayList`. O `ArrayList` pertence ao pacote `java.util.ArrayList` e consiste numa classe capaz de armazenar uma coleção de objetos. A manipulação dos objetos contidos no mesmo é dinâmica e pode ser uma lista de quaisquer objetos, ou uma lista de objetos específicos. Alguns métodos:

- `Add(Object)`: adiciona um objeto ao `ArrayList`
- `Clear()`: remove todos os objetos do `ArrayList`
- `Get(int)`: retorna o objeto do index informado
- `isEmpty()`: retorna se o `ArrayList` está vazio
- `Remove(int)`: remove o objeto do index informado
- `Size()`: retorna a quantidade de objetos contidos no `ArrayList`

Relembrando os Métodos a serem implementados em cada repositório:

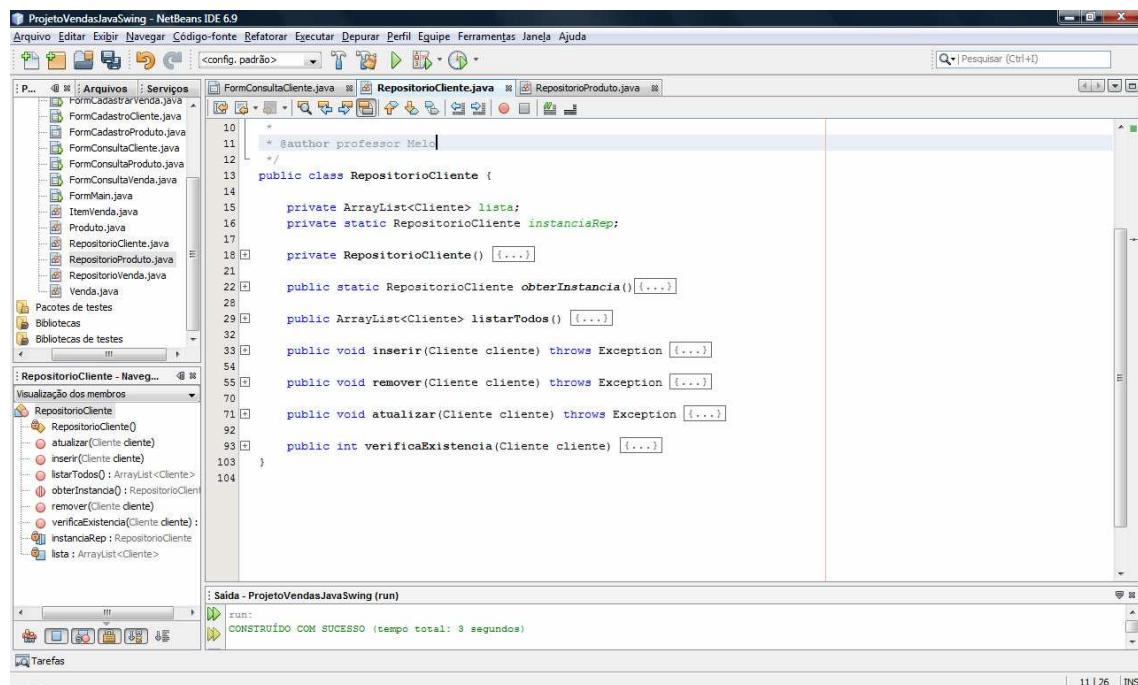
- **Inserir** – utilizado para inserir um determinado objeto no dispositivo de armazenamento.
- **remover** – utilizado para remover um determinado objeto, no qual este objeto deve estar previamente cadastrado no dispositivo de armazenamento.
- **atualizar** – utilizado para atualizar os valores contidos em um objeto previamente cadastrado no dispositivo de armazenamento.
- **verifica existência** – utilizado para verificar se as informações de um determinado objeto estão presentes no conjunto de objetos cadastrados no dispositivo de armazenamento. Este método será utilizado para:
 - Na inclusão: evitar o cadastro de dois objetos com as mesmas informações;
 - Na exclusão: Verificar se um determinado objeto está devidamente cadastrado, para posteriormente removê-lo.
 - Na alteração: Verificar se um determinado objeto está devidamente cadastrado, para posteriormente alterá-lo.

- **Listar** – utilizado para retornar todos os objetos que foram cadastrados no dispositivo de armazenamento. Obs.: as informações contidas em cada objeto deverão ser exibidas conforme o desenvolvedor da aplicação preferir, ex: apresentar as informações em um JComboBox, em uma jTable, etc.

Classe RepositorioCliente

A figura abaixo ilustra a classe RepositorioCliente, podemos observar que foi declarado um atributo do tipo ArrayList<Cliente>, este atributo tem por finalidade armazenar todos os clientes cadastrado pelo sistema. Também foi declarado um atributo do tipo RepositorioCliente, este atributo tem por finalidade implementar este repositório com o padrão **singleton**.

Também podemos observar a assinatura dos métodos definidos no diagrama de classes.



```

 10  /*
 11  * @author professor Melo
 12  */
 13  public class RepositorioCliente {
 14
 15      private ArrayList<Cliente> lista;
 16      private static RepositorioCliente instanciaRep;
 17
 18      private RepositorioCliente() { ... }
 19
 20      public static RepositorioCliente obterInstancia() { ... }
 21
 22      public ArrayList<Cliente> listarTodos() { ... }
 23
 24      public void inserir(Cliente cliente) throws Exception { ... }
 25
 26      public void remover(Cliente cliente) throws Exception { ... }
 27
 28      public void atualizar(Cliente cliente) throws Exception { ... }
 29
 30      public int verificaExistencia(Cliente cliente) { ... }
 31
 32  }
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104

```

Figura 23 – Classe RepositorioCliente

Implementação do Construtor, métodos obterIntancia e listarTodos

```

public class RepositorioCliente {
    private ArrayList<Cliente> lista;
    private static RepositorioCliente instanciaRep;

    private RepositorioCliente() {
        this.lista = new ArrayList<Cliente>();
    }

    public static RepositorioCliente obterInstancia(){
        if(instanciaRep == null){
            instanciaRep = new RepositorioCliente();
        }
        return instanciaRep;
    }

    public ArrayList<Cliente> listarTodos() {
        return this.lista;
    }

    ...
}

```

Figura 24 - Métodos listarTodos, obterIntancia e construtor do RepositórioCliente

Implementação do Método verificaExistencia

Este método tem como objetivo pesquisar um objeto pelo seu CPF e retornar o índice deste objeto, caso não seja encontrado nenhum objeto no qual corresponda ao critério de pesquisa, o método retornará -1.

```

public int verificaExistencia(Cliente cliente) {
    int retorno = -1;
    for (int i = 0; i < this.lista.size(); i++) {
        if (cliente.getCpf().trim().equals(this.lista.get(i).getCpf().trim())) {
            retorno = i;
            break;
        }
    }
    return retorno;
}

```

Figura 25 - Método verificaExistencia Cliente

Implementação do Método Inserir Cliente

The screenshot shows the NetBeans IDE interface with the project 'ProjetoVendasJavaSwing' open. The code editor displays the 'Cliente.java' file, specifically the implementation of the 'inserir' method. The code checks for null values and empty CPF fields before adding the client to a list. The 'listarTodos' and 'remover' methods are also partially visible.

```

20
21     public ArrayList<Cliente> listarTodos() {
22         return this.lista;
23     }
24
25     public void inserir(Cliente cliente) throws Exception {
26         if (cliente == null) {
27             throw new Exception("O cliente não foi instanciado");
28         }
29         if (cliente.getCpf() == null) {
30             throw new Exception("Informar o CPF do cliente");
31         }
32         if (cliente.getCpf().trim().equals("")) {
33             throw new Exception("Informar o CPF do cliente");
34         }
35         if (cliente.getNome() == null) {
36             throw new Exception("Informar o Nome do cliente");
37         }
38         if (cliente.getNome().trim().equals("")) {
39             throw new Exception("Informar o Nome do cliente");
40         }
41         if (this.verificaExistencia(cliente) >= 0) {
42             throw new Exception("O referido cliente já se encontra cadastrado");
43         }
44         this.lista.add(cliente);
45     }
46
47     public void remover(Cliente cliente) throws Exception {
48         if (cliente == null) {
49             throw new Exception("O cliente não foi instanciado");
50         }
51         if (cliente.getCpf() == null) {
52             throw new Exception("Informar o CPF do cliente");
53         }
54         if (cliente.getNome().trim().equals("")) {
55             throw new Exception("Informar o Nome do cliente");
56         }
57         if (this.verificaExistencia(cliente) >= 0) {
58             throw new Exception("O referido cliente já se encontra cadastrado");
59         }
60         this.lista.remove(this.verificaExistencia(cliente));
61     }
62
63     public void atualizar(Cliente cliente) throws Exception {
64         if (cliente == null) {
65             throw new Exception("O cliente não foi instanciado");
66         }
67         if (cliente.getCpf() == null) {
68             throw new Exception("Informar o CPF do cliente");
69         }
70         if (cliente.getCpf().trim().equals("")) {

```

Figura 26 - Método inserir Cliente

Implementação do Método Remover Cliente

The screenshot shows the NetBeans IDE interface with the project 'ProjetoVendasJavaSwing' open. The code editor displays the 'Cliente.java' file, specifically the implementation of the 'remover' method. This method first checks if the client is null or has an empty CPF. If valid, it then checks if the client already exists in the list. If it does, it removes the client from the list. The 'listarTodos' and 'inserir' methods are also partially visible.

```

38
39         if (cliente.getNome().trim().equals("")) {
40             throw new Exception("Informar o Nome do cliente");
41         }
42         if (this.verificaExistencia(cliente) >= 0) {
43             throw new Exception("O referido cliente já se encontra cadastrado");
44         }
45         this.lista.add(cliente);
46
47     public void remover(Cliente cliente) throws Exception {
48         if (cliente == null) {
49             throw new Exception("O cliente não foi instanciado");
50         }
51         if (cliente.getCpf() == null) {
52             throw new Exception("Informar o CPF do cliente");
53         }
54         if (cliente.getNome().trim().equals("")) {
55             throw new Exception("Informar o Nome do cliente");
56         }
57         if (this.verificaExistencia(cliente) == -1) {
58             throw new Exception("O referido cliente não encontra cadastrado");
59         }
60         this.lista.remove(this.verificaExistencia(cliente));
61     }
62
63     public void atualizar(Cliente cliente) throws Exception {
64         if (cliente == null) {
65             throw new Exception("O cliente não foi instanciado");
66         }
67         if (cliente.getCpf() == null) {
68             throw new Exception("Informar o CPF do cliente");
69         }
70         if (cliente.getCpf().trim().equals("")) {

