

# EducaCiência FastCode

Fala Galera,

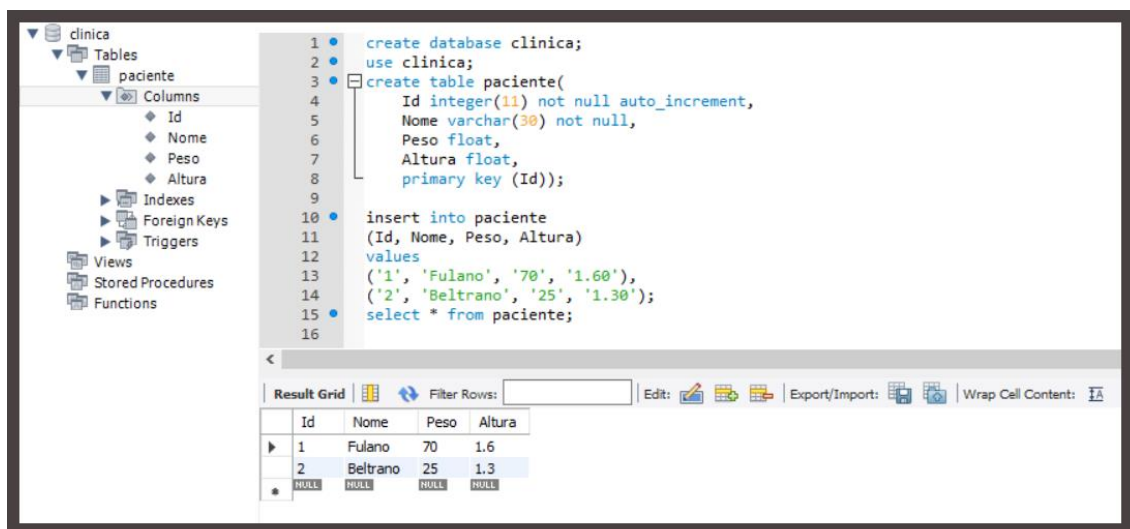
- Artigo: 24/2020 Data: Junho/2020
- Público Alvo: Desenvolvedores – Iniciantes
- Tecnologia: Java
- Tema: CRUD – criando Método Listar Por Nome
- Link: <https://github.com/perucello/DevFP>

Criaremos uma série de artigos para explanarmos o CRUD em um projeto Desktop. Neste artigo 24/2020 daremos continuidade no nosso propósito, criando o MÉTODO LISTAR POR NOME, lembrando que o artigo 20/2020 trouxemos o método Inserir e no artigo 23/2020 trouxemos o Método Listar Todos.

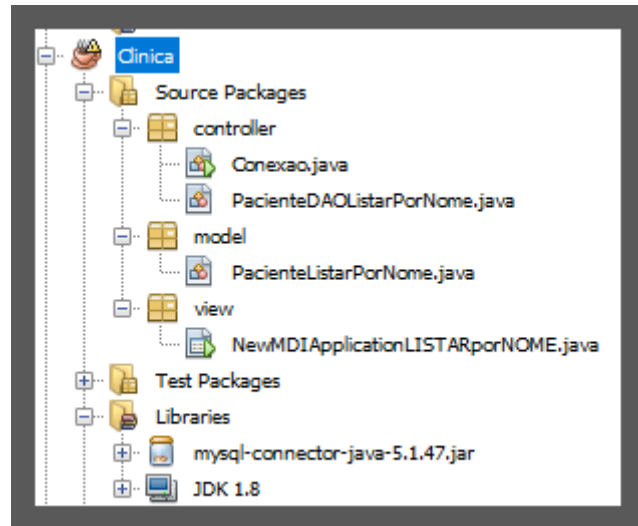
Para este ambiente , já temos nosso Banco de Dados criado e chamado “clinica”.

