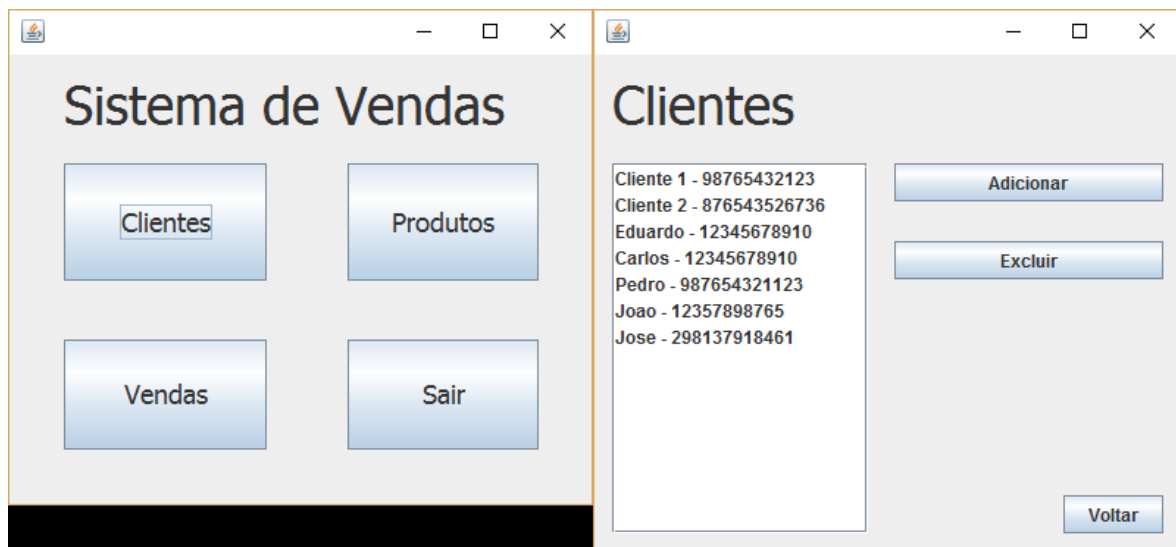


SISTEMA DE CADASTRO EXEMPLO COM NETBEANS E MYSQL COM INTERFACE



Criando um novo projeto

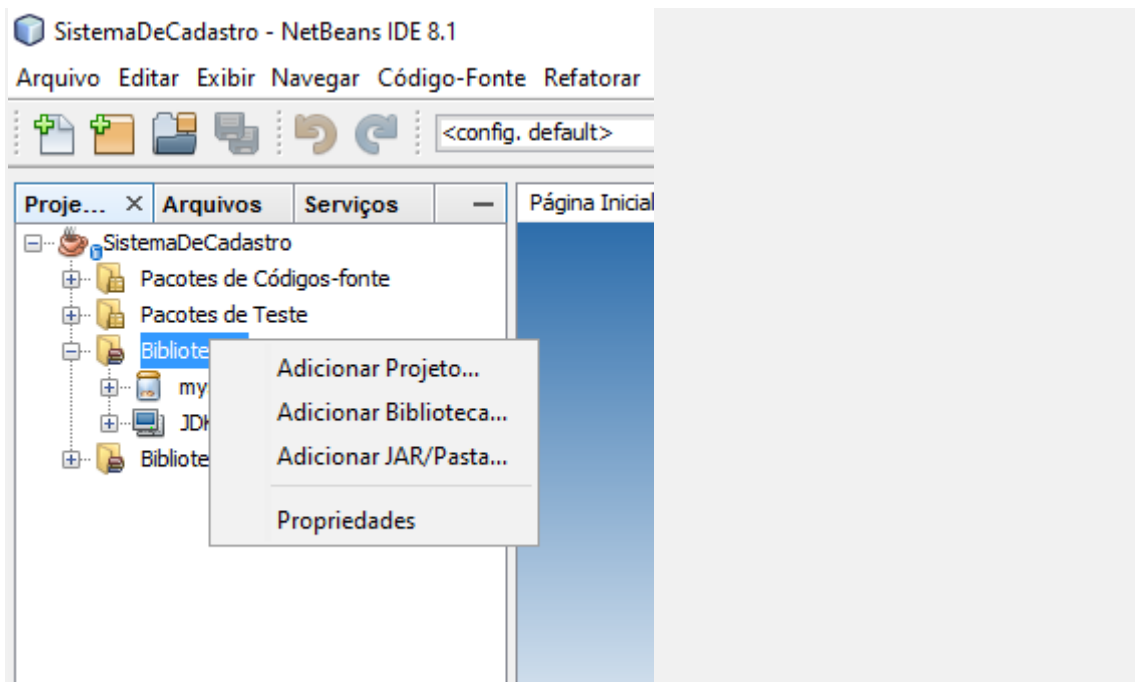
Vamos criar uma nova aplicação Java no Netbeans. A aplicação será chamada de SistemaDeCadastro.

Instalando o conector de MySQL para Netbeans

O primeiro elemento necessário para realizar a conexão de Java com MySQL é um conector que pode ser encontrado no [link a seguir](#).