```

Figura 27 - Método remover Cliente

Implementação do Método Atualizar Cliente

```
ProjetoVendasJavaSwing - NetBeans IDE 6.9
Arquivo Editar Exibir Navegar Código-fonte Refatorar Executar Depurar Perfil Equipe Ferramentas Janela Ajuda
<config. padrão> Pesquisar (Ctrl+F)
ProjetoVendasJavaSwing
  Pacotes de código-fonte
    projetovendasjavawsing
      Cliente.java
      ItemVenda.java
      Main.java
      Produto.java
      RepositorioCliente.java
      Venda.java
  Pacotes de testes
  Bibliotecas
  Bibliotecas de testes

listarTodos - Navegador
Visualização dos membros
  ReppositórioCliente
    atualizar(Cliente cliente)
    inserir(Cliente cliente)
    listarTodos() : ArrayList<Cliente>
    remover(Cliente cliente)
    verificaExistencia(Cliente cliente) : int
    lista : ArrayList<Cliente>

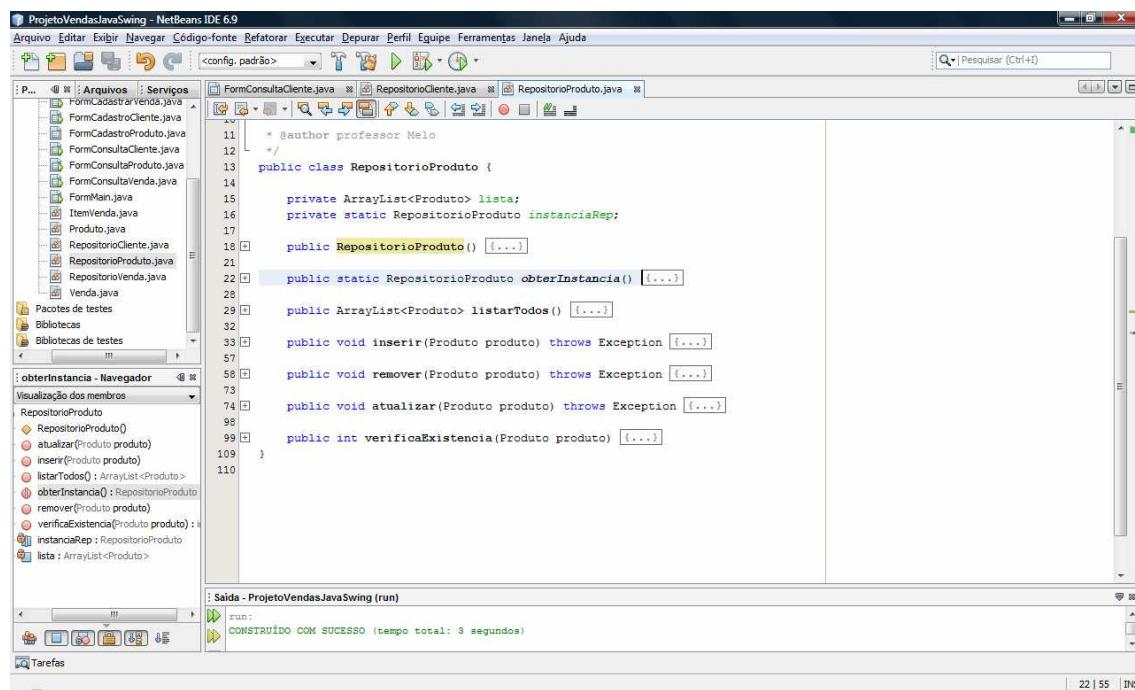
Cliente.java
  59   }
  60   this.lista.remove(this.verificaExistencia(cliente));
  61 }
  62
  63 public void atualizar(Cliente cliente) throws Exception {
  64   if (cliente == null) {
  65     throw new Exception("O cliente não foi instanciado");
  66   }
  67   if (cliente.getCpf() == null) {
  68     throw new Exception("Informar o CPF do cliente");
  69   }
  70   if (cliente.getName().trim().equals("")) {
  71     throw new Exception("Informar o Nome do cliente");
  72   }
  73   if (cliente.getName() == null) {
  74     throw new Exception("Informar o Nome do cliente");
  75   }
  76   if (cliente.getName().trim().equals("")) {
  77     throw new Exception("Informar o Nome do cliente");
  78   }
  79   if (this.verificaExistencia(cliente) == -1) {
  80     throw new Exception("O referido cliente não encontra cadastrado");
  81   }
  82   this.lista.set(this.verificaExistencia(cliente), cliente);
  83 }
  84
  85 public int verificaExistencia(Cliente cliente) {
  86   int retorno = -1;
  87   for (int i = 0; i < this.lista.size(); i++) {
  88     if (cliente.getCpf().trim().equals(this.lista.get(i).getCpf().trim())) {
  89       retorno = i;
  90       break;
  91     }
  92   }
  93 }

  94
  95
  96
  97
  98
  99
  100
  101
  102
  103
  104
  105
  106
  107
  108
  109
  110
  111
  112
  113
  114
  115
  116
  117
  118
  119
  120
  121
  122
  123
  124
  125
  126
  127
  128
  129
  130
  131
  132
  133
  134
  135
  136
  137
  138
  139
  140
  141
  142
  143
  144
  145
  146
  147
  148
  149
  150
  151
  152
  153
  154
  155
  156
  157
  158
  159
  160
  161
  162
  163
  164
  165
  166
  167
  168
  169
  170
  171
  172
  173
  174
  175
  176
  177
  178
  179
  180
  181
  182
  183
  184
  185
  186
  187
  188
  189
  190
  191
  192
  193
  194
  195
  196
  197
  198
  199
  200
  201
  202
  203
  204
  205
  206
  207
  208
  209
  210
  211
  212
  213
  214
  215
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240
  241
  242
  243
  244
  245
  246
  247
  248
  249
  250
  251
  252
  253
  254
  255
  256
  257
  258
  259
  260
  261
  262
  263
  264
  265
  266
  267
  268
  269
  270
  271
  272
  273
  274
  275
  276
  277
  278
  279
  280
  281
  282
  283
  284
  285
  286
  287
  288
  289
  290
  291
  292
  293
  294
  295
  296
  297
  298
  299
  300
  301
  302
  303
  304
  305
  306
  307
  308
  309
  310
  311
  312
  313
  314
  315
  316
  317
  318
  319
  320
  321
  322
  323
  324
  325
  326
  327
  328
  329
  330
  331
  332
  333
  334
  335
  336
  337
  338
  339
  340
  341
  342
  343
  344
  345
  346
  347
  348
  349
  350
  351
  352
  353
  354
  355
  356
  357
  358
  359
  360
  361
  362
  363
  364
  365
  366
  367
  368
  369
  370
  371
  372
  373
  374
  375
  376
  377
  378
  379
  380
  381
  382
  383
  384
  385
  386
  387
  388
  389
  390
  391
  392
  393
  394
  395
  396
  397
  398
  399
  400
  401
  402
  403
  404
  405
  406
  407
  408
  409
  410
  411
  412
  413
  414
  415
  416
  417
  418
  419
  420
  421
  422
  423
  424
  425
  426
  427
  428
  429
  430
  431
  432
  433
  434
  435
  436
  437
  438
  439
  440
  441
  442
  443
  444
  445
  446
  447
  448
  449
  450
  451
  452
  453
  454
  455
  456
  457
  458
  459
  460
  461
  462
  463
  464
  465
  466
  467
  468
  469
  470
  471
  472
  473
  474
  475
  476
  477
  478
  479
  480
  481
  482
  483
  484
  485
  486
  487
  488
  489
  490
  491
  492
  493
  494
  495
  496
  497
  498
  499
  500
  501
  502
  503
  504
  505
  506
  507
  508
  509
  510
  511
  512
  513
  514
  515
  516
  517
  518
  519
  520
  521
  522
  523
  524
  525
  526
  527
  528
  529
  530
  531
  532
  533
  534
  535
  536
  537
  538
  539
  540
  541
  542
  543
  544
  545
  546
  547
  548
  549
  550
  551
  552
  553
  554
  555
  556
  557
  558
  559
  560
  561
  562
  563
  564
  565
  566
  567
  568
  569
  570
  571
  572
  573
  574
  575
  576
  577
  578
  579
  580
  581
  582
  583
  584
  585
  586
  587
  588
  589
  590
  591
  592
  593
  594
  595
  596
  597
  598
  599
  600
  601
  602
  603
  604
  605
  606
  607
  608
  609
  610
  611
  612
  613
  614
  615
  616
  617
  618
  619
  620
  621
  622
  623
  624
  625
  626
  627
  628
  629
  630
  631
  632
  633
  634
  635
  636
  637
  638
  639
  640
  641
  642
  643
  644
  645
  646
  647
  648
  649
  650
  651
  652
  653
  654
  655
  656
  657
  658
  659
  660
  661
  662
  663
  664
  665
  666
  667
  668
  669
  670
  671
  672
  673
  674
  675
  676
  677
  678
  679
  680
  681
  682
  683
  684
  685
  686
  687
  688
  689
  690
  691
  692
  693
  694
  695
  696
  697
  698
  699
  700
  701
  702
  703
  704
  705
  706
  707
  708
  709
  710
  711
  712
  713
  714
  715
  716
  717
  718
  719
  720
  721
  722
  723
  724
  725
  726
  727
  728
  729
  730
  731
  732
  733
  734
  735
  736
  737
  738
  739
  740
  741
  742
  743
  744
  745
  746
  747
  748
  749
  750
  751
  752
  753
  754
  755
  756
  757
  758
  759
  760
  761
  762
  763
  764
  765
  766
  767
  768
  769
  770
  771
  772
  773
  774
  775
  776
  777
  778
  779
  780
  781
  782
  783
  784
  785
  786
  787
  788
  789
  790
  791
  792
  793
  794
  795
  796
  797
  798
  799
  800
  801
  802
  803
  804
  805
  806
  807
  808
  809
  810
  811
  812
  813
  814
  815
  816
  817
  818
  819
  820
  821
  822
  823
  824
  825
  826
  827
  828
  829
  830
  831
  832
  833
  834
  835
  836
  837
  838
  839
  840
  841
  842
  843
  844
  845
  846
  847
  848
  849
  850
  851
  852
  853
  854
  855
  856
  857
  858
  859
  860
  861
  862
  863
  864
  865
  866
  867
  868
  869
  870
  871
  872
  873
  874
  875
  876
  877
  878
  879
  880
  881
  882
  883
  884
  885
  886
  887
  888
  889
  890
  891
  892
  893
  894
  895
  896
  897
  898
  899
  900
  901
  902
  903
  904
  905
  906
  907
  908
  909
  910
  911
  912
  913
  914
  915
  916
  917
  918
  919
  920
  921
  922
  923
  924
  925
  926
  927
  928
  929
  930
  931
  932
  933
  934
  935
  936
  937
  938
  939
  940
  941
  942
  943
  944
  945
  946
  947
  948
  949
  950
  951
  952
  953
  954
  955
  956
  957
  958
  959
  960
  961
  962
  963
  964
  965
  966
  967
  968
  969
  970
  971
  972
  973
  974
  975
  976
  977
  978
  979
  980
  981
  982
  983
  984
  985
  986
  987
  988
  989
  990
  991
  992
  993
  994
  995
  996
  997
  998
  999
  1000
  1001
  1002
  1003
  1004
  1005
  1006
  1007
  1008
  1009
  1010
  1011
  1012
  1013
  1014
  1015
  1016
  1017
  1018
  1019
  1020
  1021
  1022
  1023
  1024
  1025
  1026
  1027
  1028
  1029
  1030
  1031
  1032
  1033
  1034
  1035
  1036
  1037
  1038
  1039
  1040
  1041
  1042
  1043
  1044
  1045
  1046
  1047
  1048
  1049
  1050
  1051
  1052
  1053
  1054
  1055
  1056
  1057
  1058
  1059
  1060
  1061
  1062
  1063
  1064
  1065
  1066
  1067
  1068
  1069
  1070
  1071
  1072
  1073
  1074
  1075
  1076
  1077
  1078
  1079
  1080
  1081
  1082
  1083
  1084
  1085
  1086
  1087
  1088
  1089
  1090
  1091
  1092
  1093
  1094
  1095
  1096
  1097
  1098
  1099
  1100
  1101
  1102
  1103
  1104
  1105
  1106
  1107
  1108
  1109
  1110
  1111
  1112
  1113
  1114
  1115
  1116
  1117
  1118
  1119
  1120
  1121
  1122
  1123
  1124
  1125
  1126
  1127
  1128
  1129
  1130
  1131
  1132
  1133
  1134
  1135
  1136
  1137
  1138
  1139
  1140
  1141
  1142
  1143
  1144
  1145
  1146
  1147
  1148
  1149
  1150
  1151
  1152
  1153
  1154
  1155
  1156
  1157
  1158
  1159
  1160
  1161
  1162
  1163
  1164
  1165
  1166
  1167
  1168
  1169
  1170
  1171
  1172
  1173
  1174
  1175
  1176
  1177
  1178
  1179
  1180
  1181
  1182
  1183
  1184
  1185
  1186
  1187
  1188
  1189
  1190
  1191
  1192
  1193
  1194
  1195
  1196
  1197
  1198
  1199
  1200
  1201
  1202
  1203
  1204
  1205
  1206
  1207
  1208
  1209
  1210
  1211
  1212
  1213
  1214
  1215
  1216
  1217
  1218
  1219
  1220
  1221
  1222
  1223
  1224
  1225
  1226
  1227
  1228
  1229
  1230
  1231
  1232
  1233
  1234
  1235
  1236
  1237
  1238
  1239
  1240
  1241
  1242
  1243
  1244
  1245
  1246
  1247
  1248
  1249
  1250
  1251
  1252
  1253
  1254
  1255
  1256
  1257
  1258
  1259
  1260
  1261
  1262
  1263
  1264
  1265
  1266
  1267
  1268
  1269
  1270
  1271
  1272
  1273
  1274
  1275
  1276
  1277
  1278
  1279
  1280
  1281
  1282
  1283
  1284
  1285
  1286
  1287
  1288
  1289
  1290
  1291
  1292
  1293
  1294
  1295
  1296
  1297
  1298
  1299
  1300
  1301
  1302
  1303
  1304
  1305
  1306
  1307
  1308
  1309
  1310
  1311
  1312
  1313
  1314
  1315
  1316
  1317
  1318
  1319
  1320
  1321
  1322
  1323
  1324
  1325
  1326
  1327
  1328
  1329
  1330
  1331
  1332
  1333
  1334
  1335
  1336
  1337
  1338
  1339
  1340
  1341
  1342
  1343
  1344
  1345
  1346
  1347
  1348
  1349
  1350
  1351
  1352
  1353
  1354
  1355
  1356
  1357
  1358
  1359
  1360
  1361
  1362
  1363
  1364
  1365
  1366
  1367
  1368
  1369
  1370
  1371
  1372
  1373
  1374
  1375
  1376
  1377
  1378
  1379
  1380
  1381
  1382
  1383
  1384
  1385
  1386
  1387
  1388
  1389
  1390
  1391
  1392
  1393
  1394
  1395
  1396
  1397
  1398
  1399
  1400
  1401
  1402
  1403
  1404
  1405
  1406
  1407
  1408
  1409
  1410
  1411
  1412
  1413
  1414
  1415
  1416
  1417
  1418
  1419
  1420
  1421
  1422
  1423
  1424
  1425
  1426
  1427
  1428
  1429
  1430
  1431
  1432
  1433
  1434
  1435
  1436
  1437
  1438
  1439
  1440
  1441
  1442
  1443
  1444
  1445
  1446
  1447
  1448
  1449
  1450
  1451
  1452
  1453
  1454
  1455
  1456
  1457
  1458
  1459
  1460
  1461
  1462
  1463
  1464
  1465
  1466
  1467
  1468
  1469
  1470
  1471
  1472
  1473
  1474
  1475
  1476
  1477
  1478
  1479
  1480
  1481
  1482
  1483
  1484
  1485
  1486
  1487
  1488
  1489
  1490
  1491
  1492
  1493
  1494
  1495
  1496
  1497
  1498
  1499
  1500
  1501
  1502
  1503
  1504
  1505
  1506
  1507
  1508
  1509
  1510
  1511
  1512
  1513
  1514
  1515
  1516
  1517
  1518
  1519
  1520
  1521
  1522
  1523
  1524
  1525
  1526
  1527
  1528
  1529
  1530
  1531
  1532
  1533
  1534
  1535
  1536
  1537
  1538
  1539
  1540
  1541
  1542
  1543
  1544
  1545
  1546
  1547
  1548
  1549
  1550
  1551
  1552
  1553
  1554
  1555
  1556
  1557
  1558
  1559
  1560
  1561
  1562
  1563
  1564
  1565
  1566
  1567
  1568
  1569
  1570
  1571
  1572
  1573
  1574
  1575
  1576
  1577
  1578
  1579
  1580
  1581
  1582
  1583
  1584
  1585
  1586
  1587
  1588
  1589
  1590
  1591
  1592
  1593
  1594
  1595
  1596
  1597
  1598
  1599
  1600
  1601
  1602
  1603
  1604
  1605
  1606
  1607
  1608
  1609
  1610
  1611
  1612
  1613
  1614
  1615
  1616
  1617
  1618
  1619
  1620
  1621
  1622
  1623
  1624
  1625
  1626
  1627
  1628
  1629
  1630
  1631
  1632
  1633
  1634
  1635
  1636
  1637
  1638
  1639
  1640
  1641
  1642
  1643
  1644
  1645
  1646
  1647
  1648
  1649
  1650
  1651
  1652
  1653
  1654
  1655
  1656
  1657
  1658
  1659
  1660
  1661
  1662
  1663
  1664
  1665
  1666
  1667
  1668
  1669
  1670
  1671
  1672
  1673
  1674
  1675
  1676
  1677
  1678
  1679
  1680
  1681
  1682
  1683
  1684
  1685
  1686
  1687
  1688
  1689
  1690
  1691
  1692
  1693
  1694
  1695
  1696
  1697
  1698
  1699
  1700
  1701
  1702
  1703
  1704
  1705
  1706
  1707
  1708
  1709
  1710
  1711
  1712
  1713
  1714
  1715
  
```

Classe RepositorioProduto

A figura abaixo ilustra a classe RepositorioProduto, podemos observar que foi declarado um atributo do tipo ArrayList<Produto>, este atributo tem por finalidade armazenar todos os produtos cadastrado pelo sistema. Também foi declarado um atributo do tipo RepositorioProduto, este atributo tem por finalidade implementar este repositório com o padrão **singleton**.

Também podemos observar a assinatura dos métodos definidos no diagrama de classes.



```
ProjetoVendasJavaSwing - NetBeans IDE 6.9
Arquivo Editar Exibir Navegar Código-fonte Refatorar Executar Depurar Perfil Equipe Ferramentas Janela Ajuda
<config. padrao> Pesquisar (Ctrl+I)
P... Arquivos Servicos FormConsultaCliente.java RepositorioCliente.java RepositorioProduto.java
FormCadastroCliente.java FormCadastroProduto.java FormConsultaCliente.java FormConsultaProduto.java
FormConsultaVenda.java FormMan.java ItemVenda.java Produto.java RepositorioCliente.java
RepositorioProduto.java RepositorioVenda.java Venda.java
Pacotes de testes Bibliotecas Bibliotecas de testes
obterinstancia - Navegador Visualização dos membros
RepositorioProduto
  • RepositorioProduto()
  • atualizar(Produto produto)
  • inserir(Produto produto)
  • listarTodos(): ArrayList<Produto>
  • obterInstancia(): RepositorioProduto
  • remover(Produto produto)
  • verificaExistencia(Produto produto)
  • instanciaRep : RepositorioProduto
  • lista : ArrayList<Produto>
Saída - ProjetoVendasJavaSwing (run)
run: CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)
Tarefas
22 | 55 | INS
```

```
11 * @author professor Melo
12 */
13 public class RepositorioProduto {
14
15     private ArrayList<Produto> lista;
16     private static RepositorioProduto instanciaRep;
17
18     public RepositorioProduto() {...}
19
20     public static RepositorioProduto obterInstancia() {...}
21
22     public ArrayList<Produto> listarTodos() {...}
23
24     public void inserir(Produto produto) throws Exception {...}
25
26     public void remover(Produto produto) throws Exception {...}
27
28     public void atualizar(Produto produto) throws Exception {...}
29
30     public int verificaExistencia(Produto produto) {...}
31
32 }
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110 }
```

Figura 29 - Estrutura da classe RepositórioProduto

Implementação do Construtor e dos Métodos Listar Todos e obterInstancia

```

    import java.util.ArrayList;
    ...
    /**
     * @author professor Melo
     */
    public class RepositorioProduto {
        ...
        private ArrayList<Produto> lista;
        private static RepositorioProduto instanciaRep;
        ...
        public RepositorioProduto() {
            this.lista = new ArrayList<Produto>();
        }
        ...
        public static RepositorioProduto obterInstancia() {
            if (instanciaRep == null) {
                instanciaRep = new RepositorioProduto();
            }
            return instanciaRep;
        }
        ...
        public ArrayList<Produto> listarTodos() {
            return this.lista;
        }
        ...
        public void inserir(Produto produto) throws Exception {
            if (produto == null) {
                throw new Exception("O cliente não foi instanciado");
            }
        }
    }

```

Figura 30 - Métodos listarTodos, obterInstancia e construtor do RepositórioProduto

Implementação do Método Inserir Produto

```

    ...
    public void inserir(Produto produto) throws Exception {
        if (produto == null) {
            throw new Exception("O cliente não foi instanciado");
        }
        if (produto.getCodigoBarras() == null) {
            throw new Exception("Informar o código de barras do produto");
        }
        if (produto.getCodigoBarras().trim().equals("")) {
            throw new Exception("Informar o código de barras do produto");
        }
        if (produto.getDescricao() == null) {
            throw new Exception("Informar a descrição do produto");
        }
        if (produto.getDescricao().trim().equals("")) {
            throw new Exception("Informar a descrição do produto");
        }
        if (produto.getPreco() <= 0) {
            throw new Exception("O preço do produto deverá ser superior a zero");
        }
        if (this.verificaExistencia(produto) >= 0) {
            throw new Exception("O referido produto já se encontra cadastrado");
        }
        this.lista.add(produto);
    }
    ...