Nosso ambiente consiste em:

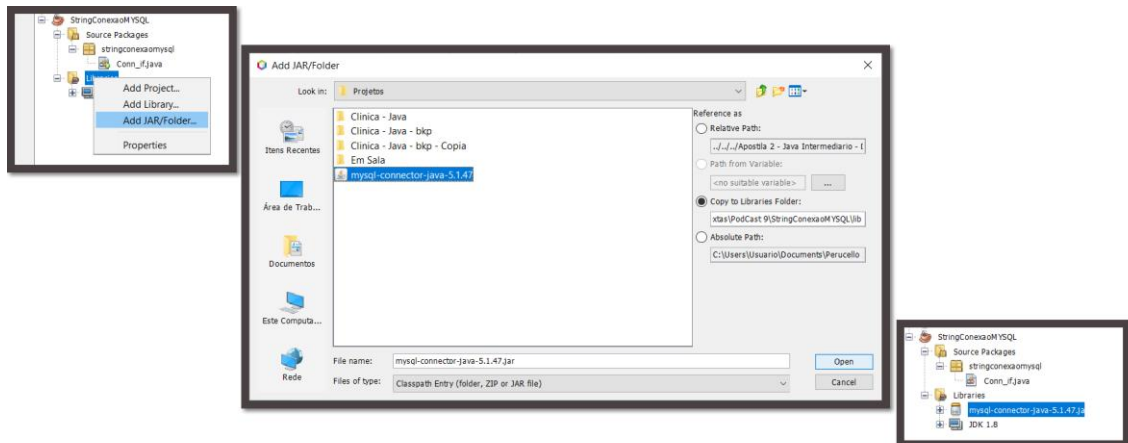
⇒ Banco de Dados MySql



⇒ Padrão MVC de arquitetura



Vale lembrar que temos que adicionar nossa biblioteca “jar” de Conexão. Vamos lá.



# CRUD

## Listar Por Nome

Vou tentar detalhar o máximo possível os passos para que não fique nenhuma dúvida , ou que possa ser replicado sem maiores problemas o nosso Projeto.

Porém, neste artigo vamos abordar o Método Listar por Nome baseado , lembrando que no Artigo 23/2020 trouxemos o Método Listar e neste artigo em questão daremos continuidade no Projeto agregando este novo Método.

**Pacote Controller** – neste pacote, teremos duas classes, a nossa Classe Conexão e nossa Classe PacienteDAOInserir.

a) Classe Conexão – nesta classe, teremos nossa String de Conexão com o Banco de Dados

```
1 package controller;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.SQLException;
5 import javax.swing.JOptionPane;
6 //Classe Conexão
7 public class Conexão {
8     //Detalhes da Conexão
9     private static final String DATABASE="clinica"; // Nome do Banco de Dados
10    private static final String HOST="jdbc:mysql://localhost:3306/clinica"; //Host
11    private static final String DRIVER="com.mysql.jdbc.Driver"; //Driver
12    private static final String URL="jdbc:mysql://localhost:3306/clinica?useTimezone=true&serverTimezone=UTC&useSSL=false"; //Para tirar erro de SSL em alguns casos em que tem varios Bancos e Certificados
13    private static final String USER="root"; //Nome do usuario de acesso ao Banco de Dados
14    private static final String PWD=""; // senha definida no Banco de Dados
15
16    //Método conectar
17    public static Connection Conectar(){
18        try{
19            Class.forName(DRIVER);
20            return DriverManager.getConnection(URL, USER, PWD);
21        }
22        catch (ClassNotFoundException | SQLException e) {
23            System.out.println("ERRO: " + e.getMessage());
24            JOptionPane.showMessageDialog(null, "Falha de comunicação com BD", "ERRO", JOptionPane.WARNING_MESSAGE);
25            return null;
26        }
27    }
28
29    //Método Desconectar
30    public static void Desconectar (Connection con){
31        try{
32            if (con != null){
33                con.close();
34            }
35        }
36        catch (SQLException e){
37            System.out.println("ERRO: " + e.getMessage());
38        }
39    }
40
41    //PARA TESTAR VIA SCRIPT
42    //Método Main
43    public static void main(String[] args){
44        if (Conectar() != null){
45            System.out.println("Conexão realizada com sucesso!");
46        }
47    }
48 }
```

- b) Classe PacienteDAOListarPorNome – nesta classe, adicionaremos as Strings Sql que será executada junto ao nosso Banco de Dados, vale ressaltar que temos alguns passos pré definidos no nosso código, onde estou utilizando do recurso *//comentário* para explicar melhor.