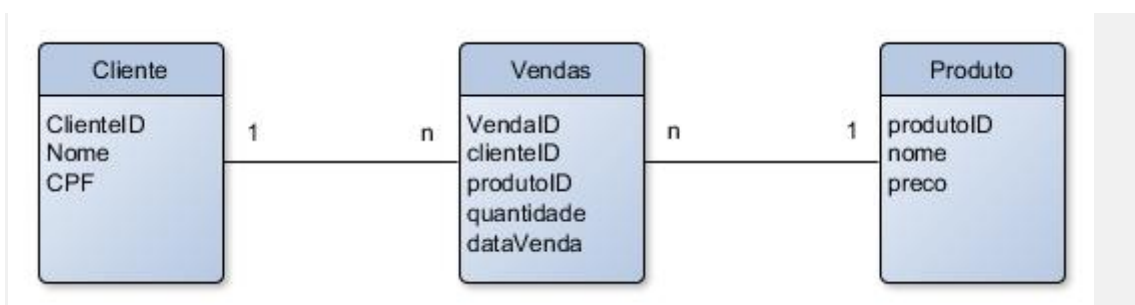
Por padrão, Java não consegue conversar com MySQL, portanto é necessário uma biblioteca (Um conjunto de códigos para realizar uma tarefa específica) para poder se comunicar com MySQL.

Após realizar o download do arquivo .jar vamos adicioná-lo ao projeto. Para isso, clique com o botão direito em Bibliotecas -> Adicionar JAR/Pasta.... A seguir, localize o arquivo .jar e o adicione.



Modelando nosso cenário

Antes de começar a codificar, temos que entender o problema e os elementos que precisaremos ter em nosso banco de dados. No cenário que trabalharemos, faremos um sistema de vendas bem simples. Teremos clientes que podem comprar produtos de uma loja. Portanto, para nosso banco de dados precisaremos de 3 tabelas conforme o Modelo de Entidade e Relacionamento (MER) abaixo.



O seguinte modelo não é o ideal. No caso específico a tabela de vendas seria dividida em duas: tabela de vendas e as linhas da venda. Como o propósito desse post é ensinar a questão de conexão do Java com banco de dados e métodos simples de inserção, atualização e remoção, não entraremos nesse mérito. A tabela de clientes pode utilizar o CPF como chave primária, uma vez que cada pessoa contém um único número de CPF, mas por questões de simplificação, CPF está como atributo e adicionamos uma chave primária que é incrementada automaticamente.

O seguinte código em SQL cria o banco de dados chamado loja, cria as 3 tabelas e cadastra com algumas informações.

```
1
2
3   create SCHEMA if not exists loja_example;
4
5   CREATE DATABASE if not exists loja;
6
7   USE loja;
8
9   /* DELETAR TABELAS */
10  DROP TABLE venda;
11  DROP TABLE produto;
12  DROP TABLE cliente;
13
14  CREATE TABLE if not exists cliente
15  (
16      clienteID integer NOT NULL AUTO_INCREMENT,
17      nome varchar(200),
18      cpf varchar(12) NOT NULL,
19      constraint chave_cliente PRIMARY KEY (clienteID)
20  );
21
22  CREATE TABLE if not exists produto
23  (
24      produtoID integer NOT NULL AUTO_INCREMENT,
25      nome varchar(200),
26      preco double NOT NULL,
27      CONSTRAINT chave_produto PRIMARY KEY (produtoID)
28  );
29
30  CREATE TABLE if not exists venda
31  (
32      vendaID integer NOT NULL AUTO_INCREMENT,
33      clienteID integer NOT NULL,
34      produtoID integer NOT NULL,
35      quantidade integer NOT NULL,
36      dataVenda date NOT NULL,
37      CONSTRAINT chave_venda PRIMARY KEY (vendaID),
38      FOREIGN KEY(clienteID) REFERENCES cliente(clienteID),
39      FOREIGN KEY(produtoID) REFERENCES produto(produtoID)
40  );
41
42  /* INSERIR ALGUNS DADOS */
43  INSERT INTO cliente (nome, cpf) VALUES ('Cliente 1','98765432123');
44  INSERT INTO cliente (nome, cpf) VALUES ('Cliente 2', '876543526736');
45
46  INSERT INTO produto (nome, preco) VALUES('Produto 1', '15.00');
47  INSERT INTO produto (nome, preco) VALUES('Produto 2', '8.00');
48
49  INSERT INTO venda (clienteID, produtoID, quantidade, dataVenda) VALUES
50  ('1', '1', '2', '2016-04-06');
51  INSERT INTO venda (clienteID, produtoID, quantidade, dataVenda) VALUES
52  ('2', '1', '1', '2016-04-06');
53  INSERT INTO venda (clienteID, produtoID, quantidade, dataVenda) VALUES
54  ('2', '2', '3', '2016-04-06');
```