```

Figura 31 - Método inserir produto

Implementação do Método Remover Produto

```

    46     }
    47     this.lista.add(Produto);
    48   }
    49
    50   public void remover(Produto produto) throws Exception {
    51     if (produto == null) {
    52       throw new Exception("O cliente não foi instanciado");
    53     }
    54     if (produto.getCodigoBarras() == null) {
    55       throw new Exception("Informar o código de barras do produto");
    56     }
    57     if (produto.getCodigoBarras().trim().equals("")) {
    58       throw new Exception("Informar o código de barras do produto");
    59     }
    60     if (this.verificaExistencia(produto) == -1) {
    61       throw new Exception("O referido produto não se encontra cadastrado");
    62     }
    63     this.lista.remove(this.verificaExistencia(produto));
    64   }
    65
    66   public void atualizar(Produto produto) throws Exception {...}
    67
    68   public int verificaExistencia(Produto produto) {...}
    69
    70 }
    71
    72
    73
    74
    75
    76
    77
    78
    79
    80
    81
    82
    83
    84
    85
    86
    87
    88
    89
    90
    91
    92
    93
    94
    95
    96
    97
    98
    99
    100
    101
    102

```

Figura 32 - Método remover produto

Implementação do Método Atualizar Produto

```

    64   }
    65
    66   public void atualizar(Produto produto) throws Exception {
    67     if (produto == null) {
    68       throw new Exception("O cliente não foi instanciado");
    69     }
    70     if (produto.getCodigoBarras() == null) {
    71       throw new Exception("Informar o código de barras do produto");
    72     }
    73     if (produto.getCodigoBarras().trim().equals("")) {
    74       throw new Exception("Informar o código de barras do produto");
    75     }
    76     if (produto.getDescricao() == null) {
    77       throw new Exception("Informar a descrição do produto");
    78     }
    79     if (produto.getDescricao().trim().equals("")) {
    80       throw new Exception("Informar a descrição do produto");
    81     }
    82     if (produto.getPreco() <= 0) {
    83       throw new Exception("O preço do produto deverá ser superior a zero");
    84     }
    85     if (this.verificaExistencia(produto) < 0) {
    86       throw new Exception("O referido produto não se encontra cadastrado");
    87     }
    88     this.lista.set(this.verificaExistencia(produto), produto);
    89   }
    90
    91   public int verificaExistencia(Produto produto) {...}
    92
    93
    94
    95
    96
    97
    98
    99
    100
    101
    102

```

Figura 33 - Método atualizar produto

Implementação do Método Verifica Existência

Este método tem como objetivo pesquisar um objeto pelo seu **código de barras** e retornar o índice deste objeto, caso não seja encontrado nenhum objeto no qual corresponda ao critério de pesquisa, o método retornará -1.

```

    if (produto.getDescricao() == null) {
        throw new Exception("Informar a descrição do produto");
    }
    if (produto.getDescricao().trim().equals("")) {
        throw new Exception("Informar a descrição do produto");
    }
    if (produto.getPreco() <= 0) {
        throw new Exception("O preço do produto deverá ser superior a zero");
    }
    if (this.verificaExistencia(produto) < 0) {
        throw new Exception("O referido produto não se encontra cadastrado");
    }
    this.lista.set(this.verificaExistencia(produto), produto);
}

public int verificaExistencia(Produto produto) {
    int retorno = -1;
    for (int i = 0; i < this.lista.size(); i++) {
        if (produto.getCodigoBarras().trim().equals(this.lista.get(i).getCodigoBarras().trim())) {
            retorno = i;
            break;
        }
    }
    return retorno;
}

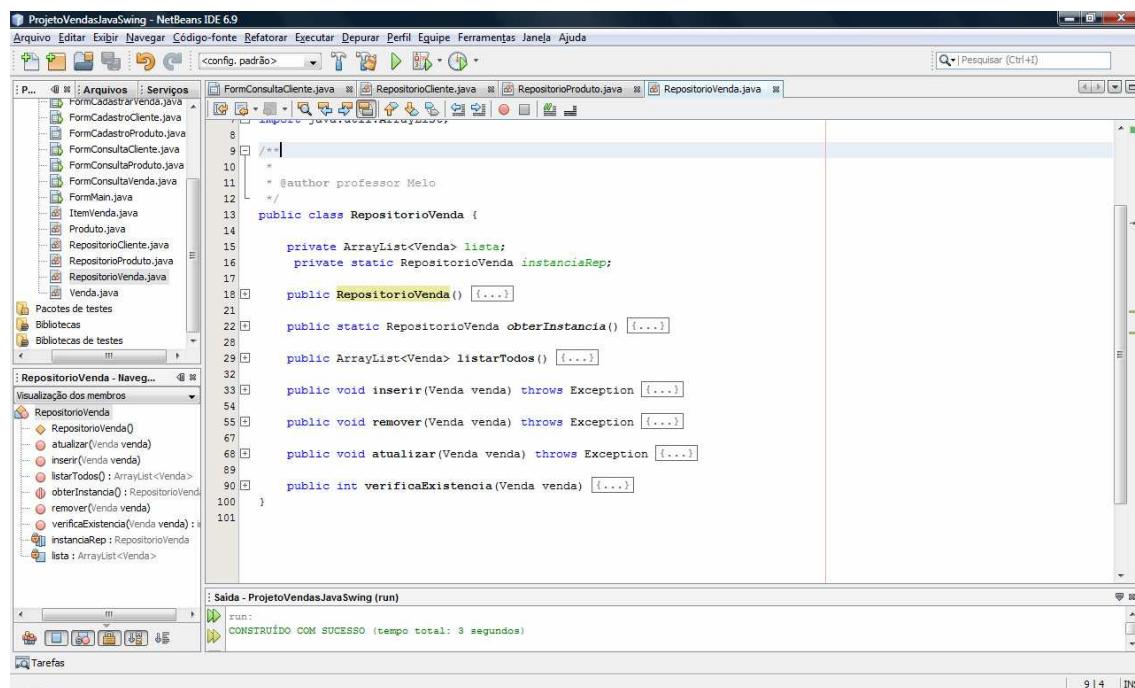
```

Figura 34 - Método verifica existência produto

Classe RepositorioVenda

A figura abaixo ilustra a classe RepositorioVenda, podemos observar que foi declarado um atributo do tipo ArrayList<Venda>, este atributo tem por finalidade armazenar todos os produtos cadastrado pelo sistema. Também foi declarado um atributo do tipo RepositorioVenda, este atributo tem por finalidade implementar este repositório com o padrão **singleton**.

Também podemos observar a assinatura dos métodos definidos no diagrama de classes.



The screenshot shows the NetBeans IDE interface with the project 'ProjetoVendasJavaSwing' open. The central window displays the code for the `RepositoryVenda.java` file. The code implements a singleton pattern for managing a list of `Venda` objects. The interface includes tabs for other files like `FormConsultaCliente.java`, `RepositorioCliente.java`, `RepositorioProduto.java`, and `RepositorioVenda.java`. The left pane shows the package structure and member visualization. The bottom pane shows the build output: 'run: CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)'.

```
8
9  /*
10 */
11 /**
12 * @author professor Melo
13 */
14 public class RepositorioVenda {
15     private ArrayList<Venda> lista;
16     private static RepositorioVenda instanciaRep;
17
18     public RepositorioVenda() {...}
19
20     public static RepositorioVenda obterInstancia() {...}
21
22     public ArrayList<Venda> listarTodos() {...}
23
24     public void inserir(Venda venda) throws Exception {...}
25
26     public void remover(Venda venda) throws Exception {...}
27
28     public void atualizar(Venda venda) throws Exception {...}
29
30     public int verificaExistencia(Venda venda) {...}
31
32 }
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 }
```

Figura 35 - Estrutura da classe RepositórioVenda

Implementação do Construtor e dos Métodos listarTodos e obterIntancia

```

    * @author professor Melo
    */
    public class RepositorioVenda {
        private ArrayList<Venda> lista;
        private static RepositorioVenda instanciaRep;

        public RepositorioVenda() {
            this.lista = new ArrayList<Venda>();
        }

        public static RepositorioVenda obterInstancia() {
            if (instanciaRep == null) {
                instanciaRep = new RepositorioVenda();
            }
            return instanciaRep;
        }

        public ArrayList<Venda> listarTodos() {
            return this.lista;
        }

        public void inserir(Venda venda) throws Exception {...}

        public void remover(Venda venda) throws Exception {...}

        public void atualizar(Venda venda) throws Exception {...}

        public int verificaExistencia(Venda venda) {...}
    }

```

Figura 36 - Construtor e métodos listar todos e obterIntancia

Implementação do Método Inserir Venda

```

        public void inserir(Venda venda) throws Exception {
            if (venda == null) {
                throw new Exception("A venda não foi instanciado");
            }
            if (venda.getNumero() <= 0) {
                throw new Exception("Informar o número da venda");
            }
            if (venda.getData() == null) {
                throw new Exception("Informar a data da venda");
            }
            if (venda.getItem().trim().equals("")) {
                throw new Exception("Informar a data da venda");
            }
            if (venda.getItemsDaVenda().size() <= 0) {
                throw new Exception("Informar pelo menos um item para a venda");
            }
            if (this.VerificaExistencia(venda) >= 0) {
                throw new Exception("A referido venda já se encontra cadastrada");
            }
            this.lista.add(venda);
        }
    }

```

Figura 37 - Método inserir venda

Implementação do Método Remover Venda

The screenshot shows the NetBeans IDE interface with the following details:

- Title Bar:** ProjetoVendasJavaSwing - NetBeans IDE 6.9.1
- Menu Bar:** Arquivo, Editar, Exibir, Navegar, Código-fonte, Refatorar, Executar, Depurar, Perfil, Equipe, Ferramentas, Janela, Ajuda.
- Toolbar:** Standard NetBeans toolbar with icons for file operations, search, and code navigation.
- Project Explorer:** Shows the project structure under 'Projetos'. It includes a 'pacote de código-fonte' named 'projetovendasjavawsing' containing classes like Cliente.java, ItemVenda.java, Main.java, Produto.java, RepositoryCliente.java, RepositoryProduto.java, RepositoryVenda.java, and Venda.java. There are also 'pacotes de testes' and 'bibliotecas' sections.
- Code Editor:** Displays the 'RepositoryVenda.java' file with the following code snippet for the 'remover' method:

```

41     if (this.verificaExistencia(venda) >= 0) {
42         throw new Exception("A referida venda já se encontra cadastrada");
43     }
44     this.lista.add(venda);
45
46
47     public void remover(Venda venda) throws Exception {
48         if (venda == null) {
49             throw new Exception("A venda não foi instanciado");
50         }
51         if (venda.getNumero() <= 0) {
52             throw new Exception("Informar o número da venda");
53         }
54         if (this.verificaExistencia(venda) < 0) {
55             throw new Exception("A referida venda não se encontra cadastrada");
56         }
57         this.lista.remove(this.VerificaExistencia(venda));
58     }
59
60     public void atualizar(Venda venda) throws Exception {...}
61
62     public int verificaExistencia(Venda venda) {...}
63
64 }
```
- Navigation:** Shows the 'inserir - Navegador' panel with methods like 'atualizar', 'inserir', 'listarTodos', 'remover', 'verificaExistencia', and 'lista'.
- Status Bar:** Shows the status '25 | 55 | INS'.

Figura 38 - Método remover venda

Implementação do Método Atualizar Venda

The screenshot shows the NetBeans IDE interface with the following details:

- Title Bar:** ProjetoVendasJavaSwing - NetBeans IDE 6.9.1
- Menu Bar:** Arquivo, Editar, Exibir, Navegar, Código-fonte, Refatorar, Executar, Depurar, Perfil, Equipe, ferramentas, Janela, Ajuda.
- Toolbar:** Standard NetBeans toolbar with icons for file operations, search, and code navigation.
- Project Explorer:** Shows the project structure under 'Projetos'. It includes a 'pacote de código-fonte' named 'projetovendasjavawsing' containing classes like Cliente.java, ItemVenda.java, Main.java, Produto.java, RepositoryCliente.java, RepositoryProduto.java, RepositoryVenda.java, and Venda.java. There are also 'pacotes de testes' and 'bibliotecas' sections.
- Code Editor:** Displays the 'RepositoryVenda.java' file with the following code snippet for the 'atualizar' method:

```

56
57     this.lista.remove(this.VerificaExistencia(venda));
58
59
60     public void atualizar(Venda venda) throws Exception {
61         if (venda == null) {
62             throw new Exception("A venda não foi instanciada");
63         }
64         if (venda.getNumero() <= 0) {
65             throw new Exception("Informar o número da venda");
66         }
67         if (venda.getData() == null) {
68             throw new Exception("Informar a data da venda");
69         }
70         if (venda.getData().trim().equals("")) {
71             throw new Exception("Informar a data da venda");
72         }
73         if (venda.getItemsDaVenda().size() <= 0) {
74             throw new Exception("Informar pelo menos um item para a venda");
75         }
76         if (this.verificaExistencia(venda) >= 0) {
77             throw new Exception("A referida venda já se encontra cadastrada");
78         }
79         this.lista.set(this.VerificaExistencia(venda), venda);
80     }
81
82     public int verificaExistencia(Venda venda) {...}
83
84 }
```
- Navigation:** Shows the 'inserir - Navegador' panel with methods like 'atualizar', 'inserir', 'listarTodos', 'remover', 'verificaExistencia', and 'lista'.
- Status Bar:** Shows the status '25 | 55 | INS'.

Figura 39 - Método atualizar venda

Implementação Método verificaExistência

Este método tem como objetivo pesquisar um objeto pelo seu **número** e retornar o índice deste objeto, caso não seja encontrado nenhum objeto no qual corresponda ao critério de pesquisa, o método retornará -1.

```
72     }
73     if (venda.getItensDaVenda().size() <= 0) {
74         throw new Exception("Informar pelo menos um item para a venda");
75     }
76     if (this.verificaExistencia(venda) >= 0) {
77         throw new Exception("A referida venda já se encontra cadastrada");
78     }
79     this.lista.set(this.verificaExistencia(venda), venda);
80 }
81
82 public int verificaExistencia(Venda venda) {
83     int retorno = -1;
84     for (int i = 0; i < this.lista.size(); i++) {
85         if (venda.getNumero() == this.lista.get(i).getNumero()) {
86             retorno = i;
87             break;
88         }
89     }
90     return retorno;
91 }
92 }
93 }
```

Figura 40 - Método verifica existência venda

Trabalhando com componentes Swing

Vamos colocar um formulário **JFrame**, para esta tarefa basta clicar com o botão direito do mouse em cima do pacote **meuprimeiroprojeto** e escolher as opções **novo->Formulário JFrame**. Conforme mostra Figura 41.

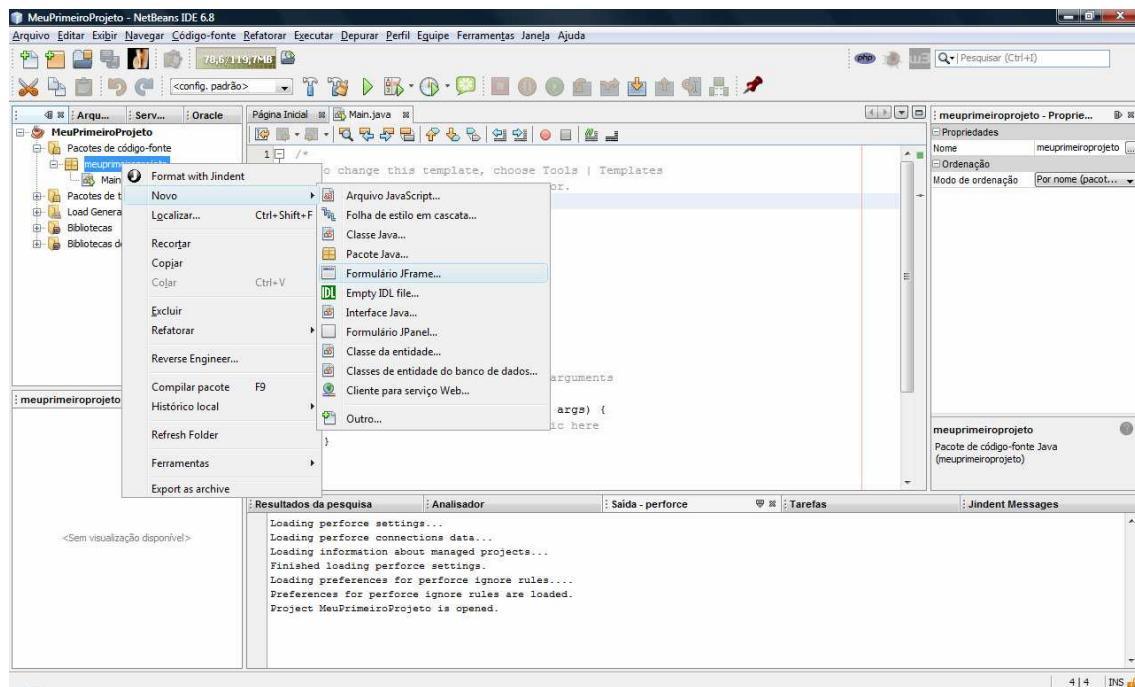


Figura 41 - Incluindo um **JFrame** ao projeto - parte 1

Aparecerá uma tela para configuração do **JFrame**, com as informações do nome da classe, localização e pacote. Figura 42.