## Metodo Listar

```

11 public class PacienteDAOListarPorNome {
12
13     public static List<PacienteListarPorNome> list() {
14         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
15     }
16
17     private final Connection con;
18     private PreparedStatement cmd;
19
20     public PacienteDAOListarPorNome() {
21         this.con = Conexao.Conectar();
22     }
23
24     public List<PacienteListarPorNome> listar() {
25         try {
26             String sql = "select * from paciente order by id";
27             cmd = con.prepareStatement(sql);
28             ResultSet rs = cmd.executeQuery();
29             List<PacienteListarPorNome> lista = new ArrayList<>();
30             while(rs.next()) {
31                 PacienteListarPorNome p = new PacienteListarPorNome();
32                 p.setId(rs.getInt("id"));
33                 p.setNome(rs.getString("nome"));
34                 p.setPeso(rs.getFloat("peso"));
35                 p.setAltura(rs.getFloat("altura"));
36                 lista.add(p);
37             }
38             return lista;
39         } catch (SQLException e) {
40             System.out.println("ERRO: " + e.getMessage());
41             return null;
42         } finally {
43             Conexao.Desconectar(con);
44         }
45     }
46
47
48
49     public List<PacienteListarPorNome> pesquisarPorNome(String nome) [...28 lines ]
50 }

```

## Metodo Listar Por Nome

```

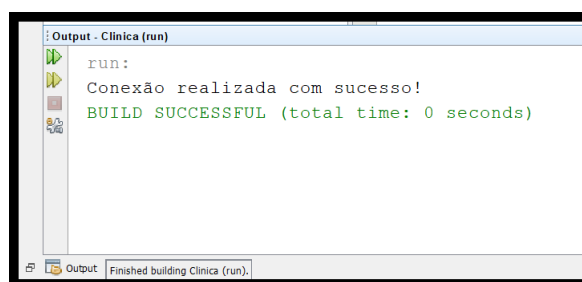
13 public static List<PacienteListarPorNome> list() {
14     throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
15 }
16
17 private final Connection con;
18 private PreparedStatement cmd;
19
20 public PacienteDAOListarPorNome() {
21     this.con = Conexao.Conectar();
22 }
23
24 public List<PacienteListarPorNome> listar() { ...25 lines }
25
26
27 public List<PacienteListarPorNome> pesquisarPorNome(String nome) {
28     try {
29         String sql = "select * from paciente where nome like ? order by nome";
30         cmd = con.prepareStatement(sql);
31         cmd.setString(1, "%" + nome + "%");
32
33         ResultSet rs = cmd.executeQuery();
34         List<PacienteListarPorNome> lista = new ArrayList<>();
35
36         while(rs.next()) {
37             PacienteListarPorNome p = new PacienteListarPorNome();
38             p.setId(rs.getInt("id"));
39             p.setNome(rs.getString("nome"));
40             p.setPeso(rs.getFloat("peso"));
41             p.setAltura(rs.getFloat("altura"));
42
43             lista.add(p);
44         }
45         return (List<PacienteListarPorNome>) lista;
46     } catch (Exception e) {
47         System.out.println("ERRO: " + e.getMessage());
48         return null;
49     } finally {
50         Conexao.Desconectar(con);
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

```

Com isso, temos nosso Pacote Controller já pronto, temos nossa String de Conexão com o Banco de Dados OK e temos nossa Classe que será responsável por deter a manipulação com o Banco de Dados Ok.

Vamos apenas testar nossa conexão com o Banco de Dados.

- Subir WampServer
- Clicar com Botão na classe Conexao e pedir para executar (run)