Fazendo a ligação com Java

Após ter a tabela criada, vamos criar uma classe em Java no Netbeans para criar a conexão com o banco de dados.

```
1    package sistemadecadastro;
2
3    import java.sql.Connection;
4    import java.sql.DriverManager;
5
6    /**
7     * @author Faqi
8     */
9    public class MinhaConexao {
10
11        //objeto estático para guardar uma instância de minha conexão
12        public static MinhaConexao conexao = null;
13
14        //método estático para criar uma instância do objeto
15        //MinhaConexao
16        public static MinhaConexao getInstance(){
17            try{
18                //verifica se já existe uma conexão. Isso é feito verificando
19                //se há algum objeto atribuído à conexao ou se a conexao sql
20                //atribuída à ele está fechada.
21                if(conexao==null || conexao.sqlConnection.isClosed()){
22                    conexao = new MinhaConexao(); //cria uma nova conexão caso não
23                                                    //exista uma.
24                }
25            }catch(Exception e){
26                e.printStackTrace();
27            }
28            return conexao;
29        }
30
31        //cria um objeto Connection chamado sqlConnection
32        public Connection sqlConnection;
33
34        //construtor para inicializar a conexão
35        private MinhaConexao()
36        {
37            try{
38
39                //cria uma nova instancia utilizando o driver que foi adicionado
40                //à biblioteca através do arquivo .jar
41                Class.forName("com.mysql.jdbc.Driver").newInstance();
42
43                //define a string de conexão com o banco de dados MySQL.
44                //Lembrando que meuUsuario e minhaSenha devem ser substituídos
45                //pelo usuário e senha utilizados para conectar com o banco de
46                dados.
47                String textoConexao =
48                "jdbc:mysql://localhost/loja?user=meuUsuario&password=minhaSenha";
49            }
50        }
51    }
```

```

44 //adquire a conexão
45     sqlConnection = DriverManager.getConnection(textoConexao);
46 }catch(Exception e){
47     e.printStackTrace();
48 }
49 }
50
51
52
53
54
55

```

A chave nessa classe é a string de conexão “jdbc:mysql://localhost/loja?user=meuUsuario&password=minhaSenha”, pois ela contém as informações do banco de dados que está sendo conectado:

localhost é o endereço onde o banco de dados está rodando. No caso específico, localhost direciona para o endereço local.

loja é o nome do banco de dados que está sendo utilizado.

meuUsuario e minhaSenha representam o usuário e senha utilizados para conectar ao banco de dados.

Revisão de Classes e Objetos

Faremos uma simples revisão do que são classes e objetos em Java para aqueles que nunca viram ou para aqueles que fazem tempo que não utilizam.

Classes são estruturas que dão direção de como criar um objeto. Um objeto é um elemento criado baseado na estrutura definida pela classe.

Um objeto é composto por dois elementos:

Atributos: atributos são características que guardam informações do objeto.

Métodos: são ações executadas pelo objeto.

Vamos tomar um carro como exemplo. Se quisermos criar uma classe chamada carro, iremos definir alguns parâmetro para esse carro, ou seja, características que descrevem o carro. São exemplos de atributos:

cor

placa

velocidade

aceleração

para-brisa

tempo

Cada atributo é de um tipo de dado. Alguns tipos de dados padrões de Java são: int, double, String, Boolean. Os atributos acima podem ser classificados conforme os atributos de forma que eles possam ser melhor representados:

String cor – a cor será salva como uma palavra (“vermelho”, “amarelo”, “verde”, etc). Portanto será do tipo String.

String placa – uma placa de carro (pelo menos no Brasil), é composta por letras e números, ou seja, uma palavra. (“ABC 1234”, “DEV 2016”, etc). Portanto será do tipo String.

double velocidade – a velocidade é um número e que geralmente não é inteiro. É possível estar a 8.5 km/h. Quando falamos a velocidade que estamos, geralmente arredondamos para inteiros, entretanto a velocidade real pode não ser inteira. Por conta disso, seu tipo será inteiro.

double aceleração – da mesma forma que a velocidade, aceleração não é um número inteiro, portanto, será considerado um double.

bool para-brisa – nesse caso, utilizaremos essa variável para indicar se o parabrisa está ligado ou desligado. Bool é uma abreviação para boolean, que significa binário. Essa variável pode conter apenas dois valores: True ou False (Verdadeiro ou Falso). Portanto, bool é um bom tipo para essa variável.

int tempo – no caso do problema, apenas os segundos contam, portanto não é necessário contabilizar segundos como um número decimal. Nesse caso, o tempo será contado como a quantidade inteira de segundos. Portanto o tipo dessa variável pode ser inteiro.

Claro que a maneira como você modela a classe depende de do seu problema. Se formos considerar que o para-brisa não apenas liga e desliga, mas tem 3 velocidades (lerdo, médio e rápido), não será possível representar apenas por verdadeiro ou falso. Nesse caso, pode ser que a representação seja feita por inteiros, onde 0 representa desligado, 1 representa a velocidade letra, 2 médio e 3 rápido. Outra possibilidade de representação é por caracteres, onde ‘d’ pode significar desligado, ‘l’ pode representar a velocidade lerda, ‘m’ média, e ‘r’ rápido. Como visto, o tipo de representação depende do que se quer modelar e como os dados serão trabalhados.

Os métodos são funções realizadas pelo objeto. No nosso exemplo, um carro pode por exemplo ligar o para-brisa, acelerar, querer saber sua velocidade dado um certo tempo e uma aceleração, ou saber qual a distância percorrida dada uma certa velocidade e um certo tempo.

Exemplos:

```
1      //ligar parabrisa
2
3      public void ligaParaBrisas() {
4
5          parabrisa = True;
6      }
```

```
7
8    //desligar parabrisa
9
10   public void desligarParaBrisa() {
11
12       parabrisa = False
13
14   }
15
16   //acelerar o carro
17
18   public void acelerar(double addAceleracao) {
19
20       aceleracao = aceleracao + addAceleracao
21
22   }
23
24   //conseguir velocidade apos um certo tempo com a aceleração atual
25
26   public double getVelocidade(int tempo) {
27
28       velocidade_final = velocidade + aceleracao*tempo;
29       return velocidade_final;
30   }
31
32
```

Os métodos acima podem alterar os atributos sem que seja passado nenhum parâmetro (ligarParaBrisa, desligarParaBrisa), alterar atributos utilizando parâmetros (acelerar), ou mesmo retornar novos valores que podem ser atribuídos à variáveis (getVelocidade).