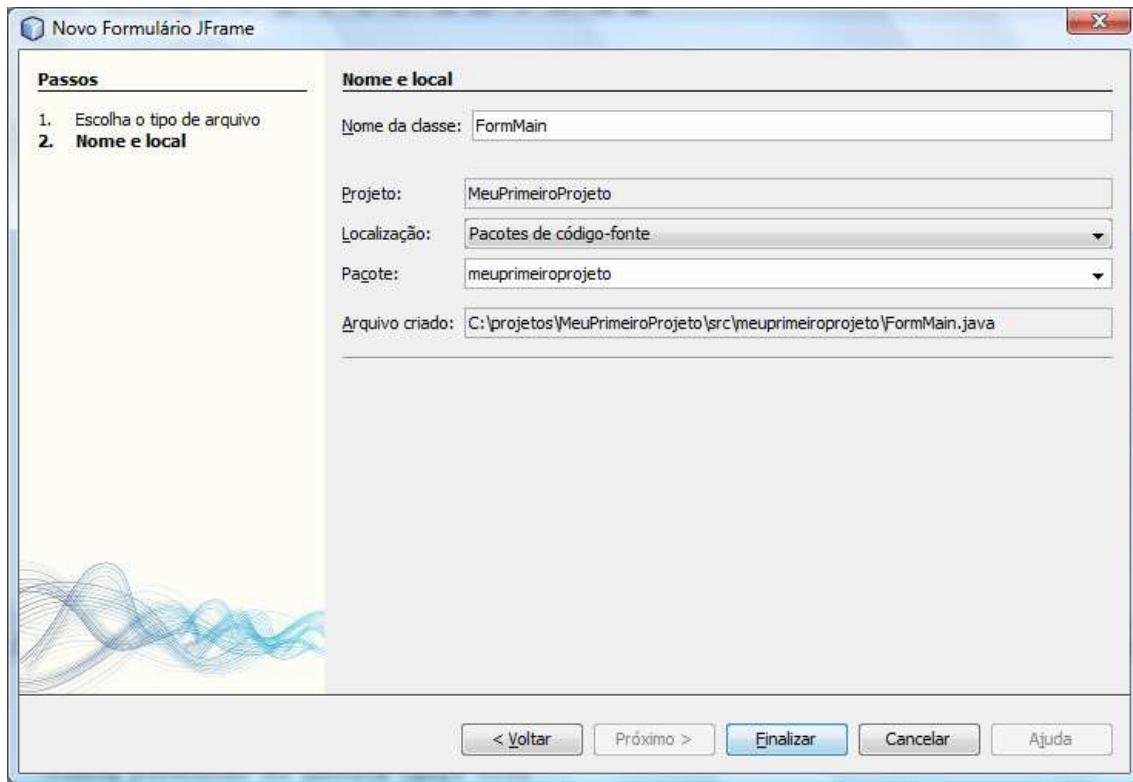


Figura 42 - Incluindo um JFrame ao projeto - parte 2

Após a configuração inicial do **JFrame**, aparecerá um formulário (meio da tela) e a paleta de componentes (lado direito superior da tela), a paleta de componentes contem os principais componentes Swing, para colocar no formulário basta clicar no componente e arrastar para o formulário. Figura 43.

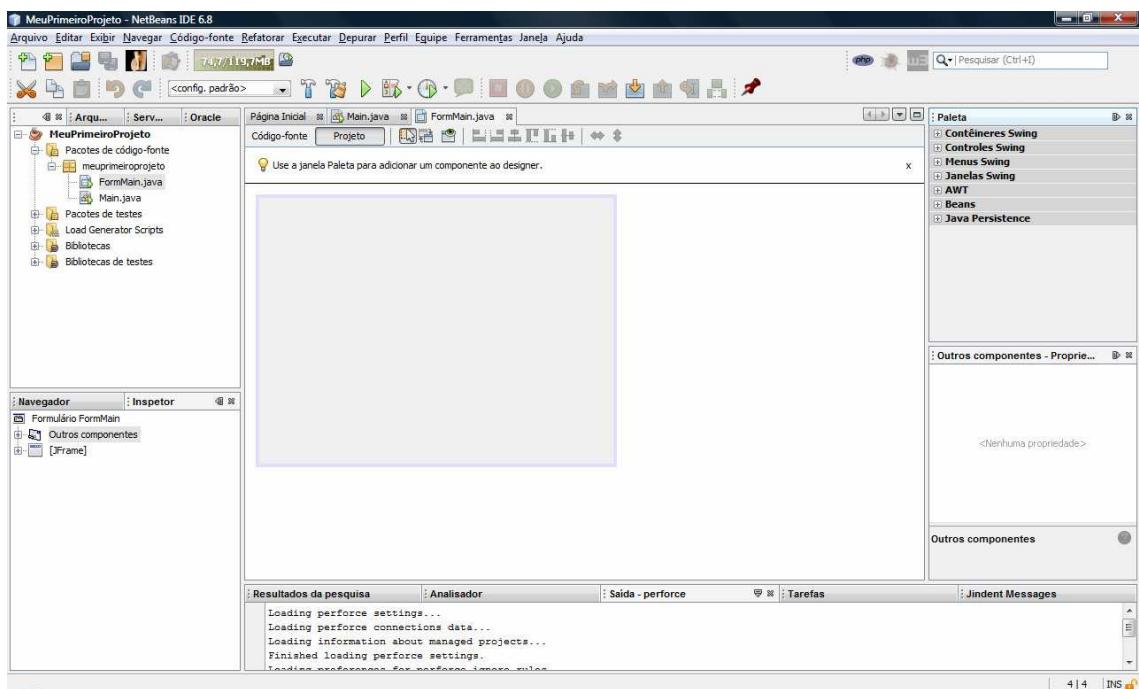


Figura 43 - Incluindo um JFrame ao projeto - parte 3

Configurando a Classe Principal do Projeto

Por padrão quando criamos um projeto aplicação Java no **NetBeans**, ele cria uma **Classe Main.java** automaticamente e a coloca como classe principal do sistema.

No projeto que foi criado **MeuPrimeiroProjeto**, existem duas classes a **Main.java** e **FormMain.java**. Para configurar a Classe principal do projeto é necessário ir ao menu **Arquivo->Projeto Properties**. Figura 44.

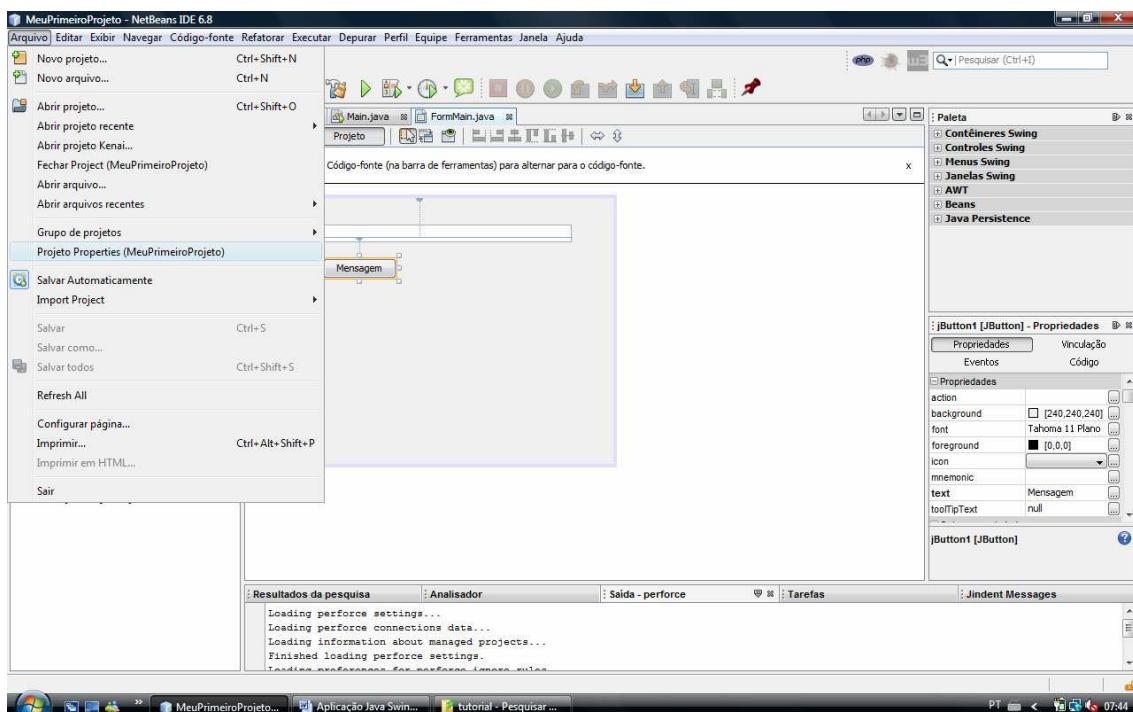


Figura 44 - Configurando a classe principal - parte 1

Aparecerá a tela de propriedades do projeto, onde poderemos personalizar nossa aplicação. Figura 45.

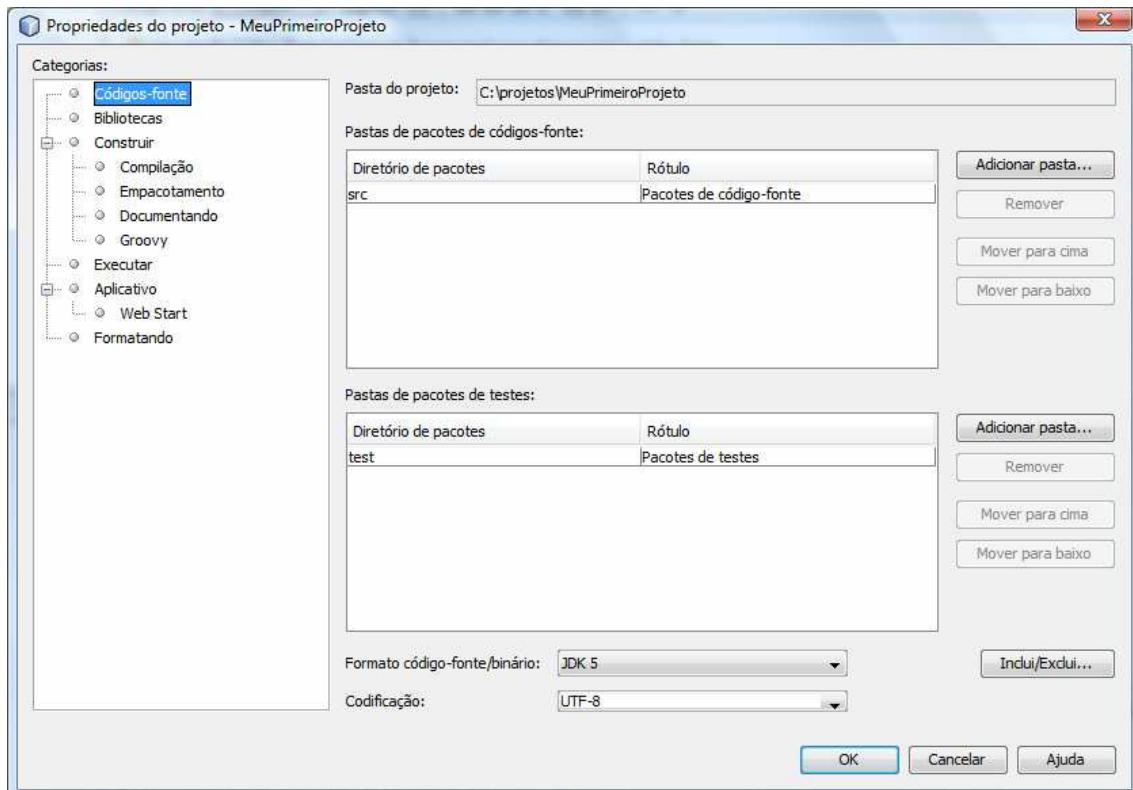


Figura 45 - Configurando a classe principal - parte 2

Para escolher a Classe principal do projeto iremos escolher a opção **Executar** (lado esquerdo da tela), Ao clicar na opção **Executar** aparecerá a opções de escolha da **Classe principal** (centro da tela). Clicar no botão **Procurar**. Figura 46.

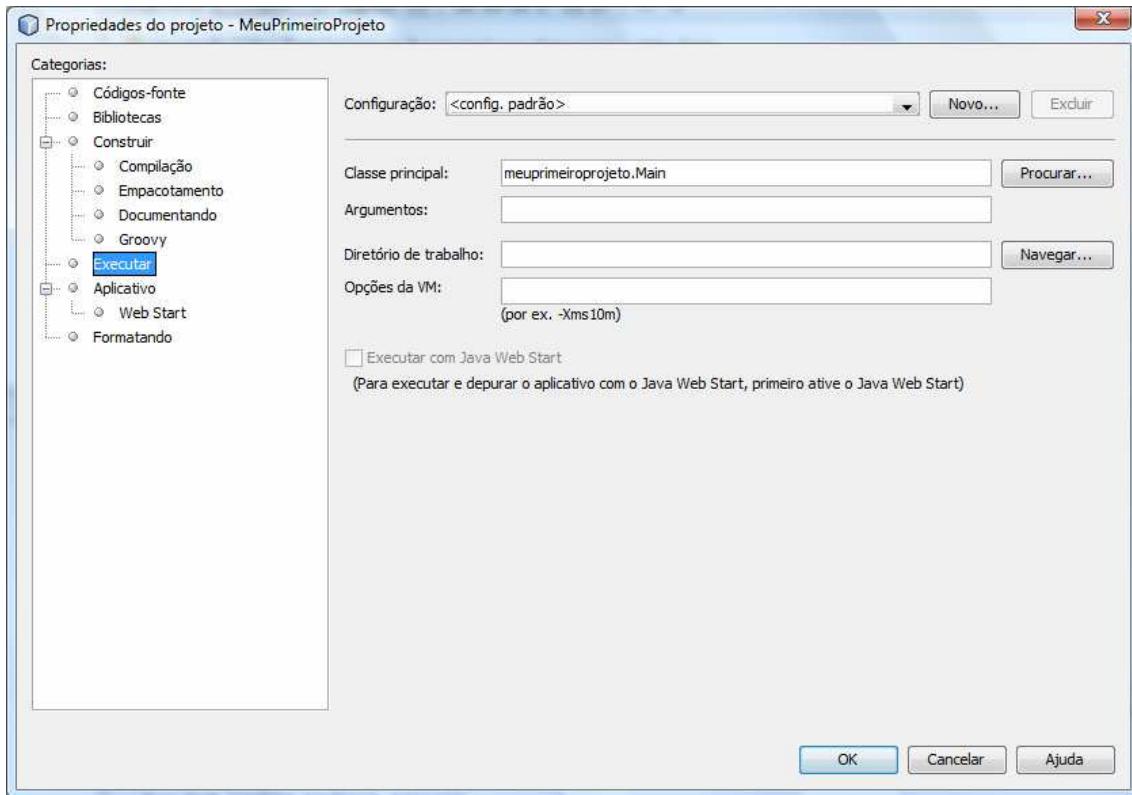


Figura 46 - Configurando a classe principal - parte 3

Após clicar no botão procurar (Figura 46), aparecerão as **Classes** do projeto que contem o método **public static void main(String args[]){}**. Escolher uma das **Classes** listadas e clicar no botão **Selecionar classe principal**. Figura 47.

