Java – Exemplo MDI

Nesta aula criaremos uma aplicação simples de cadastro para avaliar o nível de dificuldade/facilidade que é desenvolver *softwares* com a tecnologia Java.

No NetBeans, crie um novo projeto e dê o nome de “ExemploMDI”. Escolha um local adequado para salvar o seu projeto.

Na construção deste sistema simplificaremos ao máximo o nível de detalhamento das telas, pois, temos três fatores primordiais a serem levados em consideração:

- não é objetivo desta aula ensinar como se deve construir de sistemas de informação,
- o tempo é curto e
- quanto mais simples for o sistema, mais fácil será para avaliarmos o grau de dificuldade de utilização da tecnologia.

Sua tela do NetBeans deve estar semelhante à apresentada na Figura 1.

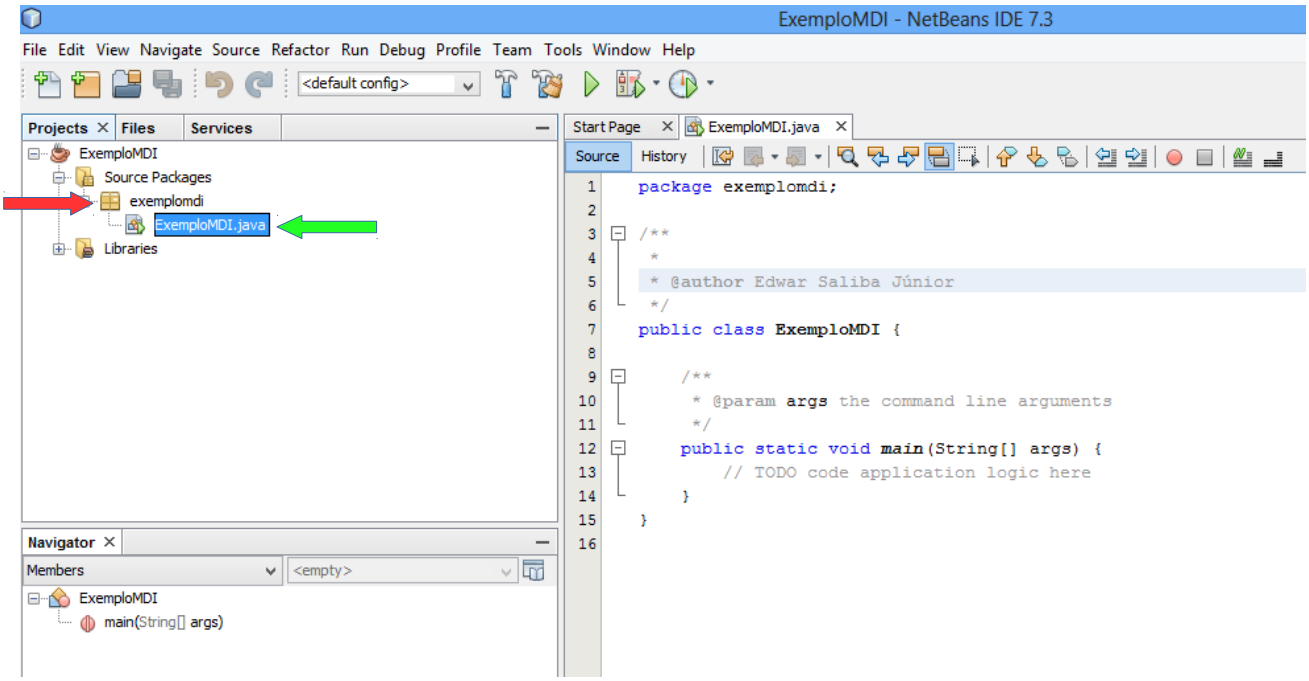

Figura 1: Criação do Projeto "ExemploMDI"

Tela “Principal”

A tela Principal será constituída apenas de um *menu* para acesso aos demais formulários que comporão o sistema.

Adicionando Componentes ao Formulário

Vamos criar a tela principal, faça o seguinte:

- clique com o botão direito do *mouse* encima do *package* (pacote) “exemplomdi” (seta vermelha na Figura 1), em seguida em “New | JFrame Form...” (Figura 2). Na janela que abrir, preencha o campo *Class Name* com: “Principal” (Figura 3) e pressione o botão *Finish*. Será criado um novo formulário no pacote “exemplomdi”;
- clique com o botão direito do *mouse* sobre a classe “ExemploMDI.java” (seta verde na Figura 1) e escolha a opção “Delete”, pois, não faremos uso desta classe em nosso projeto.