Criando objetos de nosso sistema

Para nosso sistema, serão criados 3 classes para criar objetos que receberam as informações das tabelas do banco de dados. Dessa maneira, será possível ler os objetos e trabalhar com eles em memória em uma estrutura parecida com a das tabelas.

Classe cliente

```
1    package sistemadecadastro;
2
3    /**
4     *
5     * @author Faqi
6     */
7    public class Cliente {
8
9        private String nome;
10       private String cpf;
11
12       public Cliente(String nome, String cpf){
13
14           this.nome = nome;
15           this.cpf = cpf;
16       }
17
18       public void setNome(String nome){
19
20           this.nome = nome;
21       }
22
23       public String getNome(){
24
25           return this.nome;
26       }
27
28       public void setCPF(String cpf){
29
30           this.cpf = cpf;
31       }
32   }
```



```
26
27     public String getCPF() {
28         return this.cpf;
29     }
30
31     }
32
33
```

A classe cliente contém os atributos Nome e CPF. Não iremos colocar o ClienteID pois não terá utilidade para o que faremos.

Como métodos temos os métodos Set e Get de cada atributo para adquirir ou modificar esse atributo. A mesma coisa se aplica para as classes Produto e Vendas.

Classe Produto

```
1     package sistemadecadastro;
2
3     /**
4      *
5      * @author Faqi
6      */
7
8     public class Produto {
9
10
11     private String nome;
12     private double preco;
13
14
15     public Produto(String nome, double preco) {
16         this.nome = nome;
17         this.preco = preco;
18     }
19
20     public void setNome(String nome) {
21         this.nome = nome;
22     }
23
24 }
```

```
18
19     public String getNome() {
20         return this.nome;
21     }
22
23     public void setPreco(double preco) {
24         this.preco = preco;
25     }
26
27     public double getPreco() {
28         return this.preco;
29     }
30 }
31
32
33
```

Classe Vendas

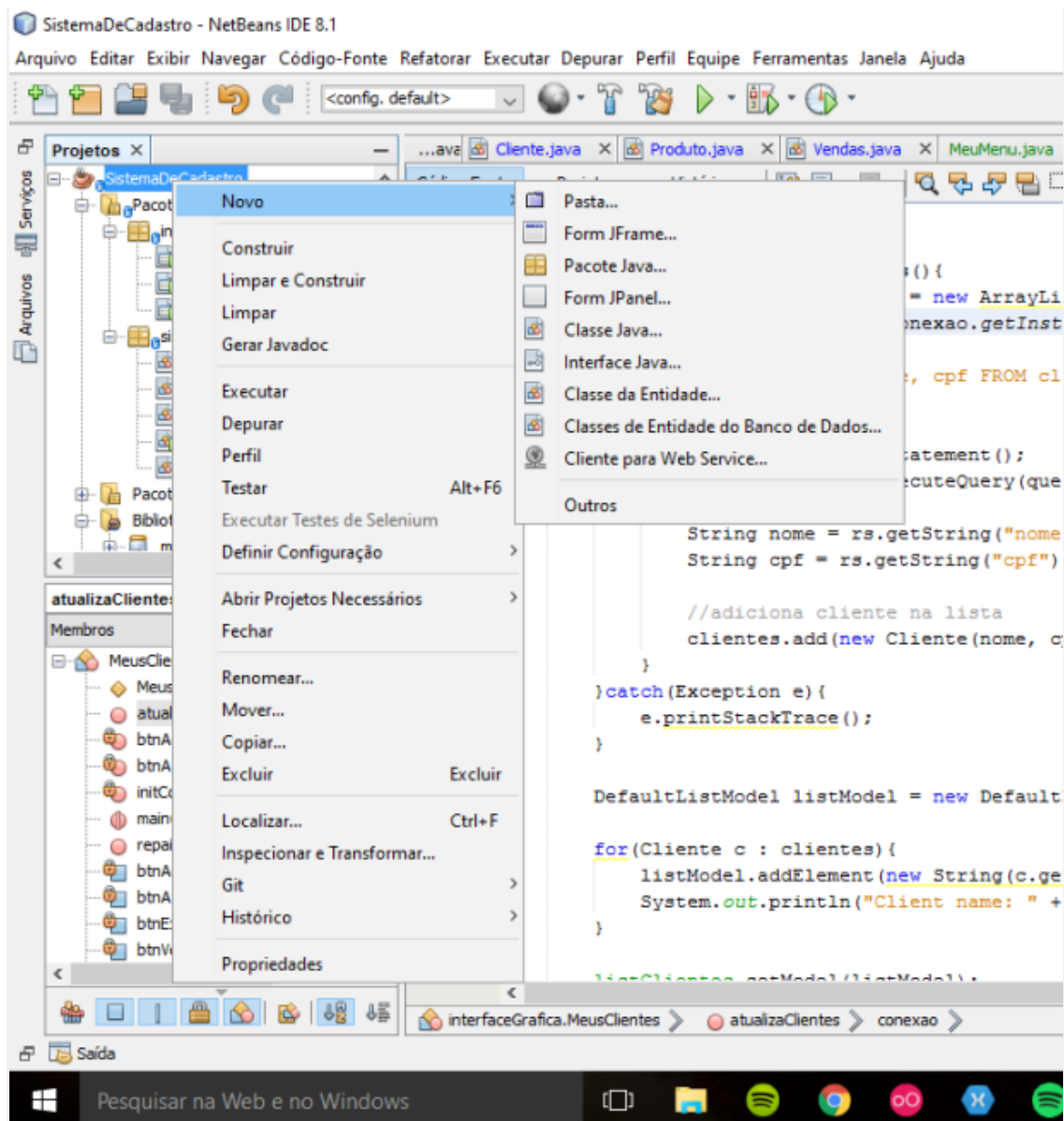
```
1     package sistemadecadastro;
2
3     /**
4      *
5      * @author Faqi
6      */
7     public class Vendas {
8
9         int clienteID;
10        int produtoID;
11        int quantidade;
12        String date;
13
14        public Vendas(int clienteID, int produtoID, int quantidade, String date) {
```

```
14     this.clienteID = clienteID;
15     this.produtoID = produtoID;
16     this.quantidade = quantidade;
17     this.date = date;
18 }
19
20 public int getClienteID() {
21     return clienteID;
22 }
23
24 public void setClienteID(int clienteID) {
25     this.clienteID = clienteID;
26 }
27
28 public int getProdutoID() {
29     return produtoID;
30 }
31
32 public void setProdutoID(int produtoID) {
33     this.produtoID = produtoID;
34 }
35
36 public int getQuantidade() {
37     return quantidade;
38 }
39
40 public void setQuantidade(int quantidade) {
41     this.quantidade = quantidade;
42 }
43
44 public String getDate() {
45     return date;
46 }
```

```
45
46     public void setDate(String date) {
47         this.date = date;
48     }
49
50     }
51
52
53
```