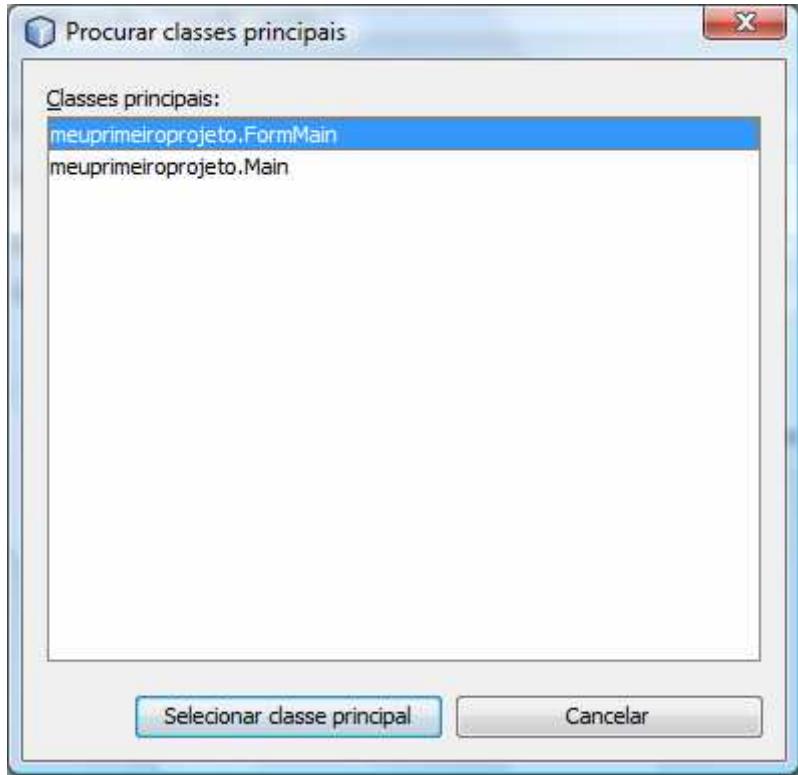


Figura 47 - Configurando a classe principal - parte 4

Manipulando Componentes Swing (básico)

Antes de começarmos a implementar a parte visual(telas) de nosso projeto é necessário estudarmos alguns componentes **Swing**, então veremos os componentes JTextField, JComboBox, JButton e JTable.

JTextField

O componente **JTextField** serve para receber entradas de texto em um formulário.

O método **getText()** serve para obter o conteúdo informado no **JTextField**, ex:
JOptionPane.showMessageDialog(null, jTextField1.getText()); //aparecerá uma mensagem com o conteúdo do **JTextField**.

O método **setText(String texto)** serve para colocar um valor String no **JTextField**, ex: **jTextField1.setText("Eu amo o professor Melo");**//escreverá a String passada no método para o componente **JTextField**.

Foi colocado no formulário dois **JTextField** e dois **JButtons**, onde o clique do primeiro botão exibirá uma mensagem com o conteúdo do **JTextField** do nome, e o segundo botão colocará o texto do **JTextField** do nome 2 para o **JTextField** do nome.

Figura 48.

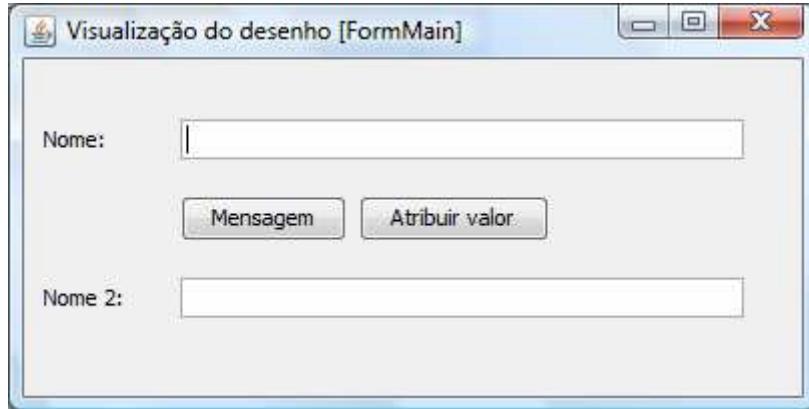


Figura 48 - Manipulando JTextField - parte 1

Colocar no evento do Clique dos botões os códigos que aparecem na Figura 49. Para isso basta dar dois cliques no botão e colocar o código-fonte correspondente.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    JOptionPane.showMessageDialog(null, jTextField1.getText());
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    jTextField1.setText(jTextField2.getText());
}
```

Figura 49 - Manipulando JTextField - parte 2

JComboBox

Vamos adicionar um **JComboBox** e três botões ao formulário. O primeiro botão servirá para mostrar o índice do item escolhido do **JComboBox**, o segundo botão servirá para mostrar o conteúdo do item escolhido no **JComboBox** e o terceiro botão servirá para carregar dinamicamente informações no **JComboBox**. Figura 50.



Figura 50 - Manipulando JComboBox - parte 1

O método **getSelectedIndex()** retorna o índice (valor inteiro) do item escolhido do **JComboBox**, O método **getSelectedItem()** retorna um Objeto que representa o item do **JComboBox** escolhido. A Figura 51 mostra o código-fonte correspondente ao clique de cada botão para manipular o **JComboBox**.

```
private void jButtonIndiceEscolhidoActionPerformed(java.awt.event.ActionEvent evt) {  
    JOptionPane.showMessageDialog(null, jComboBox1.getSelectedIndex());  
}  
  
private void jButtonTextoEscolhidoActionPerformed(java.awt.event.ActionEvent evt) {  
    JOptionPane.showMessageDialog(null, (String) jComboBox1.getSelectedItem());  
}  
  
private void jButtonCarregarDinamActionPerformed(java.awt.event.ActionEvent evt) {  
    DefaultComboBoxModel modelFuncao = new DefaultComboBoxModel();  
    for (int i = 0; i < 10; i++) {  
        modelFuncao.addElement("Melo " + i);  
    }  
    jComboBox1.setModel(modelFuncao);  
}
```

Figura 51 - Manipulando JComboBox - parte 2

A Figura 52 ilustra a execução dos cliques dos **JButtons** definidos anteriormente. Na Figura 52 área 1 temos a tela com a configuração padrão do **JComboBox**, com apenas quatro itens. A Figura 52 área 2 temos a execução dos cliques dos botões e suas ações correspondentes, e na Figura 52 área 3 temos o

JComboBox com os itens carregados no clique do **JButton Carregar dinamicamente**.

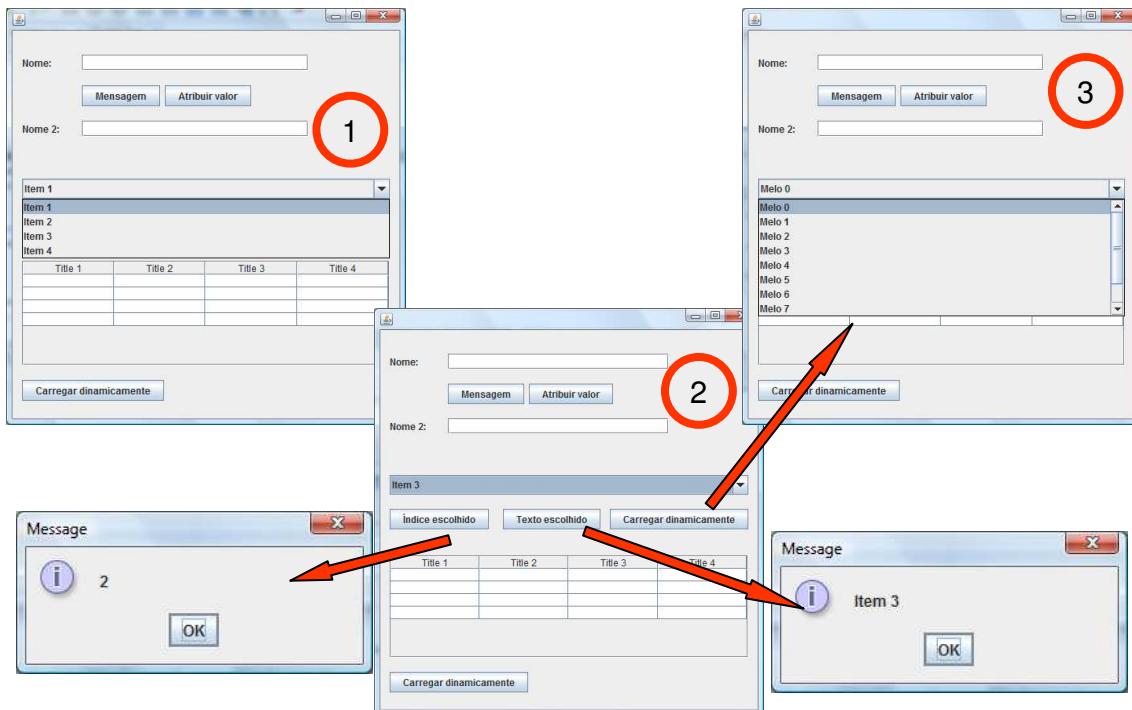


Figura 52 - Manipulando JComboBox - parte 3

JTable

Vamos adicionar um **JTable** e um **JButton** ao formulário. Figura 53.

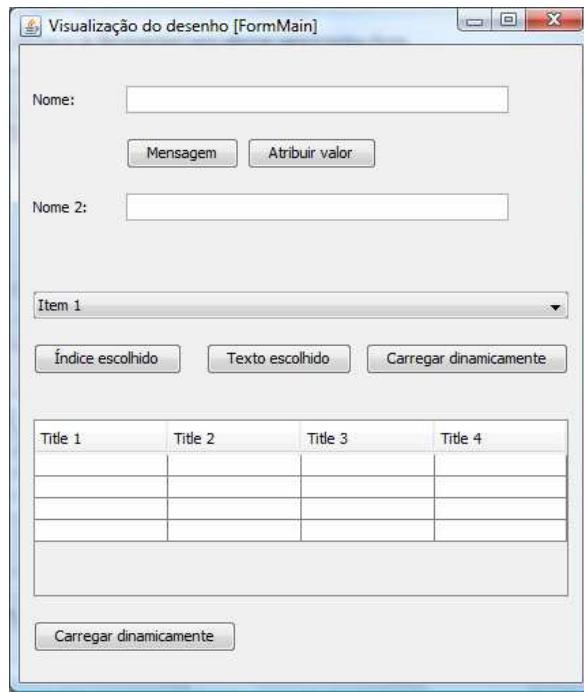


Figura 53 - Manipulando JTable - parte 1

A Figura 54 mostra o código-fonte para carregar informações dinamicamente no **JTable**, onde serão colocadas duas colunas: uma para o nome e outra para o sobre nome, e serão colocados 10 linhas na **JTable**.