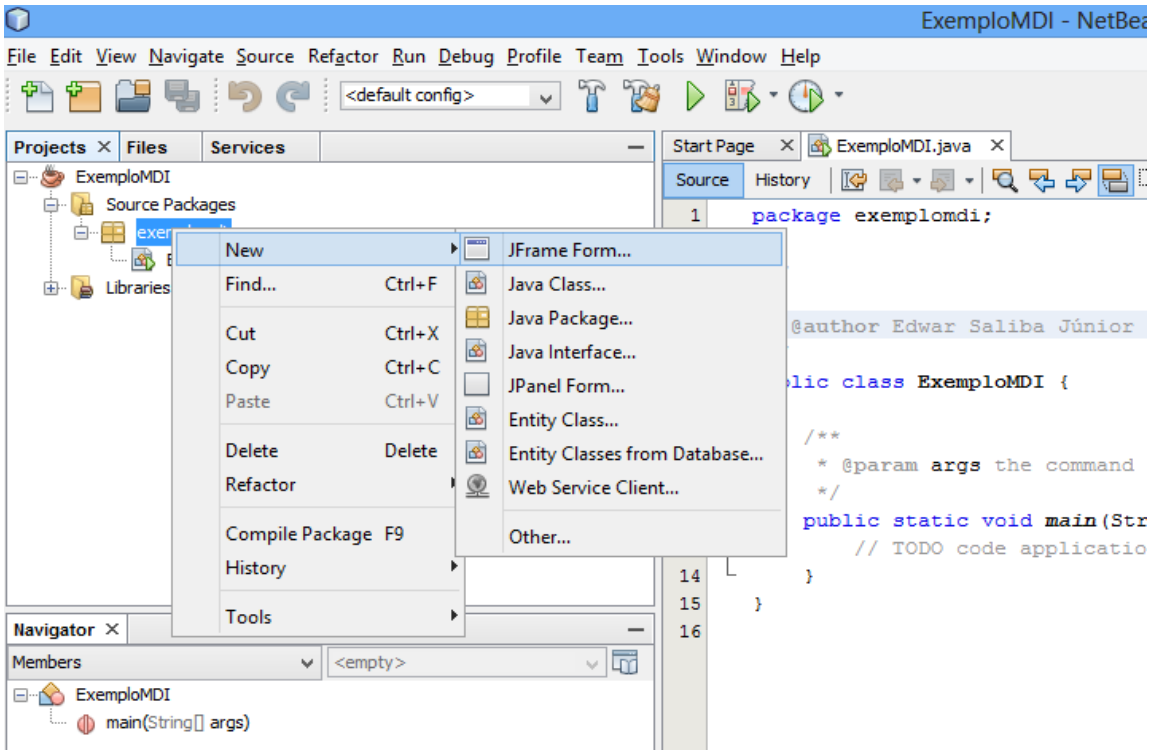

Figura 2: Inclusão de um novo formulário.

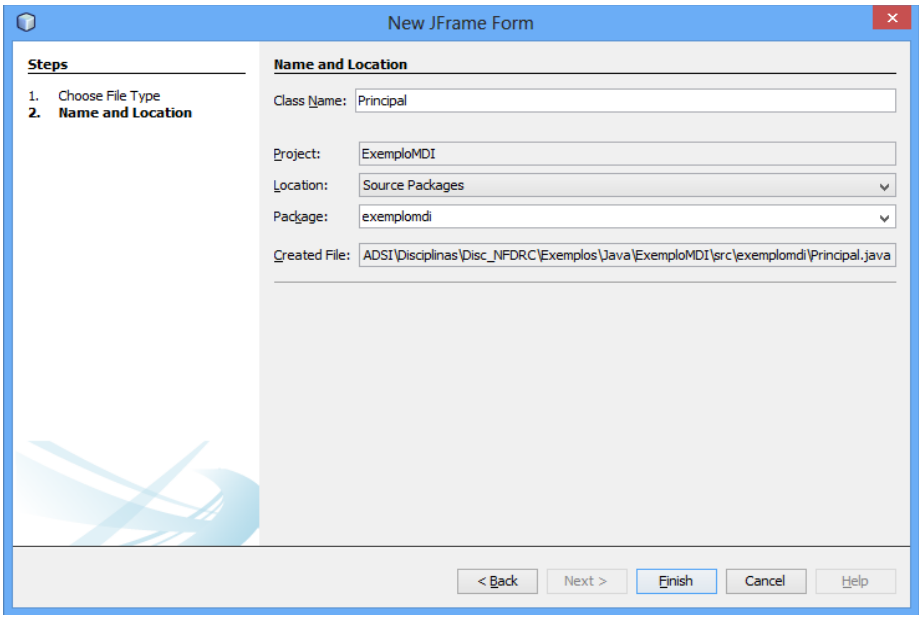

Figura 3: Criação do formulário Principal.

E por último, o seu ambiente de trabalho deve estar semelhante ao apresentado na Figura 4.