**Pacote Model** – nesta classe, teremos nossa Regra de Negócio, porém, neste Projeto em questão, vamos ter nossos Métodos Getters e Setters – lembrando que no Netbeans, temos um recurso para gerar estes Métodos automáticos, onde estamos informando o momento em nosso Script.

```

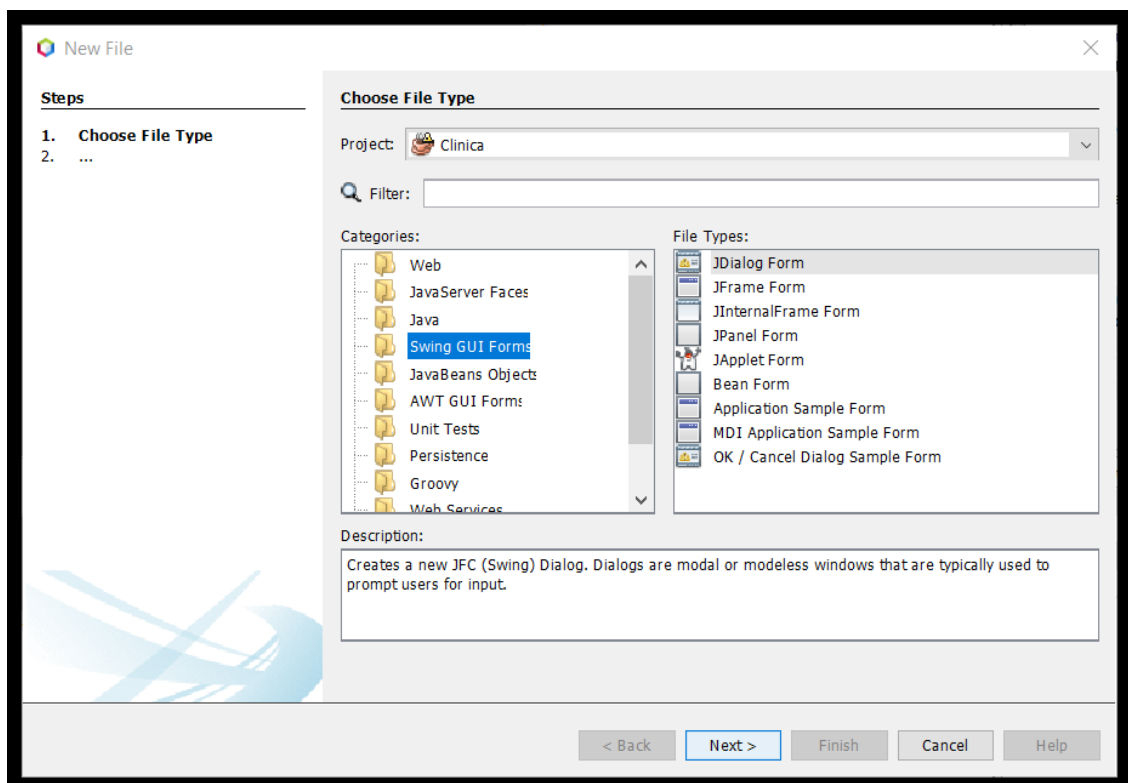
1 package model;
2 public class PacienteListarPorNome {
3     //Criando nossos atributos - variaveis
4     private int id;
5     private String nome;
6     private float peso;
7     private float altura;
8
9     public PacienteListarPorNome() {
10    }
11    public PacienteListarPorNome(int id, String nome, float peso, float altura) {
12        this.id = id; //referenciando ao id do Banco de Dados
13        this.nome = nome; //referenciando ao nome no Banco de Dados
14        this.peso = peso; //referenciando ao peso no Banco de Dados
15        this.altura = altura; //referenciando à altura no Banco de Dados
16    }
17    //Métodos Getters e Setters - para este momento, podemos clicar com botão direito, irmos em INSERTCODE e pedimos para gerar automaticamente os Métodos.
18    public int getId() {
19        return id;
20    }
21    public void setId(int id) {
22        this.id = id;
23    }
24    public String getNome() {
25        return nome;
26    }
27    public void setNome(String nome) {
28        this.nome = nome;
29    }
30    public float getPeso() {
31        return peso;
32    }
33    public void setPeso(float peso) {
34        this.peso = peso;
35    }
36    public float getAltura() {
37        return altura;
38    }
39    public void setAltura(float altura) {
40        this.altura = altura;
41    }
42    @Override
43    public String toString() {
44        return nome;
45    }
46 }

```

**Pacote View** – Nesse momento, após criarmos as classes nos Pacotes Controller e Model vamos criar a interação do Projeto com o Usuario, ou seja, a Tela em que o usuário irá ver e manipular os dados para que seja inserido no Banco de Dados as informações como é nossa proposta neste artigo – CRUD Método Listar Por Nome.

Não é minha intenção deixar a Tela com o visual bem apresentado, mas, sim explorarmos os códigos do lado do BackEnd, sendo assim, vamos utilizar de um artefato visual que o próprio netbeans nos fornece.