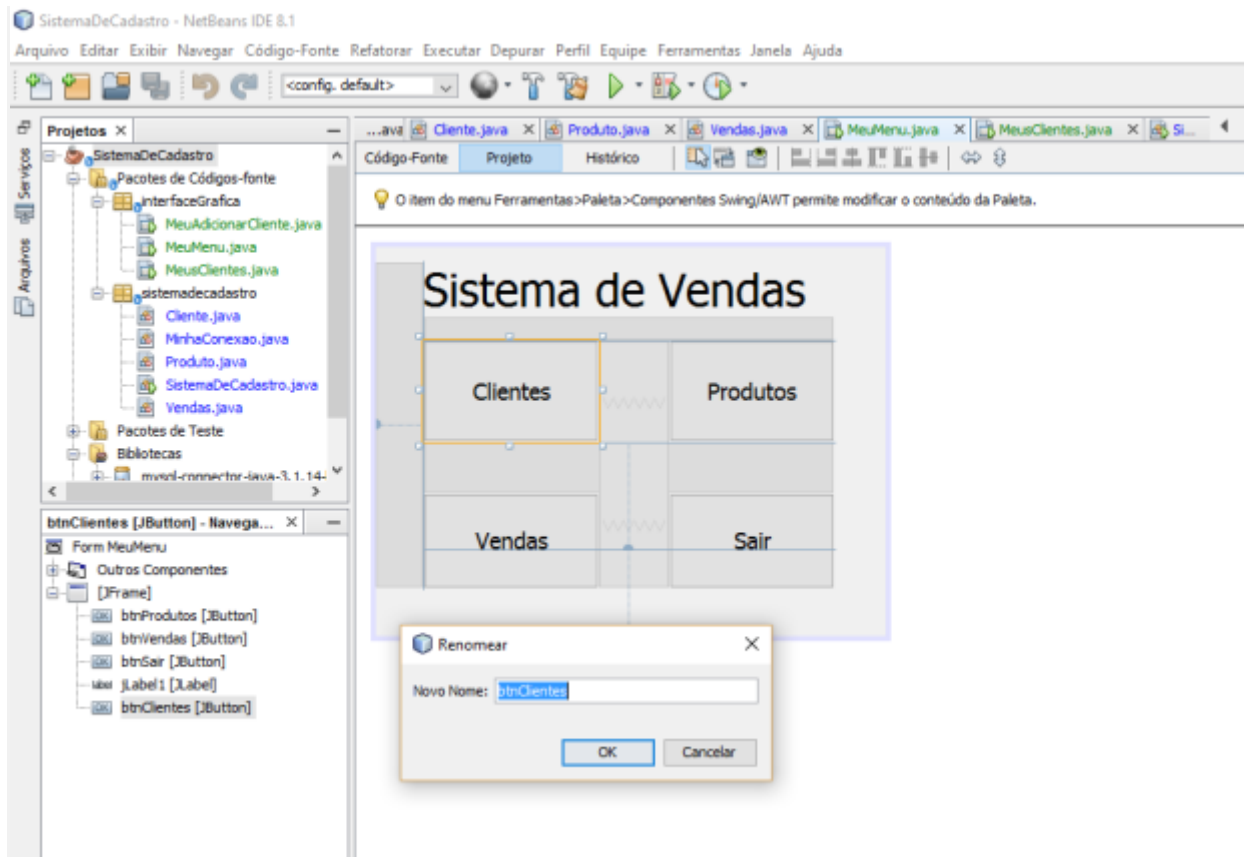
Criando a interface gráfica

Para criar a interface, clicamos com o botão direito no nome do Projeto -> Novo -> Form JFrame...