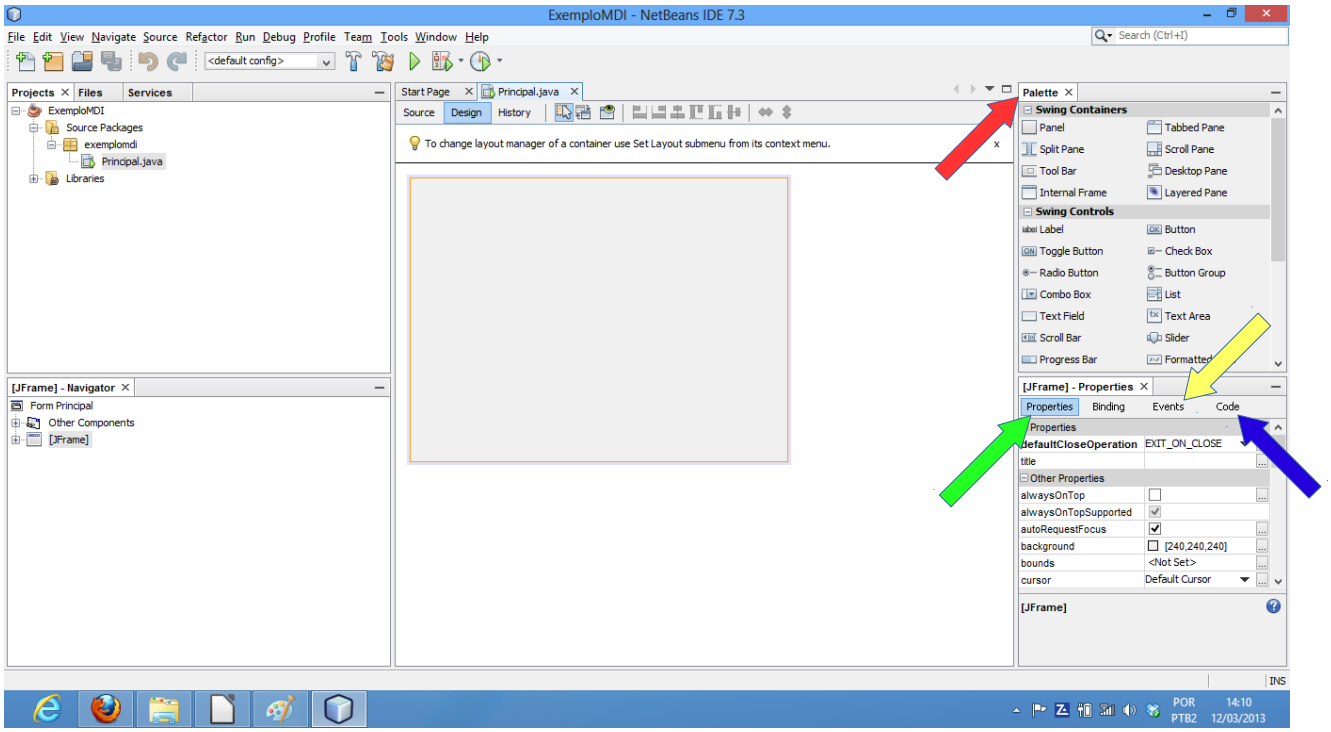

Figura 4: Ambiente de trabalho.

Dando continuidade ao nosso trabalho, no formulário que aparece na Figura 4, insira um componente chamado *MenuBar*, encontrado na aba *Palette* (seta vermelha na Figura 4) na divisão *Swing Menus*. Caso a aba *Palette* não esteja visível, acesse a opção de *menu "Window | Palette"* e a aba aparecerá. Lembrando que esta aba só se faz visível na presença de um formulário que contenha um objeto do tipo *contêiner*.

Após a colocação do componente sua tela deverá estar assim (Figura 5):

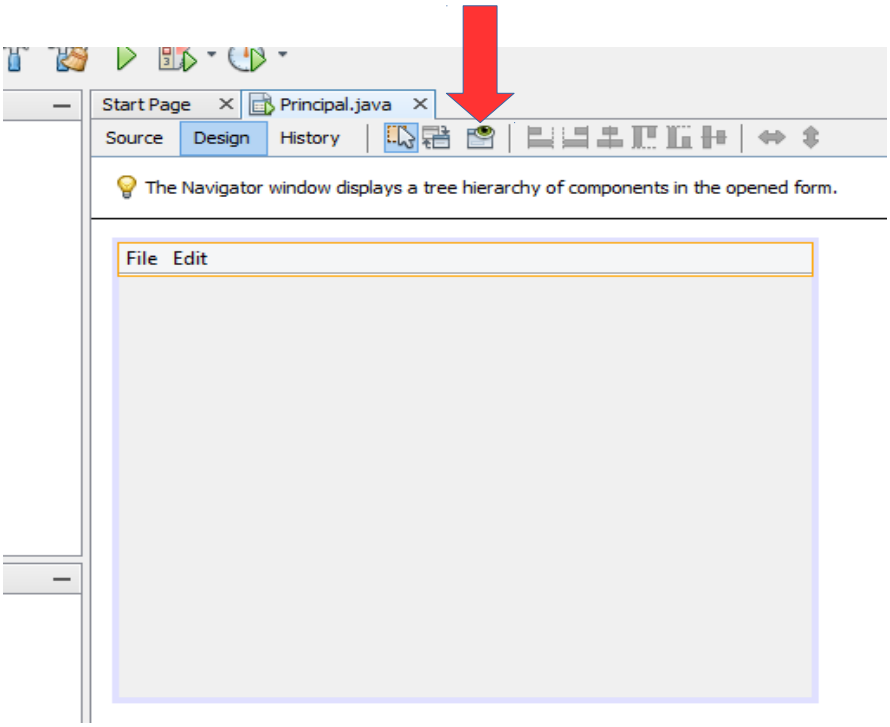

Figura 5: Formulário com componente "MenuBar".