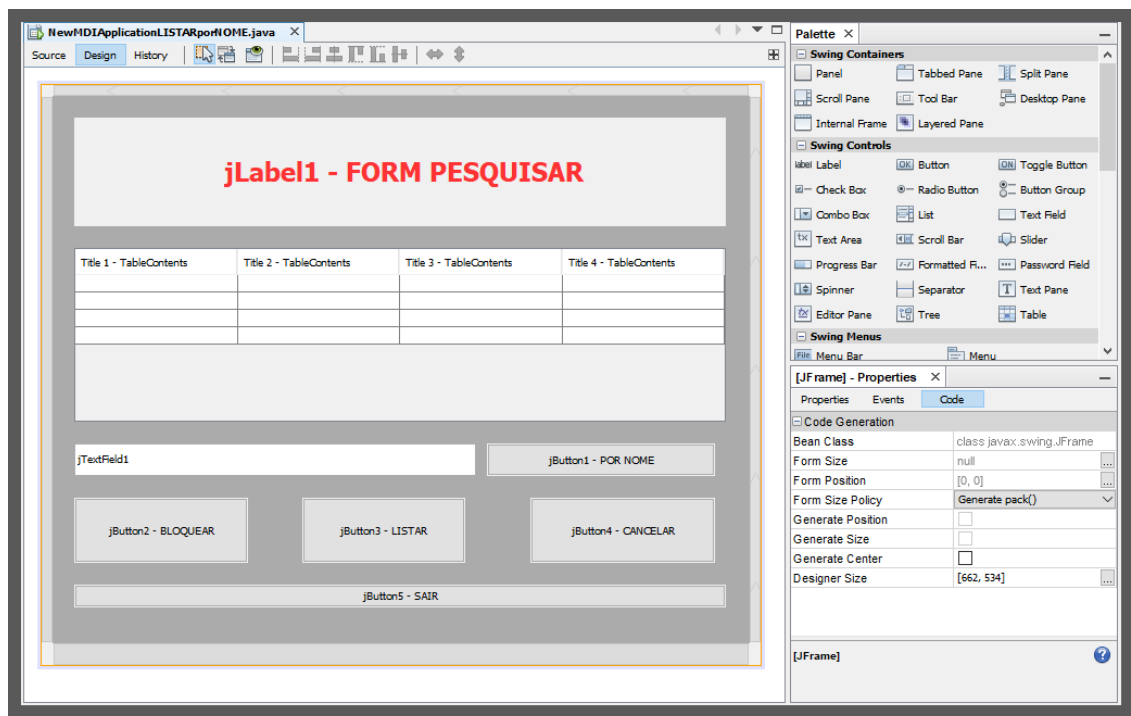
Vamos clicar com Botão direito no pacote View irmos em **New/Other** e selecionar **Swing Gui Forms**, nesta categoria vamos selecionar **MDI Application Sampe Form**.



Vamos nomear nosso Form criado MDI e vamos manipulá-lo agora.

Visto que temos nossa tabela no Banco de Dados já criada, vamos definir a tela do nosso Form MDI e criarmos a interface.

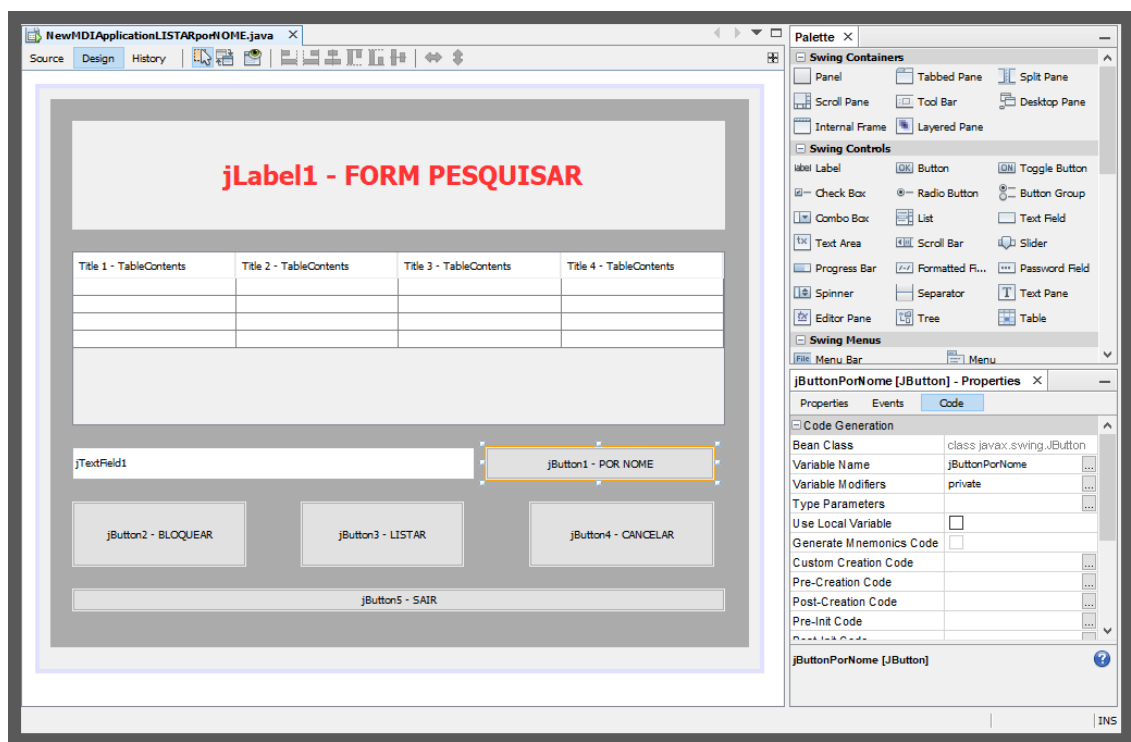
Para maior familiaridade, vou pôr a tela criada com os recursos do Palette utilizado para fins didáticos.