```
private void jButtonTableDinamActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultTableModel model = new DefaultTableModel();
    model.setColumnIdentifiers(new String[]{
        "Nome", "Sobrenome"});
    try {
        for (int i = 0; i < 10; i++) {
            model.addRow(new Object[]{
                "Hildeberto " + i, "Melo " + i
            });
        }
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
    jTable1.setModel(model);
}
```

Figura 54 - Manipulando JTable - parte 2

A Figura 55 ilustra a execução do clique do **JButton** definido anteriormente. Onde na tela da esquerda aparece o **JTable** com a configuração padrão e na tela da direita aparece o **JTable** preenchido com duas colunas e dez linhas.

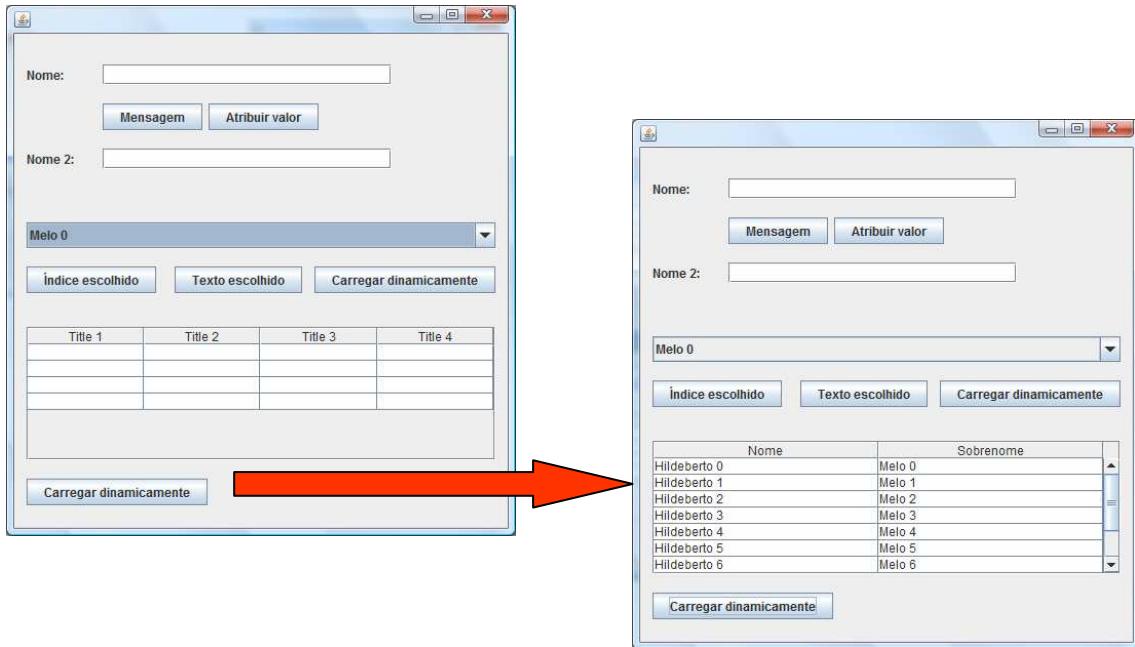


Figura 55 - Manipulando JTable - parte 3

Codificando a Parte Visual do Sistema de Vendas

Nosso sistema terá as seguintes telas:

- Tela principal, a partir desta tela serão chamadas às telas de:
 - Tela de consulta de cliente, a partir desta tela serão chamadas às telas de:
 - § Tela de cadastro de cliente
 - § Tela de alteração de cliente
 - Tela de consulta de produto, a partir desta tela serão chamadas às telas de:
 - § Tela de cadastro de produto
 - § Tela de alteração de produto
 - Tela de consulta de venda, a partir desta tela serão chamadas às telas de:
 - § Tela de cadastro de venda
 - § Tela de alteração de venda

Tela Principal do Sistema

Na tela principal teremos três botões, o primeiro para chamar a tela de consulta de clientes, o segundo para chamar a tela de consulta de produtos e o terceiro botão para chamar a tela de consulta de vendas.

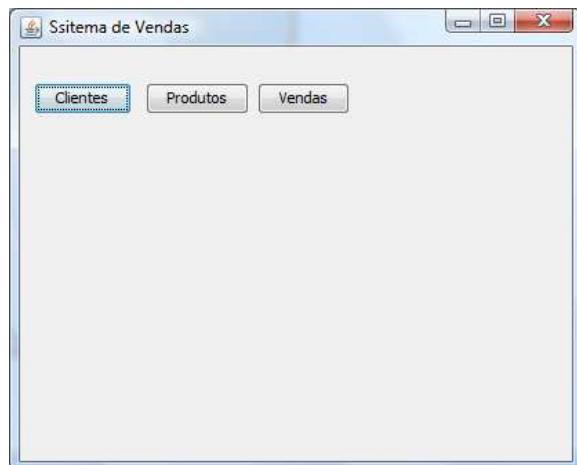


Figura 41 - Tela principal do Sistema de Vendas

Para as telas de consultas, cadastro e alteração, configurar a propriedade defaultCloseOperation de todas as telas para DISPOSE, com esta configuração quando for clicado o ícone de fechar (x) destas telas, não irá fechar o formulário principal.

Telas de Consultas

Tela de Consulta de Clientes

A tela de consulta de clientes possui uma tabela onde serão exibidos os clientes cadastrados. Também possuirá quatro botões:

- Listar – no evento do seu click todos os clientes cadastrados no sistema;
- Novo – no evento do seu click será aberta uma tela para ser cadastrado um novo cliente;
- Alterar – no evento do seu click será aberta uma tela para ser alterada as informações de um cliente previamente cadastrado;
- Remover – no evento do seu click será removido um cliente previamente cadastrado.

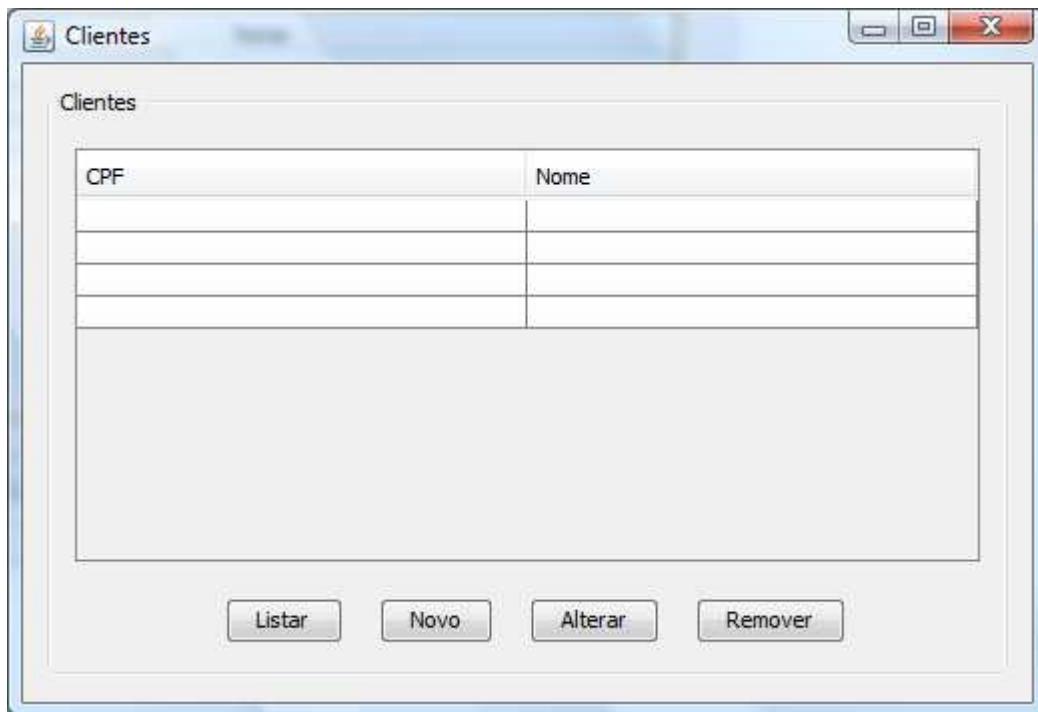


Figura 42 - Tela de Consulta de Clientes

Tela de Consulta de Produtos

Seguindo o padrão da tela de consulta de clientes, a tela de consulta de produtos possui uma tabela onde serão exibidos os produtos cadastrados. Também possuirá quatro botões:

- Listar – no evento do seu click todos os produtos cadastrados no sistema;
- Novo – no evento do seu click será aberta uma tela para ser cadastrado um novo produto;
- Alterar – no evento do seu click será aberta uma tela para ser alterada as informações de um produto previamente cadastrado;
- Remover – no evento do seu click será removido um produto previamente cadastrado.

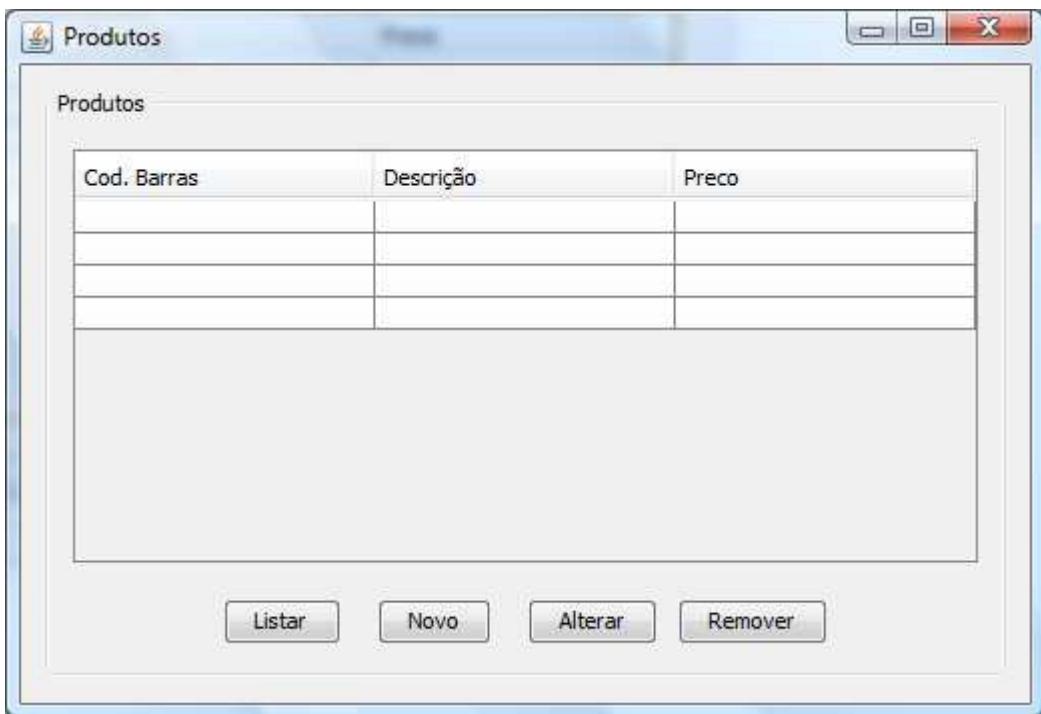


Figura 43 - Tela de Consulta de Produtos

Tela de Consulta de Vendas

Seguindo o padrão da tela de consulta de clientes e produtos, a tela de consulta de vendas possui duas tabelas, na primeira tabela serão exibidas as vendas cadastradas, e na segunda tabela os itens correspondentes à venda selecionada da primeira tabela. Também possuirá quatro botões:

- Listar – no evento do seu click todos os produtos cadastrados no sistema;
- Novo – no evento do seu click será aberta uma tela para ser cadastrado um novo produto;

- Remover – no evento do seu click será removido um produto previamente cadastrado.

The screenshot shows a Windows application window titled "Vendas". The main area contains a table with columns: Nº, Data, Cliente, and Total. Below this table are three buttons: "Listar", "Novo", and "Remover". A secondary section titled "Itens da venda" displays another table with columns: Produto, Qtd, and Preco. The entire application has a standard Windows-style interface with a title bar and a close button.

Figura 44 - Tela de Consulta de Vendas

Telas de Cadastro e Alteração

Tela de Cadastro e Alteração de Cliente

A tela de cadastro de cliente possuirá dois campos de texto, que servirão para informar o cpf e nome do cliente. E dois botões um para salvar as informações do cliente e outro para cancelar o cadastro do cliente.

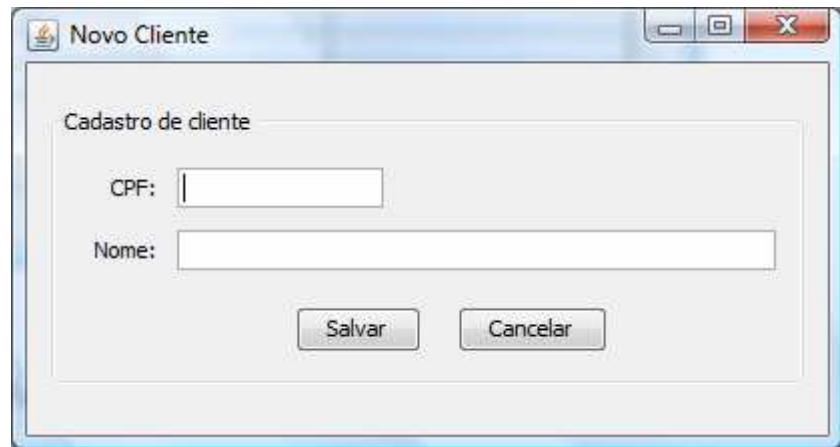


Figura 45 - Tela de Cadastro de Cliente

A tela de alteração de cliente possuirá dois campos de texto, que servirão para atualizar o cpf e nome do cliente. E dois botões um para salvar as informações do cliente e outro para cancelar a operação de alteração do cliente. Quando a tela for aberta, aparecerá automaticamente as informações do cliente a ser alterado.

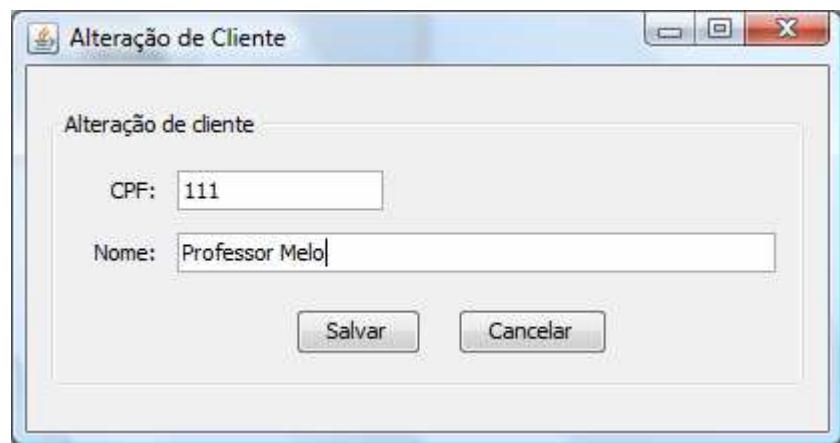


Figura 46 - Tela de Alteração de Cliente

Tela de Cadastro e Alteração de Produto

A tela de cadastro de produto possuirá três campos de texto, que servirão para informar o código de barras, descrição e preço do produto. E dois botões um para salvar as informações do produto e outro para cancelar o cadastro do produto.

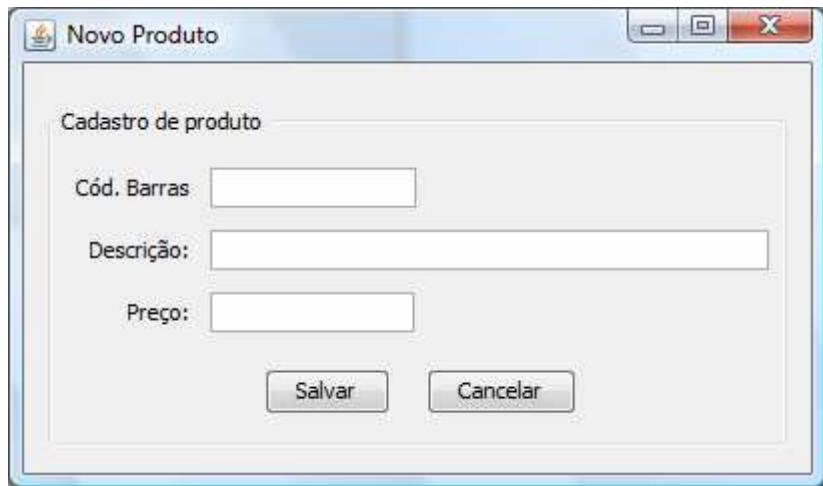


Figura 47 - Tela de Cadastro de Produto

A tela de alteração de produto possuirá três campos de texto, que servirão para atualizar o código de barras, descrição e preço do produto. E dois botões um para salvar as informações do produto e outro para cancelar a operação de alteração do produto. Quando a tela for aberta, aparecerá automaticamente as informações do produto a ser alterado.



Figura 48 - Tela de Alteração de Produto

Tela de Cadastro de Venda

A tela de cadastro de venda está dividida em duas partes. Na primeira parte ela possui dois campos de texto, que servirão para informar o número e a data da venda. E um combobox que mostrará todos os clientes cadastrados e servirá para escolher o cliente da venda.

Na segunda parte foi colocado uma tabela que mostrará os itens da venda, dois campos texto, um para a quantidade e outro para o preço, mais um combobox que listará todos

os produtos cadastrados. Teremos também dois botões uma para adicionar um item na venda e o outro para remover um item da venda.

E dois botões um para salvar as informações do produto e outro para cancelar o cadastro do produto.

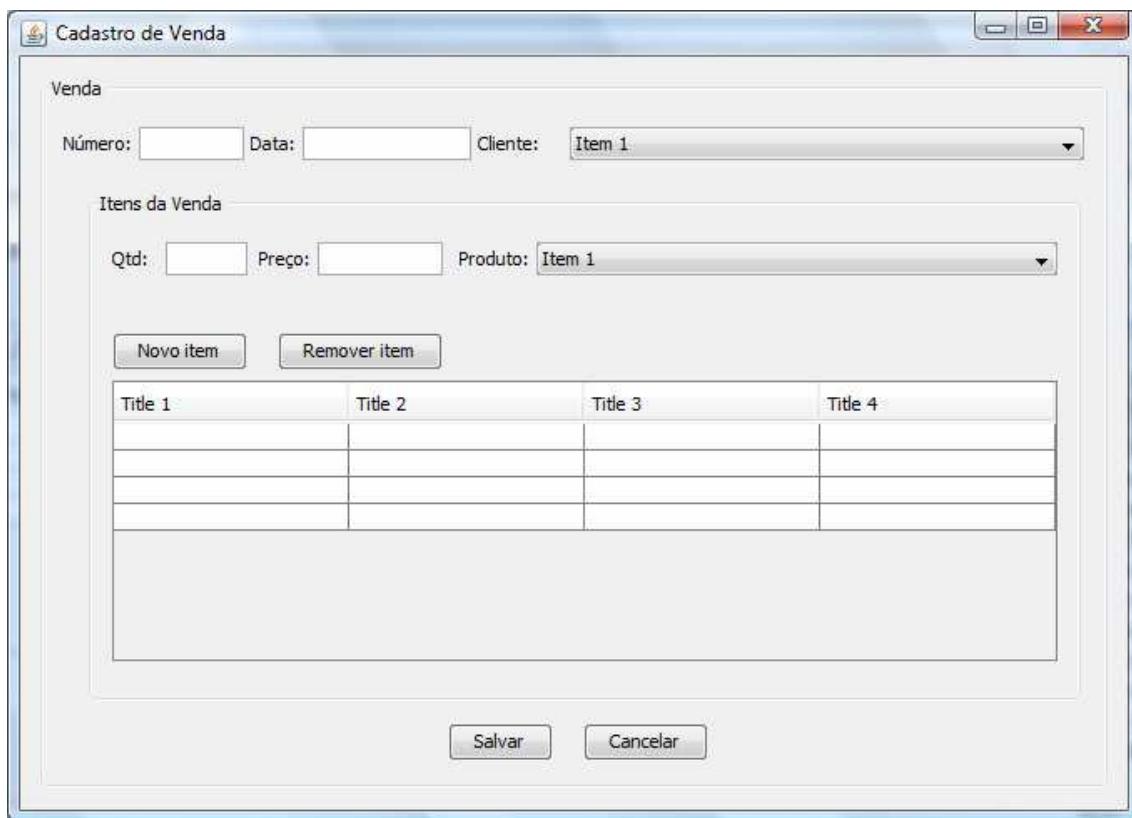


Figura 49 - Tela de Cadastro de Venda

Código fonte das telas

Métodos auxiliares

Centralizar frame

Este método tem por finalidade centralizar a tela no desktop

```
public void centralizeFrame() {
    int x, y;
    Rectangle scr = this.getGraphicsConfiguration().getBounds();
    Rectangle form = this.getBounds();
    x = (int) (scr.getWidth() - form.getWidth()) / 2;
    y = (int) (scr.getHeight() - form.getHeight()) / 2;
    this.setLocation(x, y);
}
```

Figura 50 - Método centralizar frame

Tela principal

A figura abaixo apresenta o código fonte do clique dos três botões, onde cada botão chamará a tela de consulta correspondente, por exemplo, o botão clientes chamará a tela de consulta de clientes.

```
private void jButtonClienteActionPerformed(java.awt.event.ActionEvent evt) {
    FormConsultaCliente form = new FormConsultaCliente();
    form.show();
}

private void jButtonProdutoActionPerformed(java.awt.event.ActionEvent evt) {
    FormConsultaProduto form = new FormConsultaProduto();
    form.show();
}

private void jButtonVendaActionPerformed(java.awt.event.ActionEvent evt) {
    FormConsultaVenda form = new FormConsultaVenda();
    form.show();
}
```

Figura 51 - Código Fonte tela principal

Telas do Cliente

Consulta de Clientes

A figura abaixo apresenta o código fonte do clique dos quatro botões, no caso dos botões de novo e alterar chamará a tela correspondente, também podemos observar os comandos para os botões listar e remover.