Pois bem, agora dê um clique simples sobre a palavra *File* do componente *MenuBar*. Aparecerá uma borda laranjada em volta da palavra *File*. Isto significa que você selecionou o componente, agora, vá até a aba *Properties* (seta verde na Figura 4) e na propriedade *text* substitua a palavra *File* pela palavra *Arquivo*.

Em seguida clique na aba *Code* (seta azul na Figura 4), acesse a propriedade *Variable Name* e mude o texto que lá estiver para *mnuArquivo*. Isto permitirá que nós identifiquemos com mais facilidade, o respectivo evento associado a este item de *menu* quando chegar a hora. Repita estes passos para todos os itens de *menu* que você vai criar a seguir, porém cada qual terá sua propriedade *Variable Name* escrita da seguinte forma *mnuXXXX* (onde o XXXX deverá ser substituído pelo texto referente ao item de *menu* em questão.).

Agora, praticando os mesmos passos acima, substitua o item de *menu Edit* pela palavra *Cadastros*.

Feito isto, clique com o botão direito do *mouse* sobre a opção *Arquivo*. Aparecerá um *menu pop-up* onde você deverá escolher a opção *Add From Palette | Menu Item* (Figura 6). Aparecerá um novo item de *menu*, mude seu texto para *Sair*.

Observação: Se você quiser ver como está ficando seu formulário, clique no botão *Preview Design* (seta vermelha na Figura 5) e você terá uma prévia da maneira que seu formulário ficará após ser compilado e executado. Para fechar o formulário que aparecer basta clicar no botão X com fundo vermelho, no canto superior direto do próprio formulário.

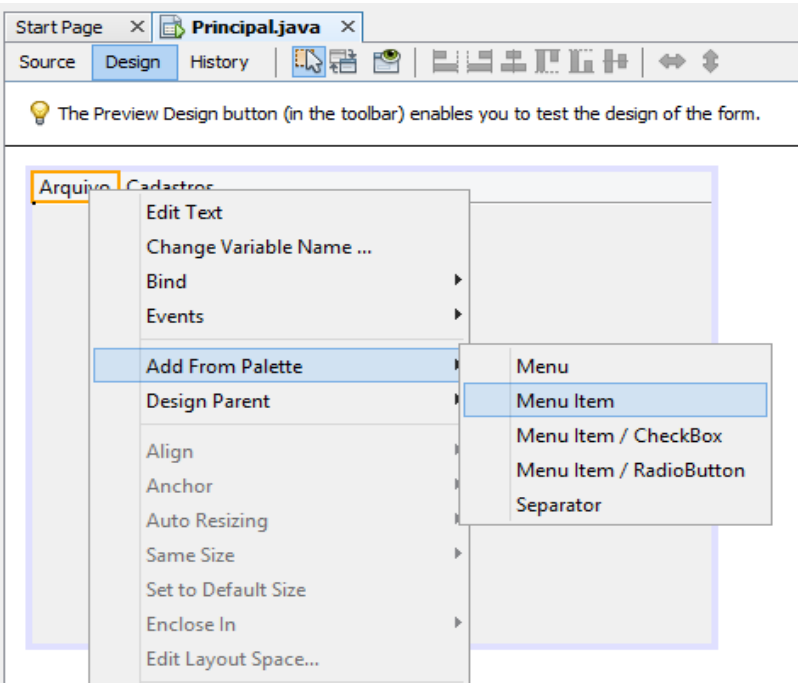

Figura 6: Adicionando itens de menu.

Agora adicione mais dois itens de *menu* a opção *Cadastros*. O primeiro deverá possuir o texto *Clientes* e o segundo o texto *Estados*. Não se esqueça de mudar as propriedades *Variable Name* (na aba *Code*) dos itens criados.

Finalizando, clique com o botão direito do *mouse* sobre o componente JMenuBar e clique na opção *Add Menu*. No novo *menu* que aparecer, mude o texto para *Ajuda*. Novamente usando o botão direito do *mouse* clique sobre o *menu Ajuda* que acabamos de criar e adicione um *submenu* com o texto *Sobre*. Não se esqueça de mudar a propriedade *Variable Name* para os itens de *menu* que acabamos de criar, ou seja, respectivamente: *mnuAjuda* e *mnuSobre*.

Criando os Outros Formulários do Software

Vamos criar os outros formulários que devem estar presentes no sistema. No nosso caso, nosso banco de dados possui apenas duas tabelas: Cliente e Estado. Assim sendo, nós

criaremos mais dois formulários. São eles: Clientes e Estados.

Sendo que os formulários de Clientes e Estados deverão ser dois para cada qual, pois, será um para visualização e outro para manipulação dos dados.