Com estes passos criados, precisamos nomear nossas variáveis, neste momento, basta você clicar no Palette inserido, ir em Propriedades, CODE e nomear nossa variável.

Para este artigo, estou nomeando as variáveis para serem didáticas na exemplificação.

Faremos este processo para todos os campos.





O netbeans, se formos na seção de códigos (source) , veremos que o Netbeans nos fornece todas estas informações que manipulamos.

```
// Variables declaration - do not modify
private javax.swing.JDesktopPane desktopPane;
private javax.swing.JButton jButtonBlock;
private javax.swing.JButton jButtonCancelar;
private javax.swing.JButton jButtonListar;
private javax.swing.JButton jButtonPorNome;
private javax.swing.JButton jButtonSair;
private javax.swing.JLabel jLabel1;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JTable jTableResultado;
private javax.swing.JTextField jTextFieldPESQUISAR;
// End of variables declaration
```

Agora com nosso Layout da tela pronto, nossas variáveis já nomeadas , vamos aos códigos. Na Opção Source, iremos manipular os códigos, estarei aqui colocando um Passo a Passo para que acompanhem a linha de raciocínio.

Vamos até o final do nosso código, após as declarações das variáveis, vamos identificar a última Chave” e antes dela, vamos incluir os seguintes passos:

```
230 //Passo 1 - nesse momento, iremos configurar nosso formulario
231 private void configurarFormulario() {
232     this.setTitle("Java Intermediário - Escola Evolua Sumaré"); //definiremos o Titulo do nosso Form
233     this.setResizable(false); //Desabilitaremos o botao Maximizar
234     this.setLocationRelativeTo(null); //configuraremos nosso form para iniciar ja no centro da tela
235     estadoControle(true); // add após criar estado controle ( passo 2)
236     preencherTabela(new PacienteDAOListarPorNome().listar()); //add apos criar preencherTabela (passo 4)
237 }
238
239 //passo 2 - neste momento iremos criar um método Estado Controle para nossos Botoes.
240 private void estadoControle(boolean e) {
241     //Botões
242     jButtonBlock.setEnabled(e);
243     jButtonCancelar.setEnabled(!e);
244     jButtonListar.setEnabled(e);
245     //txt
246     jTextFieldPESQUISAR.setEnabled(e);
247     //metodo para limpar o conteudo do texto
248     LimparTexto();
249 }
250
251 //Passo 3 - configurar nosso formulario - tabela
252 private void configurarTabela() {
253     DefaultTableModel m = new DefaultTableModel() {
254         @Override
255         public boolean isCellEditable(int row, int column) {
256             return false;
257         }
258     };
259     m.addColumn("Identificação");
260     m.addColumn("Nome do Paciente");
261     m.addColumn("Peso do Paciente");
262     m.addColumn("Altura do Paciente");
263
264     jTableResultado.setModel(m);
265 }
266
267 //passo 4 - preencher nosso formulario - tabela
268 private void preencherTabela(List<PacienteListarPorNome> lista) {
269     if(lista.size() > 0) {
270         configurarTabela();
271         DefaultTableModel m = (DefaultTableModel) jTableResultado.getModel();
272         for(PacienteListarPorNome p : lista) {
273             m.addRow(new Object[] {
274                 p.getId(), p.getNome(), p.getPeso(), p.getAltura()
275             });
276         }
277         jTableResultado.setModel(m);
278     }
279 }
280
281 // Passo 5 - criamos um metodo apenas para limpar o conteudo do campo texto
282 private void LimparTexto() {
283     jTextFieldPESQUISAR.setText("");
284 }
285
286 }
```