Crie uma janela chamada MeuMenu.

Será aberto uma janela onde componentes podem ser arrastados para a janela. Para o exemplo, foram adicionados 4 botões, mas apenas um é utilizado.



Para alterar o texto, clicar com o botão direito no botão e selecionar “Alterar Texto”.

Para alterar o nome da variável, clicar com o botão direito no botão e selecionar “Alterar o Nome da Variável”.

Dessa mesma maneira, crie mais dois JFrames: MeusClientes e MeusAdicionarCliente. Os modelos são mostrados nas imagens abaixo.



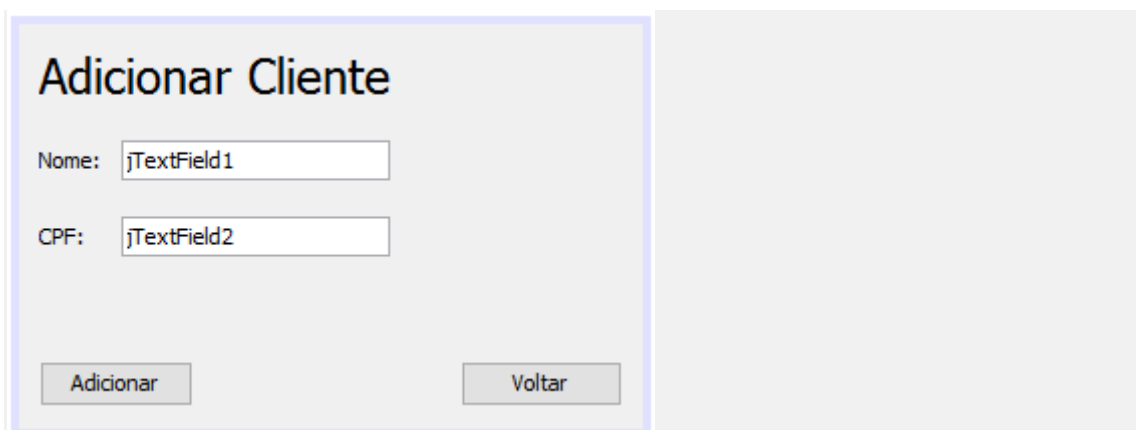
Os nomes importantes nessa janela são:

A lista de clientes (elemento que tem os textos Item 1, Item 2, etc, que chamará listClientes;

Botão de Adicionar, que chamará btnAddContato;

Botão de Excluir, que chamará btnExcluirContato;

Botão de Voltar, que chamará btnVoltar.



Os nomes importantes nessa janela são:

- jTextField1, que terá o nome txtNome;
- jTextField2, que terá o nome txtCPF;
- Botão Adicionar, que terá o nome btnAdicionar;
- Botão Voltar, que terá o nome btnVoltar.

Implementando a Interface Gráfica

SistemaDeCadastro.java

Primeiramente temos que chamar o MeuMenu da primeira classe que é iniciada no programa. A primeira classe é a classe que é criada no início e geralmente tem o mesmo nome do projeto. No caso do exemplo, a classe chama SistemaDeCadastro.java.

Para iniciar o JFrame de menu, basta declará-lo:

```
1  public static void main (String[] args) {
2
3  MeuMenu mm = new MeuMenu();
4
5  mm.setDefaultCloseOperation (WindowConstants.EXIT_ON_CLOSE);
6
7  mm.setVisible(true);
8
9  }
```

MeuMenu.java

Precisamos agora configurar o botão de Clientes para que quando clicado, ele abra a janela de MeusClientes.java. Para que o Java crie no Código-Fonte a função que irá executar quando o botão for clicado, basta clicar duas vezes no botão do modo edição.

Nesse caso precisaremos apenas abrir a janela, então o código será semelhante ao código que abriu a janela MeuMenu.java.

```
1  private void btnClientesActionPerformed (java.awt.event.ActionEvent evt) {
2  // TODO add your handling code here:
3
4  MeusClientes mc = new MeusClientes();
5  mc.setDefaultCloseOperation (WindowConstants.DISPOSE_ON_CLOSE);
6  mc.setVisible(true);
7
8  }
```

MeusClientes.java

Na classe `MeusClientes.java` a brincadeira fica mais interessante, pois logo de início iremos carregar as informações de contatos contidos no banco de dados. Para isso, iremos modificar o construtor, lembrando que o construtor é a função que é chamada quando um objeto é criado.

```
1 public MeusClientes() {  
2     initComponents();  
3  
4     atualizaClientes();  
5 }
```

O construtor contém duas funções. Uma que o Netbeans cria por padrão, que é a `initComponents()`, que faz com que os componentes gráficos sejam criados. A outra é o `atualizaClientes`, que iremos criar agora.