Então vamos lá:

- clique com o botão direito do *mouse* sobre o pacote do projeto (conforme mostrado na Figura 2), acesse a opção *New | Other...* aparecerá a janela mostrada na Figura 7. Então, no campo *Categories* clique em *Swing GUI Forms* e no campo *File Types* escolha *InternalFrame Form* e clique no botão *Next*;
- na próxima janela que abrir, no campo *Class Name*, digite o nome *ClientesVisao*;
- siga os mesmos passos para a criação do formulário *EstadosVisao*.

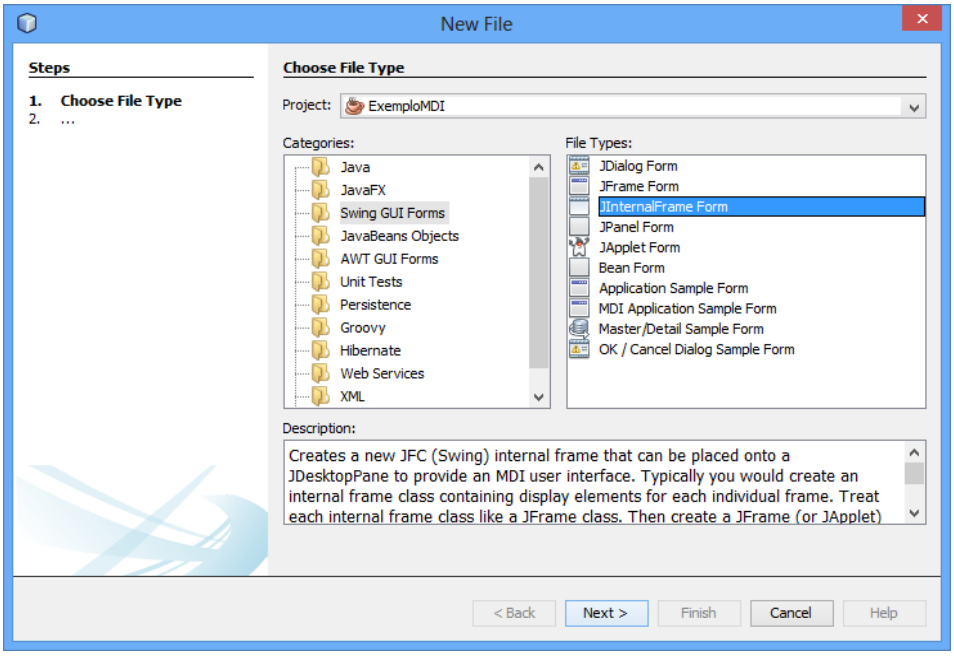

Figura 7: Janela para criação de novo tipo de formulário.

Acabamos de criar os formulários para visualização dos dados. Agora vamos criar os formulários para edição:

- clique com o botão direito do *mouse* sobre o pacote do projeto e acesse a opção *New | Other...* No campo *Categories* escolha *Swing GUI Forms* e no campo *File Types* escolha *JDialog Form*. Aperte o botão *Next* e na tela que abrir, no campo *Class Name* digite *ClienteEdicao*;
- seguindo os mesmos passos do tópico anterior crie o formulário *EstadoEdicao* e o formulário *Sobre*.

Sua IDE NetBeans deve estar semelhante à apresentada na Figura 8.