```
private void jButtonListarActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultTableModel modelo = new DefaultTableModel();
    modelo.setColumnIdentifiers(new String[]{"CPF", "Nome"});
    RepositorioCliente rep = RepositorioCliente.obterInstancia();
    this.lista = rep.listarTodos();
    for (int i = 0; i < lista.size(); i++) {
        modelo.addRow(new Object[]{this.lista.get(i).getCPF(), this.lista.get(i).getNome()});
    }
    jTable1.setModel(modelo);
}

private void jButtonNovoClienteActionPerformed(java.awt.event.ActionEvent evt) {
    FormCadastroCliente form = new FormCadastroCliente();
    form.show();
}

private void jButtonAlterarClienteActionPerformed(java.awt.event.ActionEvent evt) {
    //this.lista.get(jTable1.getSelectedRow()) serve para pegar um dos clientes que foi listado da jtable
    FormAlterarCliente form = new FormAlterarCliente(this.lista.get(jTable1.getSelectedRow()));
    form.show();
}

private void jButtonRemoverClienteActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        //this.lista.get(jTable1.getSelectedRow()) serve para pegar um dos clientes que foi listado da jtable
        RepositorioCliente.obterInstancia().remover(this.lista.get(jTable1.getSelectedRow()));
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage());
    }
}
```

Figura 52 - Código fonte tela consulta de clientes

Cadastro de Clientes

Código dos botões Cancelar e Salvar

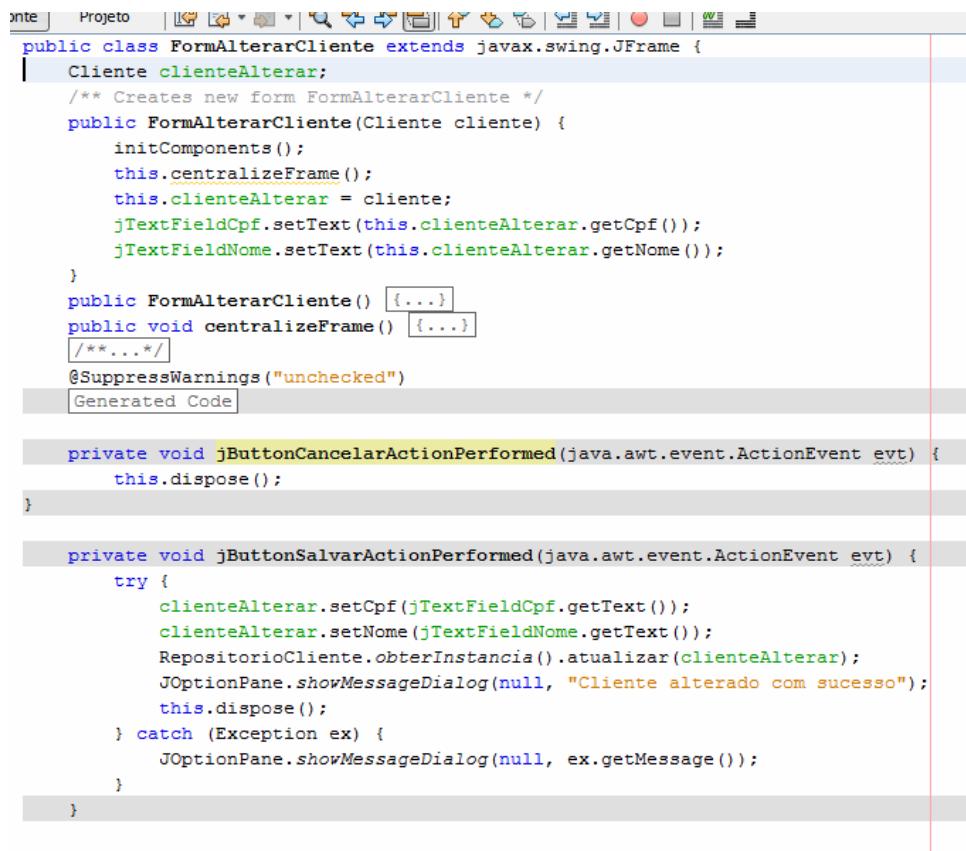
```
private void jButtonCancelarActionPerformed(java.awt.event.ActionEvent evt) {
    this.dispose();
}

private void jButtonSalvarActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        Cliente cliente = new Cliente();
        cliente.setCPF(jTextFieldCPF.getText());
        cliente.setNome(jTextFieldNome.getText());
        RepositorioCliente.obterInstancia().inserir(cliente);
        JOptionPane.showMessageDialog(null, "Cliente cadastrado com sucesso");
        this.dispose();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage());
    }
}
```

Figura 53 - Código fonte tela novo cliente

Alteração de Cliente

Para realizarmos a alteração de um cliente é necessário receber as informações do cliente a ser alterado. Para esta tarefa, redefinimos o construtor do formulário para que recebesse um atributo do tipo Cliente.



The screenshot shows a Java code editor with the following code:

```
public class FormAlterarCliente extends javax.swing.JFrame {
    Cliente clienteAlterar;
    /* Creates new form FormAlterarCliente */
    public FormAlterarCliente(Cliente cliente) {
        initComponents();
        this.centralizeFrame();
        this.clienteAlterar = cliente;
        jTextFieldCpf.setText(this.clienteAlterar.getCpf());
        jTextFieldNome.setText(this.clienteAlterar.getNome());
    }
    public FormAlterarCliente() {...}
    public void centralizeFrame() {...}
    /**...*/
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButtonCancelarActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
    }

    private void jButtonSalvarActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            clienteAlterar.setCpf(jTextFieldCpf.getText());
            clienteAlterar.setNome(jTextFieldNome.getText());
            RepositorioCliente.obterInstancia().atualizar(clienteAlterar);
            JOptionPane.showMessageDialog(null, "Cliente alterado com sucesso");
            this.dispose();
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage());
        }
    }
}
```

Figura 54 - Código fonte tela alterar cliente

Podemos observar que no construtor do formulário recebemos o cliente passado como parâmetro e em seguida colocamos as informações do cliente nos jtextfield correspondentes. Também podemos observar os comandos correspondentes aos cliques dos botões cancela e salvar.

Telas do Produto

Consulta de produtos

A figura abaixo apresenta o código fonte do clique dos quatro botões, no caso dos botões de novo e alterar chamará a tela correspondente, também podemos observar os comandos para os botões listar e remover.

```

private void jButtonListarActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultTableModel modelo = new DefaultTableModel();
    modelo.setColumnIdentifiers(new String[]{"Cod. Barras", "Descrição", "Preço"});
    this.lista = RepositorioProduto.obterInstancia().listarTodos();
    for (int i = 0; i < lista.size(); i++) {
        modelo.addRow(new Object[]{this.lista.get(i).getCodigoBarras(), this.lista.get(i).getDescricao(), this.lista.get(i).getPreco()});
    }
    jTable1.setModel(modelo);
}

private void jButtonNovoClienteActionPerformed(java.awt.event.ActionEvent evt) {
    FormCadastroProduto form = new FormCadastroProduto();
    form.show();
}

private void jButtonRemoverClienteActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        //this.lista.get(jTable1.getSelectedRow()) serve para pegar um dos produtos que foi listado da jTable
        RepositorioProduto.obterInstancia().remover(this.lista.get(jTable1.getSelectedRow()));
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage());
    }
}

private void jButtonAlterarClienteActionPerformed(java.awt.event.ActionEvent evt) {
    //this.lista.get(jTable1.getSelectedRow()) serve para pegar um dos produtos que foi listado da jTable
    FormAlterarProduto form = new FormAlterarProduto(this.lista.get(jTable1.getSelectedRow()));
    form.show();
}

```

Figura 55 - Código fonte tela consulta de produtos

Cadastro de produtos

Código dos botões **Cancelar** e **Salvar**

```

private void jButtonCancelarActionPerformed(java.awt.event.ActionEvent evt) {
    this.dispose();
}

private void jButtonSalvarActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        Produto produto = new Produto();
        produto.setCodigoBarras(jTextFieldCodBarras.getText());
        produto.setDescricao(jTextFieldDescricao.getText());
        produto.setPreco(Float.parseFloat(jTextFieldPreco.getText()));
        RepositorioProduto.obterInstancia().inserir(produto);
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage());
    }
}

```

Figura 56 - Código fonte tela novo produto

Alteração de produto

Para realizarmos a alteração de um produto é necessário receber as informações do produto a ser alterado. Para esta tarefa, redefinimos o construtor do formulário para que recebesse um atributo do tipo produto.

```

    Produto produtoAlterar;
    /** Creates new form FormCadastroCliente */
    public FormAlterarProduto() {...}
    public void centralizeFrame() {...}

    public FormAlterarProduto(Produto produto) {
        initComponents();
        this.centralizeFrame();
        this.produtoAlterar = produto;
        jTextFieldCodBarras.setText(this.produtoAlterar.getCodigoBarras());
        jTextFieldDescricao.setText(this.produtoAlterar.getDescricao());
        jTextFieldPreco.setText(" " + this.produtoAlterar.getPreco());
    }
    /**
     * @generated
     */
    @SuppressWarnings("unchecked")
    private void jButtonCancelarActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
    }

    private void jButtonSalvarActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            produtoAlterar.setCodigoBarras(jTextFieldCodBarras.getText());
            produtoAlterar.setDescricao(jTextFieldDescricao.getText());
            produtoAlterar.setPreco(Float.parseFloat(jTextFieldPreco.getText()));
            RepositorioProduto.obterInstancia().atualizar(produtoAlterar);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage());
        }
    }
}

```

Figura 57 - Código fonte tela alterar produto

Podemos observar que no construtor do formulário recebemos o produto passado como parâmetro e em seguida colocamos as informações do produto nos jTextField correspondentes. Também podemos observar os comandos correspondentes aos cliques dos botões cancela e salvar.

Telas do Venda

Consulta de vendas

A figura abaixo apresenta o código fonte do clique dos três botões, no caso dos botões de novo chamará a tela correspondente. Também podemos observar os comandos para os botões listar e remover.

```
private void jButtonListarActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultTableModel modelo = new DefaultTableModel();
    modelo.setColumnIdentifiers(new String[]{"Nº", "Data", "Cliente", "Total"});
    this.lista = RepositorioVenda.obterInstancia().listarTodos();
    for (int i = 0; i < lista.size(); i++) {
        modelo.addRow(new Object[]{this.lista.get(i).getNumero(),
        this.lista.get(i).getData(),
        this.lista.get(i).getCliente().getNome(),
        this.lista.get(i).getValorTotal()});
    }
    jTableVenda.setModel(modelo);
}

private void jButtonNovoClienteActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    FormCadastrarVenda form = new FormCadastrarVenda();
    form.show();
}

private void jButtonRemoverClienteActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        //this.lista.get(jTable1.getSelectedRow()) serve para pegar uma das vendas que foi listada da jtable
        RepositorioVenda.obterInstancia().remover(this.lista.get(jTableVenda.getSelectedRow()));
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage());
    }
}
```

Figura 58 - Código fonte tela consulta de vendas 1

A figura abaixo apresenta o código fonte correspondente à apresentação dos itens de um determinada venda. Neste caso quando selecionarmos a venda com o clique do mouse será executado este código.

```
private void jTableVendaMouseClicked(java.awt.event.MouseEvent evt) {
    DefaultTableModel modelo = new DefaultTableModel();
    modelo.setColumnIdentifiers(new String[]{"Produto", "Qtd", "Preço"});
    Venda venda = this.lista.get(jTableVenda.getSelectedRow());
    for (int i = 0; i < venda.getItensDaVenda().size(); i++) {
        modelo.addRow(new Object[]{venda.getItensDaVenda().get(i).getProduto().getDescricao(),
        venda.getItensDaVenda().get(i).getQuantidade(),
        venda.getItensDaVenda().get(i).getPreco()});
    }
    jTableItens.setModel(modelo);
}
```

Figura 59 - Código fonte tela consulta de vendas 2

Cadastro de Venda

A tela de cadastro de venda possui três atributos, sendo dois deles utilizados para listar os clientes e produtos cadastrados pelo sistema. E um atributo do tipo Venda, será neste atributo que colocaremos as informações manipuladas nesta tela.

```

public class FormCadastrarVenda extends javax.swing.JFrame {
    ArrayList<Cliente> listaCliente = RepositorioCliente.obterInstancia().listarTodos();
    ArrayList<Produto> listaProduto = RepositorioProduto.obterInstancia().listarTodos();
    Venda venda;
}

```

Figura 60 - Código fonte tela cadastro de vendas 1

O método carregarCombos() servirá para listar os produtos e clientes cadastrados em seus respectivos combobox. O método listarItens() serve para apresentar os itens desta venda na jTable dos itens desta venda.

```

public class FormCadastrarVenda extends javax.swing.JFrame {
    ArrayList<Cliente> listaCliente = RepositorioCliente.obterInstancia().listarTodos();
    ArrayList<Produto> listaProduto = RepositorioProduto.obterInstancia().listarTodos();
    Venda venda;
    /* Creates new form FormCadastrarVenda */
    public FormCadastrarVenda() {
        initComponents();
    }
    private void carregarCombos(){
        DefaultComboBoxModel modeloCli = new DefaultComboBoxModel();
        for (int i = 0; i < this.listaCliente.size(); i++) {
            modeloCli.addElement(this.listaCliente.get(i).getNome());
        }
        jComboBoxCliente.setModel(modeloCli);
        jComboBoxCliente.setSelectedIndex(-1);
        DefaultComboBoxModel modeloProd = new DefaultComboBoxModel();
        for (int i = 0; i < this.listaProduto.size(); i++) {
            modeloProd.addElement(this.listaProduto.get(i).getDescricao());
        }
        jComboBoxProduto.setModel(modeloProd);
        jComboBoxProduto.setSelectedIndex(-1);
    }
    public void listarItens(){
        DefaultTableModel modelo = new DefaultTableModel();
        modelo.setColumnIdentifiers(new String[]{"Produto", "Qtd", "Preço"});
        for (int i = 0; i < venda.getItensDaVenda().size(); i++) {
            modelo.addRow(new Object[]{venda.getItensDaVenda().get(i).getProduto().getDescricao(),
            venda.getItensDaVenda().get(i).getQuantidade(),
            venda.getItensDaVenda().get(i).getPreco()});
        }
        jTableItens.setModel(modelo);
    }
}

```

Figura 61 - Código fonte tela cadastro de vendas 2

A figura abaixo apresenta as ações para a escolha do produto no combobox dos produtos, e as ações para os botões remover item e novo item.

```

    }
    private void carregarCombos(){
        DefaultComboBoxModel modeloCli = new DefaultComboBoxModel();
        this.listaCliente = RepositorioCliente.obterInstancia().listarTodos();
        for (int i = 0; i < this.listaCliente.size(); i++) {
            modeloCli.addElement(this.listaCliente.get(i).getNome());
        }
        jComboBoxCliente.setModel(modeloCli);
        DefaultComboBoxModel modeloProd = new DefaultComboBoxModel();
        this.listaProduto = RepositorioProduto.obterInstancia().listarTodos();
        for (int i = 0; i < this.listaProduto.size(); i++) {
            modeloProd.addElement(this.listaProduto.get(i).getDescricao());
        }
        jComboBoxProduto.setModel(modeloProd);
    }
    public void listarItens(){
        DefaultTableModel modelo = new DefaultTableModel();
        modelo.setColumnIdentifiers(new String[]{"Produto", "Qtd", "Preço"});
        for (int i = 0; i < venda.getItensDaVenda().size(); i++) {
            modelo.addRow(new Object[]{venda.getItensDaVenda().get(i).getProduto().getDescricao(),
            venda.getItensDaVenda().get(i).getQuantidade(),
            venda.getItensDaVenda().get(i).getPreco()});
        }
        jTableItens.setModel(modelo);
    }
}

```

Figura 62 - Código fonte tela cadastro de vendas 3

A figura abaixo apresenta o código fonte para as ações para os botões de salvar e cancelar.