Antes de terminar a classe `MeusClientes`, iremos colocar o seguinte código:

```
1 public void atualizaClientes() {  
2  
3     ArrayList<Cliente> clientes = new  
4     ArrayList<Cliente>();  
5     Connection conexao = MinhaConexao.getInstance().sqlConnection;  
6     //fill client list  
7     String query = "SELECT nome, cpf FROM cliente";  
8     Statement stmt = null;  
9     try{  
10        stmt = conexao.createStatement();  
11        ResultSet rs = stmt.executeQuery(query);  
12        while(rs.next()) {  
13            String nome = rs.getString("nome");  
14            String cpf = rs.getString("cpf");  
15  
16            //adiciona cliente na lista  
17            clientes.add(new Cliente(nome, cpf));  
18        }  
19    } catch (Exception e) {  
20        e.printStackTrace();  
21    }
```

```

21     DefaultListModel listModel = new DefaultListModel();
22
23     for(Cliente c : clientes){
24         listModel.addElement(new String(c.getNome() + " - " + c.getCPF()));
25         System.out.println("Client name: " + c.getNome());
26     }
27
28     listClientes.setModel(listModel);
29
30

```

Vamos analisar o código linha por linha (considerando as mais importantes).

Linha 3– cria um ArrayList do tipo clientes. Um ArrayList, para quem não conhece, é uma estrutura parecida com um vetor, mas que se atualiza sozinha e fornece vários recursos que facilitam muito a vida de quem programa. Pode-se pensar que é um vetor do tipo Cliente, onde as informações são adicionadas utilizando o método add(Elemento do tipo Cliente), removido utilizando o método remove(índice de onde o elemento está guardado ou o objeto que deve ser removido), e o método get(índice de onde o método está guardado).

Linha 4 – Cria uma conexão utilizando a classe MinhaConexao.

Linha 6 – Nessa linha é criada a String com a consulta na sintaxe do SQL. Basicamente é selecionado o nome e cpf da tabela cliente.

Linha 9 – É criado um Statement, que é uma classe que auxilia a execução das queries.

Linha 10 – A query é executada e o resultado é guardado dentro da instância do objeto ResultSet.

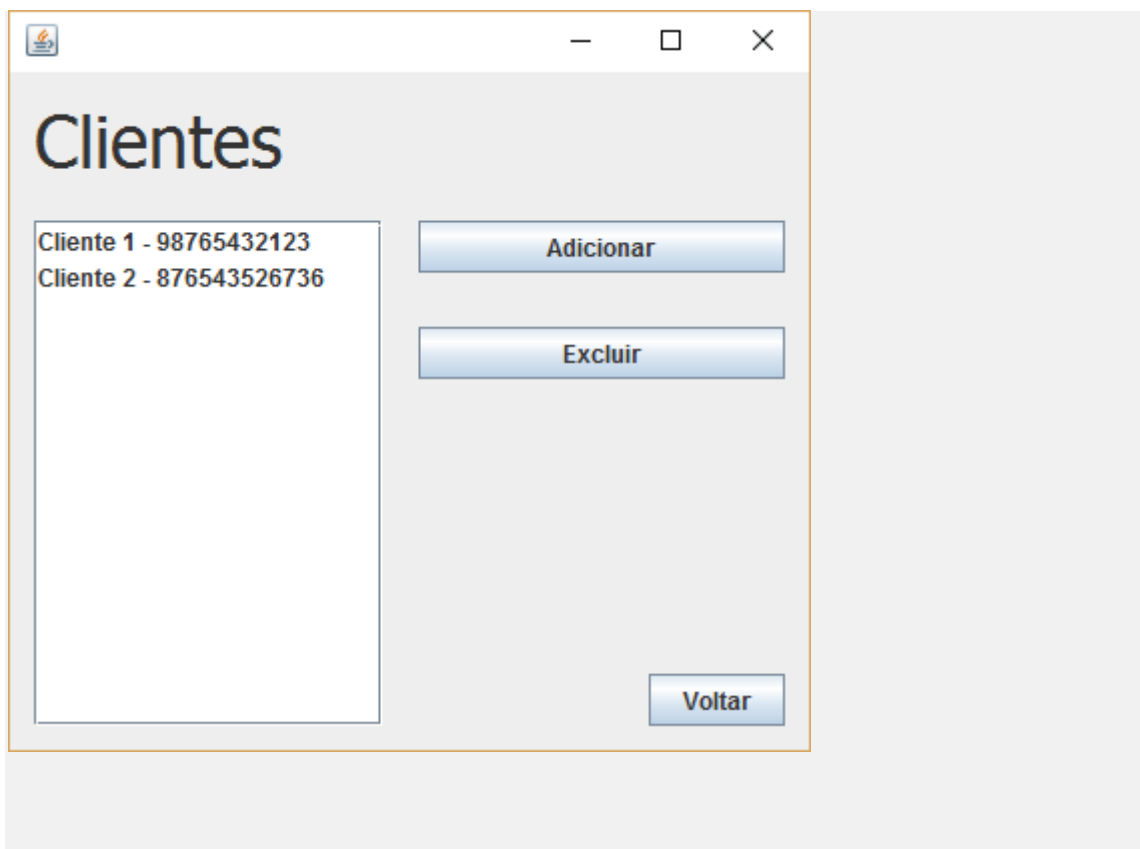
Linha 11 a 16– Faz uma iteração sobre cada elemento do resultado obtido pela execução da query. A variável rs aponta para uma linha do resultado. Ao utilizar o método next(), esse ponteiro aponta para a próxima linha do resultado. Dessa maneira é possível iterar sobre todos os elementos dos resultados. E a cada linha que é apontada, é retirado os valores da coluna nome pelo comando “rs.getString(“nome”)” e os valores da coluna cpf através do comando “rs.getString(“cpf”)”. O método getString é utilizado pois a coluna guarda uma variável do tipo String (varchar no MySQL). Caso a coluna guardasse um valor de inteiro, o método utilizado seria “rs.getInt(“nome da coluna do inteiro”)”. Por fim, uma nova instância da classe Cliente é criada (new Cliente(nome, cpf)) e é adicionado ao ArrayList de clientes. No final da repetição while, clientes conterá todos os clientes contidos na base de dados.