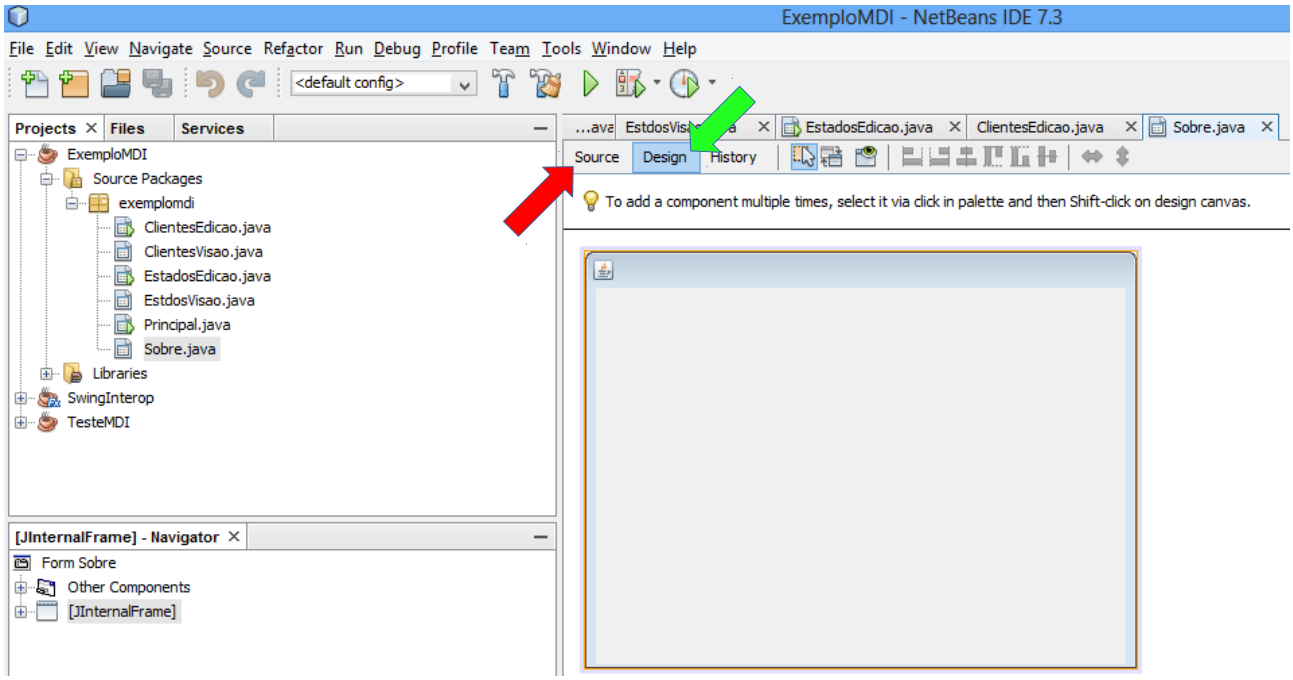

Figura 8: Visualização da IDE NetBeans após a criação dos formulários do sistema.

Agora, vamos fazer o seguinte:

- dê um clique duplo no formulário *ClienteEdicao*, mostrado na aba *Projects*, para que o mesmo se abra na janela central da IDE;
- clique na aba *Source* (seta vermelha na Figura 8) da aba do formulário aberto. Então poderá ser visto o código-fonte deste formulário. Vá até o método `public static void main(String args[])` e o apague por completo. Faça o mesmo no formulário *EstadoEdicao* e *Sobre*.

Explicação: já existe no formulário *Principal* um método *main*, assim sendo, não deverá existir outros no nosso projeto, pois, o método *main* deve ser único no projeto inteiro. A Figura 9 mostra (colorido em azul) todo o código-fonte do método *main* que deverá ser apagado na classe *ClienteEdicao*.

Muito bem, continuando nosso trabalho, dê um clique duplo no formulário *ClientesVisao* na aba *Projetcs*, se o formulário não estiver aparecendo, ou seja, se você estiver vendo o código-fonte do mesmo, então clique na aba *Design* deste formulário (seta verde na Figura 8). Em seguida, dê um clique simples sobre o formulário (apenas para selecioná-lo).

Vá até a aba *Properties* da IDE, clique na aba interna *Properties* (seta verde da Figura 4) e **marque** as seguintes propriedades:

- *closable,*
- *maximazable e*

- *resizable.*

Agora, na propriedade *title* escreva: *Visão de Clientes*. Na propriedade *name* escreva *frmClientesVisao*.

Siga os passos anteriores para configurar o formulário *EstadosVisao*.

```
26      @SuppressWarnings("unchecked")
27      Generated Code
45
46      /**
47       * @param args the command line arguments
48       */
49      public static void main(String args[]) {
50          /* Set the Nimbus look and feel */
51          Look and feel setting code (optional)
72
73          /* Create and display the dialog */
74          java.awt.EventQueue.invokeLater(new Runnable() {
75              public void run() {
76                  ClientesEdicao dialog = new ClientesEdicao(new javax.swing.JFrame(), true);
77                  dialog.addWindowListener(new java.awt.event.WindowAdapter() {
78                      @Override
80                      public void windowClosing(java.awt.event.WindowEvent e) {
81                          System.exit(0);
82                      }
83                  });
84                  dialog.setVisible(true);
85              }
86          });
87      }
88      // Variables declaration - do not modify
89      // End of variables declaration
90  }
```

Figura 9: Método main desnecessário.

Para os formulários *EstadoEdicao*, *ClienteEdicao* e *Sobre*, preencha as propriedades *title* e *name* de cada qual conforme a seguir:

- para o formulário *EstadoEdicao* a propriedade *title* deverá ser preenchida com *Edição de Estado*,
- a propriedade *name* com *dlgEstadoEdicao* e
- deverá ser marcada a propriedade *modal*.

Preenchendo os Eventos do Formulário Principal

Voltando ao formulário *Principal*, dê um clique simples sobre a palavra *Arquivo* no componente *menu*. Aparecerá o *submenu Sair*, clique nele e vá até a aba *Properties* da IDE NetBeans, então escolha a aba interna *Events* (seta amarela na Figura 4) e clique no

ComboBox que aparece logo a frente da propriedade *ActionPerformed*.

Ao clicar no *ComboBox*, aparecerá a opção *mnuSairActionPerformed* numa lista *drop-down*. Escolha esta opção e você automaticamente criará o evento *ActionPerformed* desta opção de *menu*. Logo, você será remetido ao código-fonte do formulário exatamente no interior do evento criado.

Para este evento você deve digitar a linha de código que se segue:

```
System.exit(0);
```

Siga os passos anteriores para criar os eventos *ActionPerformed* dos demais itens de *menu* do nosso *software*. No entanto, não preencha os eventos criados.

Independente da ordem de criação dos eventos, seu código-fonte deve estar semelhante ao apresentado na Figura 10, mantenha-o assim por enquanto.