Agora que criamos nossos métodos, vamos aproveitar, e instanciar o método **configurarformulario** para iniciar junto à aplicação.

```

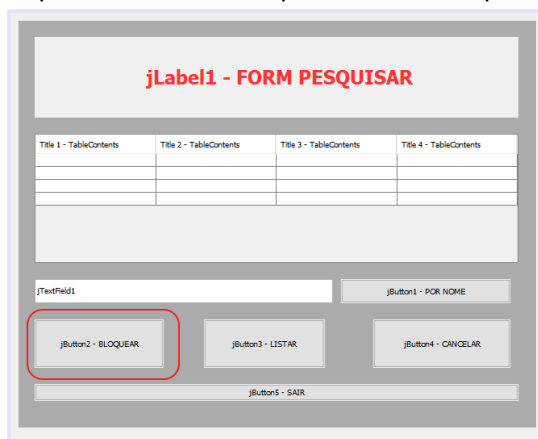
9 public class NewMDIApplicationLISTARporNOME extends javax.swing.JFrame {
10
11     public NewMDIApplicationLISTARporNOME() {
12         initComponents();
13         configurarFormulario();
14     }
15

```

Feito isso, vamos manipular ou melhor, colocar nossos códigos em nossos botões.

#### Volte no Design do Projeto e dê um duplo click BOTAO jButton2 - BLOQUEAR

Aqui definimos que o nosso **estadocontrole** ao ter o Botão jButton2 – BLOQUEAR pressionado, habilitará os campos que até então temos ele bloqueado e chamamos o Método Limpar para limpar todos os textos que estiverem disponíveis.



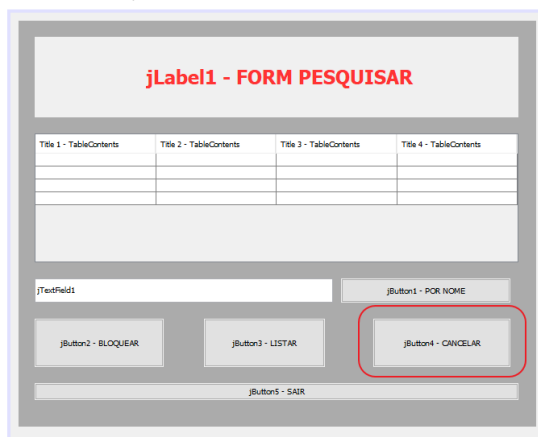
```

173 private void jButtonBlockActionPerformed(java.awt.event.ActionEvent evt) {
174     estadoControle(false);
175     configurarTabela();

```

#### Volte no Design do Projeto e dê um duplo click BOTAO jButton4 - CANCELAR

Neste botão, vamos fazer o inverso do estadocontrole do botão novo, pois ao clicar em CANCELAR, queremos que os campos textos e os botões SALVAR e CANCELAR sejam bloqueados, no entanto, basta neste momento definirmos o **estadocontrole** como (true).



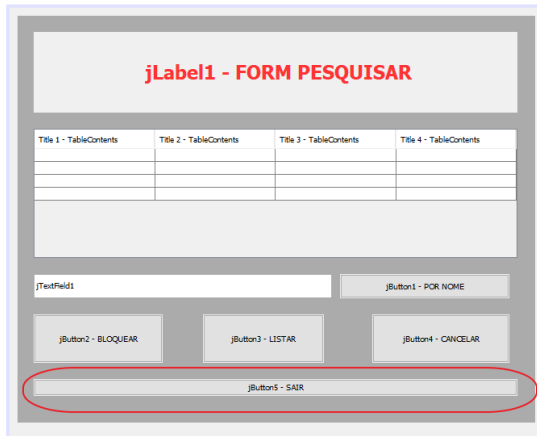
```

165 private void jButtonCancelarActionPerformed(java.awt.event.ActionEvent evt) {
166     estadoControle(true);
167 }