Linha 22 – É criado um ListModel, que é uma classe que vai guardar as informações que desejamos mostrar no elemento gráfico de lista.

Linha 24 a 26 – Usa um foreach do Java. Colocando em português, a linha 24 estaria dizendo: “para cada elemento c do tipo Cliente dentro da lista clientes, faça”. Ou seja, esse for vai pegar cada elemento contido na lista de clientes, e a cada iteração Cliente c vai receber o valor desse elemento da lista. A linha 25 adiciona um novo elemento no objeto ListModel. O elemento a ser adicionado precisa ser do tipo String, por conta disso é criado uma nova instância do objeto String onde serão concatenados o nome e o CPF do cliente separados por um traço. A linha 26 imprime no console o nome do cliente, possibilitando ver de outra maneira as informações que estão sendo utilizadas.

Linha 29 – Finalmente, após o ListModel ser preenchido com todos os nomes e cpfs dos clientes contidos no banco de dados, ele é atribuído ao elemento gráfico ListClientes.

Após esse passo, o programa pode ser executado. A janela de clientes deverá mostrar os dois clientes Cliente 1 e Cliente 2, que foram adicionados através do código SQL.



Programando o botão Adicionar

Clicando duas vezes no botão editar, podemos começar a colocar o código para chamar a tela para adicionar um novo contato. O código é o abaixo:

```
1    MeuAdicionarCliente mac = new MeuAdicionarCliente(this);  
2    mac.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);  
3    mac.setVisible(true);
```

Criando um novo objeto MeuAdicionarCliente podemos chamar a tela. Observando bem, esse trecho é um pouco diferente dos outros que utilizamos até agora para chamar novas janelas.

Ele contém o argumento `this` ao criar um novo `MeuAdicionarCliente`. Esse `this` irá passar uma referência do `JFrame` `MeusClientes` para a próxima janela que irá abrir. Dessa maneira poderemos controlar esse frame através do próximo.

`MeuAdicionarCliente.java`

A primeira coisa que devemos reparar nessa classe que é diferente das outras é o trecho logo abaixo da declaração da classe:

```
1  public class MeuAdicionarCliente extends javax.swing.JFrame {  
2  
3      Connection conexao = MinhaConexao.getInstance().sqlConnection;  
4      JFrame father;
```

Temos duas variáveis que estão fora de qualquer método, isso significa que elas podem ser utilizadas dentro de qualquer função que estiver declarada dentro dessa classe. Se alterarmos uma delas, dentro de uma função, ela permanecerá alterada dentro de outra função que a chame posteriormente.

O próximo ponto a se atentar são os construtores que serão diferentes das outras classes:

```
1  public MeuAdicionarCliente(JFrame father) {  
2      initComponents();  
3      this.father = father;  
4  }  
5  
6  public MeuAdicionarCliente() {  
7      initComponents();  
8  }
```

O primeiro construtor contém um argumento `JFrame father`. Esse argumento é para que a classe que chamar essa classe possa passar a referência da interface para essa classe. Dessa maneira é possível controlar o `JFrame` passado por parâmetro nesse próprio `JFrame`.

O outro construtor é utilizado para que a classe principal da classe possa inicializar os elementos gráficos. Nesse caso, ele não pode ter nenhum argumento.

Programando o botão de adicionar

O botão de adicionar deve fazer algumas tarefas:

Pegar o texto que estiver escrito nos `txtNome` e `txtCPF`;

Validar os textos;

Criar uma query de inserção;

Executar a query;

Atualizar a lista de clientes do JFrame anterior.

O seguinte código realiza essas tarefas, exceto a validação do texto.

```
1      private void btnAdicionarActionPerformed(java.awt.event.ActionEvent evt) {  
2  
3          String nome = txtNome.getText();  
4          String cpf = txtCPF.getText();  
5  
6          String query = "INSERT INTO cliente (nome, cpf) VALUES ('" + nome + "', '"  
7              + cpf + "')";  
8  
9          try{  
10             Statement stmt = conexao.createStatement();  
11             stmt.executeUpdate(query);  
12             conexao.close();  
13         }catch(Exception e){  
14             e.printStackTrace();  
15         }  
16  
17         father.invalidate();  
18         father.validate();  
19         father.repaint();  
20         this.dispose();  
21     }
```

As linhas 3 e 4 realizam a tarefa 1. A linha 6 realiza a tarefa 3. As linhas 9 a 11 realizam a tarefa 4, e as linhas 16 a 18 realizam a tarefa 5.

Nas linhas 16 a 18 é possível reparar que a variável `father` é utilizada. É nesse momento que estamos atualizando as informações do JFrame que mostra os clientes para que ele possa ser atualizado e mostrar o cliente que acabou de ser adicionado.


O resultado pode ser visto nas próximas imagens.

— □ ×

Adicionar Cliente

Nome:

CPF:

— □ ×

Clientes

Cliente 1 - 98765432123

Cliente 2 - 876543526736

PinhataDev - 12345678910