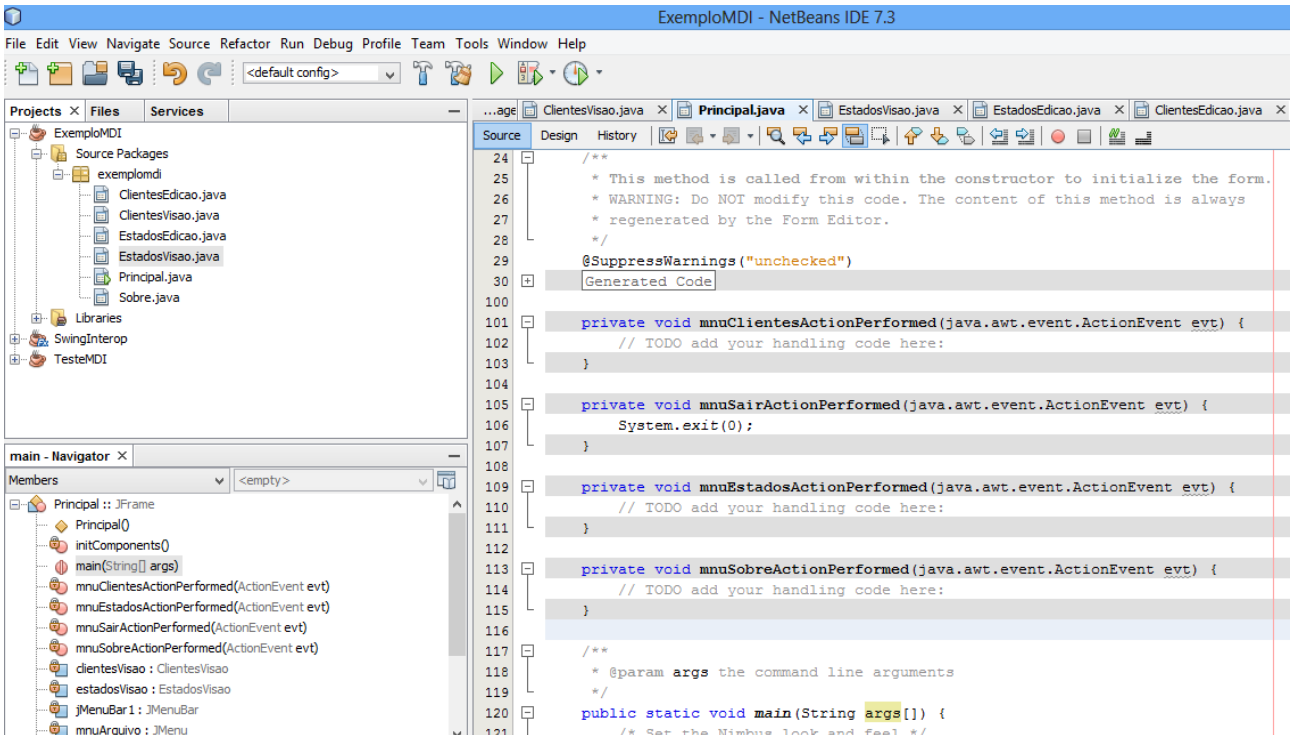

Figura 10: Criação dos eventos do menu do formulário Principal.

Agora vamos criar os atributos (ou variáveis de instância) correspondentes aos formulários que serão chamados/criados no formulário *Principal*. Para tanto, abaixo da declaração da classe no formulário *Principal*, escreva os atributos conforme mostrado na Figura 11 (chave vermelha).