```

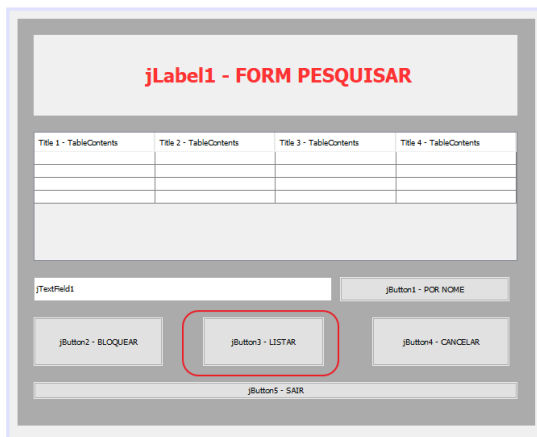
### Volte no Design do Projeto e dê um duplo click BOTAO jButton5 - SAIR

Aqui, vamos dar um “dispose” que além de fechar nossa tela, tem como proposito finalizar o processo Java.



```
179 | private void jButtonSairActionPerformed(java.awt.event.ActionEvent evt) {
180 |     dispose();
181 | }
```

### Volte no Design do Projeto e dê um duplo click BOTAO jButton1 - POR NOME



```
169 | private void jButtonListarActionPerformed(java.awt.event.ActionEvent evt) {
170 |     preencherTabela(new PacienteDAOListarPorNome().listar());
171 | }
```

Agora está faltando apenas nosso botão LISTAR POR NOME, então vamos manipular ele, pois é um código um pouco maior.

## Volte no Design do Projeto e dê um duplo click BOTAO jButton3 - LISTAR



```

151 private void jButtonPorNomeActionPerformed(java.awt.event.ActionEvent evt) {
152
153     String nome = jTextFieldPESQUISAR.getText();
154     if(!nome.isEmpty()){
155         JOptionPane.showMessageDialog(null, "Pesquisando nomes com a letra (" + jTextFieldPESQUISAR.getText() + ")",
156             "Pesquisa por nome", JOptionPane.INFORMATION_MESSAGE);
157         preencherTabela(new PacienteDAOListarPorNome().pesquisarPorNome(nome));
158     }
159     else{
160         JOptionPane.showMessageDialog(null, "Retornando lista completa", "Lista completa", JOptionPane.WARNING_MESSAGE);
161         preencherTabela(new PacienteDAOListarPorNome().listar());
162     }
163     estadoControle(false);
164 }
        
```

Agora, que finalizamos nosso projeto, chega o momento mais interessante, vamos rodar. Vamos executar nosso Form – clique com botão direito sobre o MDI do pacote VIEW e vamos selecionar “run”.

```

import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import model.PacienteListarPorNome;


public class NewMDIApplicationLISTARporNOME {

    public NewMDIApplicationLISTARporNOME() {
        initComponents();
        configurarFormulario();
    }

    @SuppressWarnings("unchecked")
    Generated Code


    private void jButtonPorNomeActionPerformed() {


        String nome = jTextFieldPESQUISAR.getText();
        if(!nome.isEmpty()){
            JOptionPane.showMessageDialog(null, "Pesquisando nomes com a letra (" + jTextFieldPESQUISAR.getText() + ")",
                "Pesquisa por nome", JOptionPane.INFORMATION_MESSAGE);
            preencherTabela(new PacienteDAOListarPorNome().pesquisarPorNome(nome));
        }
        else{
            JOptionPane.showMessageDialog(null, "Retornando lista completa", "Lista completa", JOptionPane.WARNING_MESSAGE);
            preencherTabela(new PacienteDAOListarPorNome().listar());
        }
        estadoControle(false);
    }
}
        
```



Notamos que os dados retornaram normalmente, agora apenas para manipular, vamos:

⇒ Clicar no Botão Bloquear





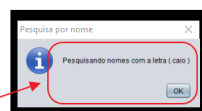
⇒ Clicar no Botão Cancelar



⇒ Clicar no Botão Listar



⇒ Clicar no Botão Listar Por Nome, vamos passar como “nome de pesquisa” o nome do Paciente “Caio” e clicar no Botão Por Nome



⇒ Clicar no Botão Sair



Nota-se que os códigos funcionaram corretamente, no entanto, saliento que este artigo 24/2020 é continuidade do Artigo 23/2020 onde assim como o artigo 20/2020 são baseados no curso de Java que ministro na Escola Evolua – Ensino Profissionalizante e como proposito de ajuda à comunidade, estamos trazendo parte da didática em forma de artigo comunitário e assim podemos contribuir com a comunidade Tecnológica como um todo.

Agradeço imensamente a Diretoria da Escola Evolua de Sumaré.

Os códigos estarão disponíveis no Git.

Até mais !

Espero ter ajudado !