```

private void jButtonCancelarActionPerformed(java.awt.event.ActionEvent evt) {
    this.dispose();
}

private void jButtonSalvarActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // TODO add your handling code here:
        this.venda.setNumero(Integer.parseInt(jTextFieldNumero.getText()));
        this.venda.setData(jTextFieldData.getText());
        this.venda.getCliente().setCpf(this.listaCliente.get(jComboBoxCliente.getSelectedIndex()).getCpf());
        this.venda.getCliente().setNome(this.listaCliente.get(jComboBoxCliente.getSelectedIndex()).getNome());
        JOptionPane.showMessageDialog(null, "Venda cadastrada com sucesso");
        this.dispose();
        RepositorioVenda.obterInstancia().inserir(venda);
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage());
    }
}

```

Figura 63 - Código fonte tela cadastro de vendas 4

Trabalhando com Banco de Dados

Anteriormente vimos como armazenar as informações do sistema em um ArrayList. A desvantagem do ArrayList é o inconveniente de toda vez que fecharmos e abrirmos a aplicação novamente, teremos que refazer todos os cadastros.

Utilizando um banco de dados teremos a possibilidade de trabalhar com informações persistentes, em outras palavras, quando fecharmos a aplicação as informações estarão salvas e poderão ser reutilizadas pela aplicação novamente.

Vamos programar com acesso a bando de dados da seguinte forma, a conexão entre o bando de dados e o sistema será feita através do Jdbc do java.

Estrutura do Banco De Dados

Antes de começar a configurar uma conexão, será necessário criar o banco de dados para armazenar as informações do sistema de vendas. Utilizaremos a seguinte estrutura:

Tabela	Estrutura
Cliente	<pre>create table cliente(cpf varchar(20) primary key, Nome varchar(100),);</pre>
Produto	<pre>create table produto(codigoBarras varchar(20) primary key, descricao varchar(100), preco numeric(18,2));</pre>
Venda	<pre>create table venda(numero int primary key, data varchar(10), cpf_cliente varchar(20) references cliente(cpf));</pre>
Itens da venda	<pre>create table itensVenda(preco numeric(18,2), quantidade numeric(18,2), codigoBarras_produto varchar(20) references produto(codigoBarras), numero_venda int references venda(numero));</pre>

Configurando uma Conexão

Veremos como criar a conexão com o banco de dados.

- Windows 7:
 - Vá na pasta **c:\windows\sysWOW64**
 - Execute o aplicativo **odbcad32.exe**
- Windows Vista e nas versões anteriores:
 - Iniciar do Windows
 - Configurações
 - Painel de Controle
 - Ferramentas Administrativas
 - Fontes de Dados ODBC

Seguindo os passos descritos acima aparecerá esta tela, Para adicionar uma nova fonte clique no botão **Adicionar**

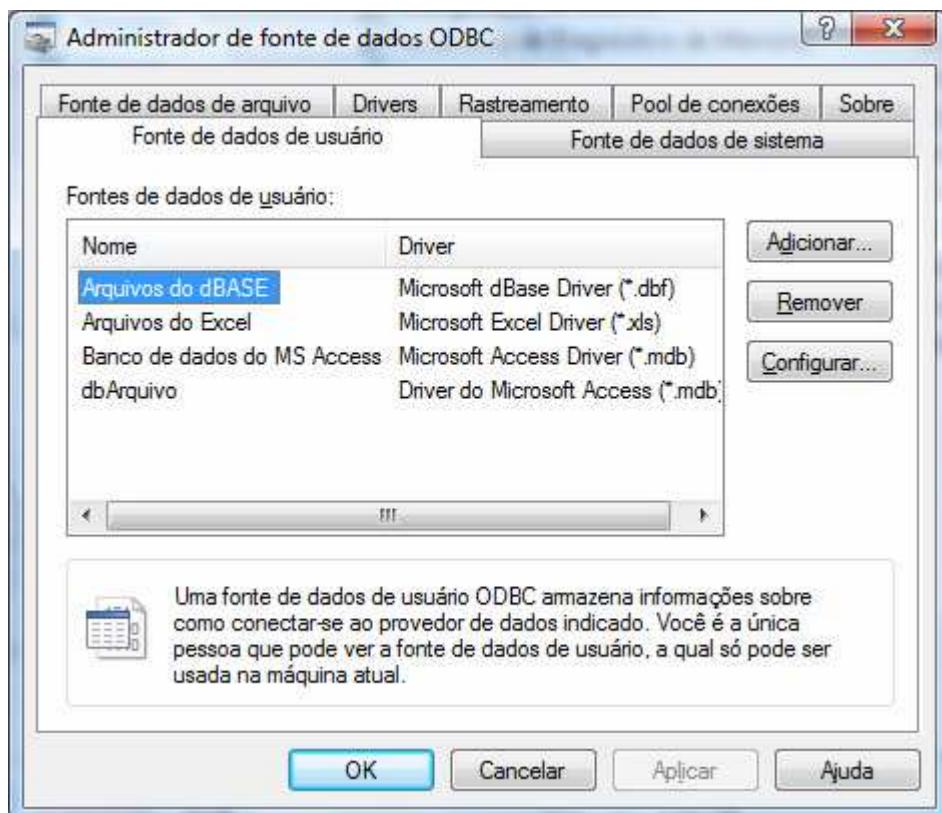


Figura 64 - Configurando a conexão com o banco de dados - 1

Aparecerá esta tela com os drivers disponíveis para conexão, vamos escolher o Driver do Microsoft Access (*.mdb) e clicar em concluir.

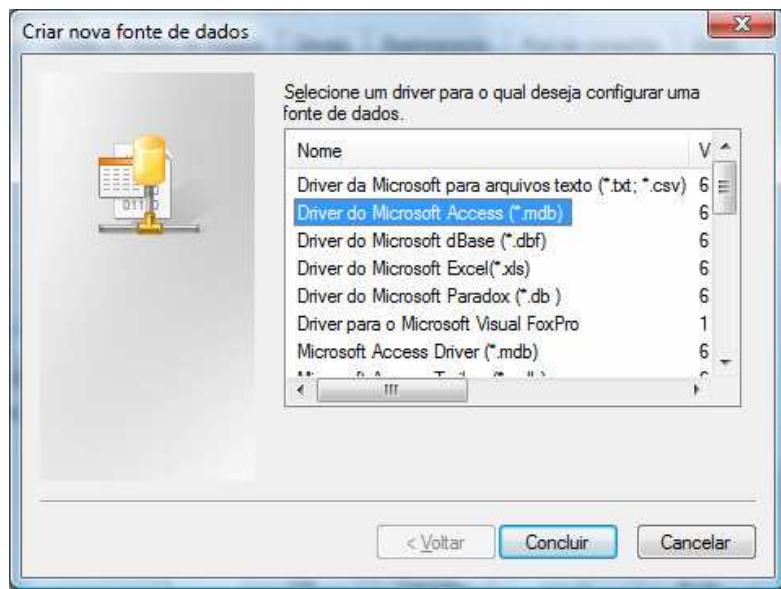


Figura 65 - Configurando a conexão com o banco de dados - 2

Informar o nome da fonte de dados, ex: **dbSistemaVendas**.



Figura 66 - Configurando a conexão com o banco de dados - 3

Selecionar o Banco de Dados, clicando no botão selecionar.

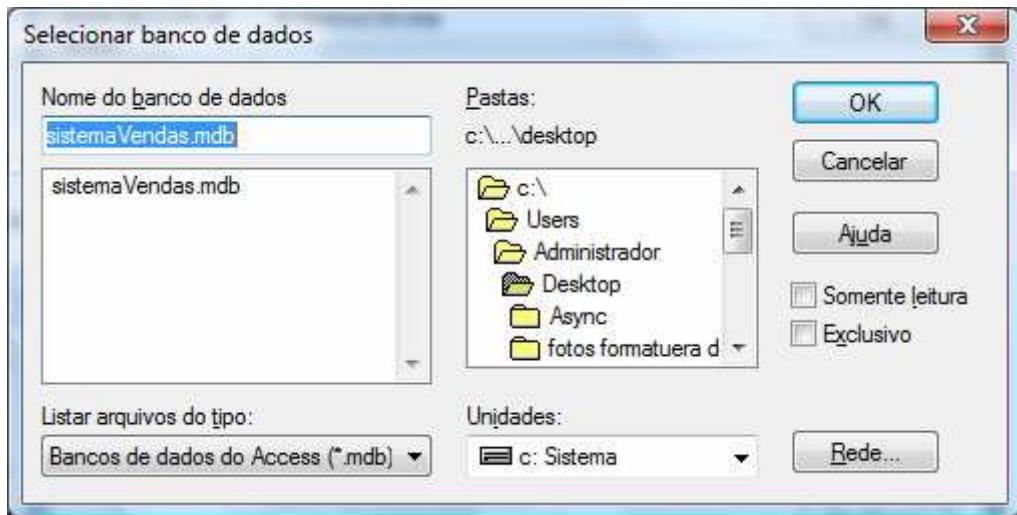


Figura 67 - Configurando a conexão com o banco de dados - 4

Depois de configurado, basta ir clicando nos botões “OK” para confirmar as operações.

Criando as Classes para Trabalhar com a Conexão Criada

Não basta apenas criar uma conexão com o banco de dados, teremos que criar os arquivos necessários para podermos trabalhar com esta conexão criada.

Classe de Conexão

Nesta classe temos dois métodos, o primeiro tem a finalidade de abrir uma conexão com o bando de dados, e a segunda em fechar esta conexão aberta.

Quando trabalhamos com banco de dados, o ideal é abrirmos uma conexão, executar o que se deseja e depois fecharmos esta conexão. Ex.: Abrir a conexão com o banco de dados, em seguida cadastrar um novo cliente, depois fechar a conexão.

```

import java.sql.*;

public class ConexaoOdbc {

    public Connection con;
    public Statement stm;

    public Statement conectar() throws Exception{
        // "Preparando para iniciar a conexao com o BD";
        try {
            /* Tenta se conectar ao Driver */
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            /* nome do banco que voce deu anteriormente ao seu alias */
            con = DriverManager.getConnection("jdbc:odbc:dbSistemaVendas");
            stm = con.createStatement();
        } catch (ClassNotFoundException e) { //capturando os erros da conexão
            throw new Exception("Erro conectar ao banco de dados: " + e.getMessage());
        } catch (SQLException sqle) {
            throw new Exception("Erro conectar ao banco de dados: " + sqle.getMessage());
        }
        return stm;
    }

    public void desconectar() {
        try {
            con.close();
        } catch (SQLException sqle) {
        }
    }
}

```

Figura 68 - Classe de Conexão com o banco de dados

Classe de Repositório

A classe de repositório com os comandos para manipulação do banco de dados, seguirá a mesma lógica da classe de repositório com ArrayList. Lembrando que para cada tipo de classe(ArrayList e Banco de dados) teremos que utilizar os respectivos comandos para seu devido manuseio.

Para cada método implementado foi seguida a seguinte lógica, primeiramente é realizada as validação necessárias para garantir a integridade e consistência das informações no banco de dados, depois é aberta a conexão com o banco de dados, em seguida é realizada a devida operação com o banco de dados, e por último é fechada à conexão com o banco de dados.

Estrutura da Classe RepositorioClienteOdbc

A figura abaixo ilustra a classe repositorioClienteOdbc, nela estão os métodos referentes à remoção, inserção, atualização e seleção dos clientes, bem como o construtor e o método obterInstancia. Esta classe também segue o padrão **singleton**.

```
[-] import java.sql.ResultSet;
    import java.sql.SQLException;
    import java.sql.Statement;
    import java.util.ArrayList;

    public class RepositorioClienteOdbc extends ConexaoOdbc {

        private static RepositorioClienteOdbc instancia;

        [-] private RepositorioClienteOdbc() {...}

        [+]public static RepositorioClienteOdbc obterInstancia() {...}

        [+]public void remover(Cliente cliente) throws Exception {...}

        [+]public void inserir(Cliente cliente) throws Exception {...}

        [+]public void atualizar(Cliente cliente) throws Exception {...}

        [+]public ArrayList<Cliente> listarTodos() throws Exception {...}
    }
```

Figura 69 - Estrutura da classe RepositorioClienteOdbc

Método remover

A figura abaixo ilustra a implementação do método remover, este método tem por finalidade excluir um determinado cliente.

```
public void remover(Cliente cliente) throws Exception {
    //validação das informações do cliente
    if (cliente == null) {
        throw new Exception("O cliente não foi instanciado");
    }
    if (cliente.getCpf() == null) {
        throw new Exception("Informar o CPF do cliente");
    }
    if (cliente.getCpf().trim().equals("")) {
        throw new Exception("Informar o CPF do cliente");
    }
    //abrindo a conexão
    Statement conex = conectar();
    //instrução sql correspondente a remoção do cliente
    String sql = "delete from cliente where cpf = "
        + cliente.getCpf() + "'";
    try {
        //executando a instrução sql
        conex.execute(sql);
    } catch (SQLException e) {
        //caso haja algum erro neste método será levantada esta exceção
        throw new Exception("Erro ao executar remoção: " + e.getMessage());
    }
    //fechando a conexão com o banco de dados
    desconectar();
}
```

Figura 70 - Método remover do RepositorioClienteOdbc

Método inserir

A figura abaixo ilustra a implementação do método inserir, este método tem por finalidade cadastrar um novo cliente.

```
public void inserir(Cliente cliente) throws Exception {
    //validação das informações do cliente
    if (cliente == null) {
        throw new Exception("O cliente não foi instanciado");
    }
    if (cliente.getCpf() == null) {
        throw new Exception("Informar o CPF do cliente");
    }
    if (cliente.getCpf().trim().equals("")) {
        throw new Exception("Informar o CPF do cliente");
    }
    if (cliente.getNome() == null) {
        throw new Exception("Informar o Nome do cliente");
    }
    if (cliente.getNome().trim().equals("")) {
        throw new Exception("Informar o Nome do cliente");
    }
    //abrindo a conexão
    Statement conex = conectar();
    //instrução sql correspondente a inserção do cliente
    String sql = "INSERT INTO cliente (nome, cpf)";
    sql += "VALUES ('" + cliente.getNome() + "', '" +
           cliente.getCpf() + "')";
    try {
        //executando a instrução sql
        conex.execute(sql);
    } catch (SQLException e) {
        //caso haja algum erro neste método será levantada esta exceção
        throw new Exception("Erro ao executar inserção: " + e.getMessage());
    }
    //fechando a conexão com o banco de dados
    desconectar();
}
```

Figura 71 - Método inserir do RepositorioClienteOdbc

Método atualizar

A figura abaixo ilustra a implementação do método atualizar, este método tem por finalidade cadastrar um novo cliente.

```
public void atualizar(Cliente cliente) throws Exception {
    //validação das informações do cliente
    if (cliente == null) {
        throw new Exception("O cliente não foi instanciado");
    }
    if (cliente.getCpf() == null) {
        throw new Exception("Informar o CPF do cliente");
    }
    if (cliente.getCpf().trim().equals("")) {
        throw new Exception("Informar o CPF do cliente");
    }
    if (cliente.getNome() == null) {
        throw new Exception("Informar o Nome do cliente");
    }
    if (cliente.getNome().trim().equals("")) {
        throw new Exception("Informar o Nome do cliente");
    }
    //abrindo a conexão
    Statement conex = conectar();
    //instrução sql correspondente a atualização do cliente
    String sql = "update cliente set " + " nome = '" + cliente.getNome()
        + "', " + " cpf = '" + cliente.getCpf() + "'"
        + " where cpf = '" + cliente.getCpf() + "'";
    try {
        //executando a instrução sql
        conex.execute(sql);
    } catch (SQLException e) {
        //caso haja algum erro neste método será levantada esta exceção
        throw new Exception("Erro ao executar atualização: " + e.getMessage());
    }
    //fechando a conexão com o banco de dados
    desconectar();
}
```

Figura 72 - Método atualizar do RepositorioClienteOdbc

Método listar todos

A figura abaixo ilustra a implementação do método listarTodos, este método tem por finalidade retornar todos os clientes cadastrados.

```
public ArrayList<Cliente> listarTodos() throws Exception {  
    //abrindo a conexão  
    Statement conex = conectar();  
    ArrayList<Cliente> retorno = new ArrayList<Cliente>();  
    //instrução sql correspondente a seleção dos clientes  
    String sql = "SELECT cpf, nome FROM cliente order by nome";  
    try {  
        //executando a instrução sql  
        ResultSet rs = conex.executeQuery(sql);  
        while (rs.next()) {  
            Cliente cliente = new Cliente();  
            cliente.setNome(rs.getString("nome"));  
            cliente.setCpf(rs.getString("matricula"));  
            retorno.add(cliente);  
        }  
    } catch (SQLException e) {  
        //caso haja algum erro neste método será levantada esta exceção  
        throw new Exception("Erro ao executar consulta: " + e.getMessage());  
    }  
    //fechando a conexão com o banco de dados  
    desconectar();  
    return retorno;  
}
```

Figura 73 - Método listarTodos do RepositorioClienteOdbc

Modificando as Telas para Trabalhar Com Banco de Dados

As mudanças são poucas e muito simples de fazer, basta chamar o método correspondente à operação desejada da Classe **RepositorioClienteOdbc**. A figura abaixo ilustra a mudança necessária, foi comentada a linha de comando que utilizava a Classe **RepositorioCliente** e foi adicionada uma linha e comando que utiliza a Classe **RepositorioClienteOdbc**.

```
private void jButtonSalvarActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        Cliente cliente = new Cliente();  
        cliente.setCpf(jTextFieldCpf.getText());  
        cliente.setNome(jTextFieldNome.getText());  
        //RepositorioCliente.obterInstancia().inserir(cliente);  
        RepositorioClienteOdbc.obterInstancia().inserir(cliente);  
        JOptionPane.showMessageDialog(null, "Cliente cadastrado com sucesso");  
        this.dispose();  
    } catch (Exception ex) {  
        JOptionPane.showMessageDialog(null, ex.getMessage());  
    }  
}
```

Figura 74 - Mudança na GUI para utilizar banco de dados

Considerações Finais

Foram utilizadas apenas as rotinas básicas e necessárias para a confecção deste sistema, bem como as validações utilizadas para o manuseio do sistema. Vale salientar que cada sistema de venda possui suas regras de negócio específicas, portanto para adaptar este sistemas as necessidades de uma empresa, será necessário adaptar as validações para que o mesmo mantenha a integridade e consistência das informações.

Foi codificado apenas a classe correspondente à manipulação das informações do cliente com banco de dados, para as demais classes (produtos e venda) basta seguir o exemplo da Classe **RepositorioClienteOdbc**.