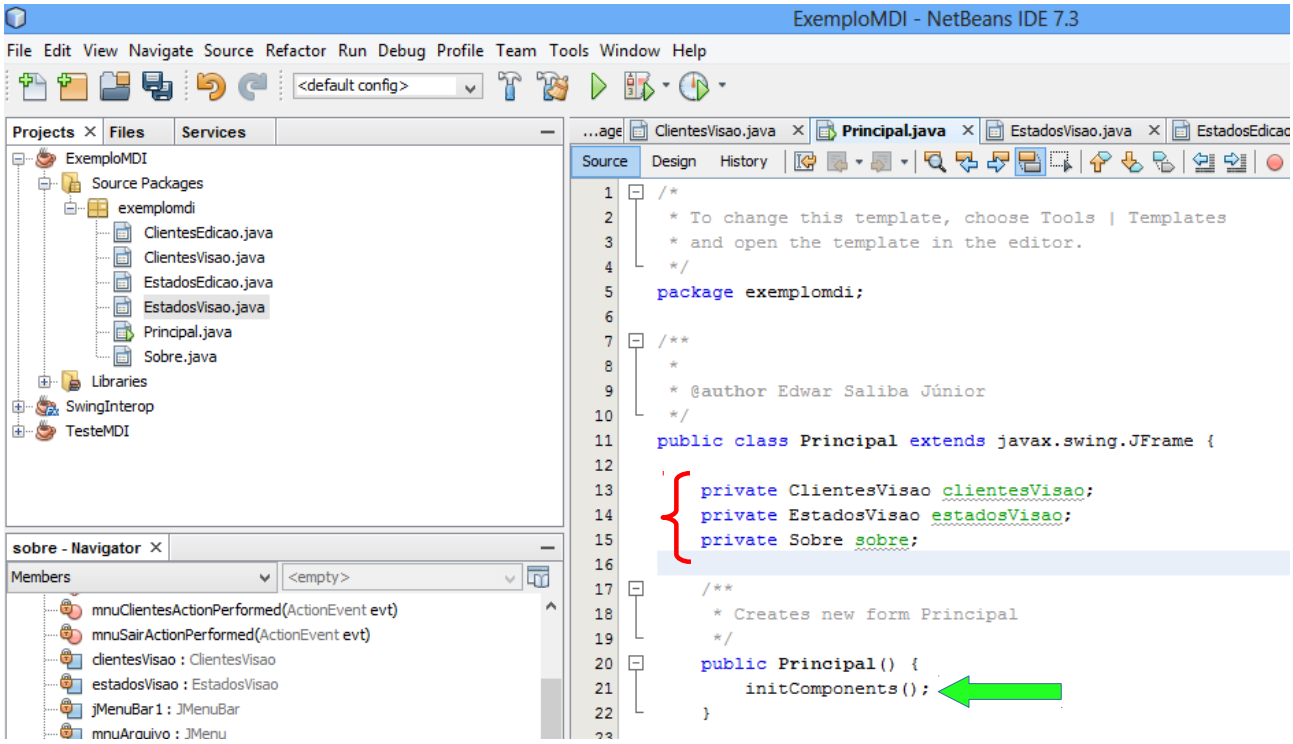

Figura 11: Criação das variáveis de instância no formulário Principal.

Bem, antes de mais nada, vamos aproveitar para maximizar nosso formulário *Principal* de forma automática. Para isto, dentro do método construtor, após a chamada do método `initComponents()` (seta verde na Figura 11), coloque a seguinte linha de código:

```
setExtendedState(JFrame.MAXIMIZED_BOTH);
```

O compilador apontará uma exceção para a classe *JFrame* declarada no comando. Faça a importação (*Ctrl + Shift + i*) do pacote *import javax.swing.JFrame* para resolver o problema;

Feito isto, vamos fazer a chamada de cada formulário nos eventos que foram criados para tal.

No evento *mnuClientesActionPerformed* coloque o código a seguir para que o formulário *ClientesVisao* seja criado quando o usuário do *software* clicar nesta opção do *menu*.

```
    private void mnuClientesActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            // Testa se o formulário já existe.
            if (clientesVisao == null) {
                // Cria o formulário.
                clientesVisao = new ClientesVisao();
```

```
            // Mostra o formulário.
            clientesVisao.setVisible(true);
            // Liga o formulário criado ao formulário Principal.
            getContentPane().add(clientesVisao);
            // Maximiza o formulário.
            clientesVisao.setMaximum(true);
        } else {
            if (!clientesVisao.isVisible()) {
                clientesVisao.setVisible(true);
                getContentPane().add(clientesVisao);
                clientesVisao.setMaximum(true);
            }
        }
    } catch (Exception ex) {
        System.out.println("Erro: " + ex);
    }
}
```

No evento , para a criação do formulário *EstadosVisao* preencha conforme abaixo:

```
private void mnuEstadosActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // Testa se o formulário já existe.
        if (estadosVisao == null) {
            // Cria o formulário.
            estadosVisao = new EstadosVisao();
            // Mostra o formulário.
            estadosVisao.setVisible(true);
            // Liga o formulário criado ao formulário Principal.
            getContentPane().add(estadosVisao);
            // Maximiza o formulário.
            estadosVisao.setMaximum(true);
        } else {
            if (!estadosVisao.isVisible()) {
                estadosVisao.setVisible(true);
                getContentPane().add(estadosVisao);
```

```
                    estadosVisao.setMaximum(true);
                }
            }
        } catch (Exception ex) {
            System.out.println("Erro: " + ex);
        }
    }
```

E para finalizar esta etapa, no evento `mnuSobreActionPerformed`, escreva o código a seguir:

```
    private void mnuSobreActionPerformed(java.awt.event.ActionEvent evt) {
        // Testa se o formulário já existe.
        if (sobre == null) {
            // Cria o formulário.
            sobre = new Sobre(this, true);
            // Mostra o formulário.
            sobre.setVisible(true);
        } else {
            if (!sobre.isVisible()) {
                sobre.setVisible(true);
            }
        }
    }
```

Componentizando os Formulários de Visão

Agora, abra o formulário *ClientesVisao*, vá a aba *Palette* e clique no componente *ToolBar* (seta verde na Figura 12) arraste o *mouse* e solte o componente no formulário. Coloque logo abaixo do componente *ToolBar*, um componente *Table* (seta vermelha na Figura 12) e cole no componente *ToolBar*, quatro componentes do tipo *Button* (seta azul na Figura 12).

Veja que foi adicionado, ao formulário *ClientesVisao*, um componente que mais parece uma planilha eletrônica. A ideia é esta, pois assim facilitaremos a visualização e manipulação dos dados que estarão no SGBD[1].

Utilizando dos mesmos passos acima, adicione também os componentes *ToolBar*, *Table* e os quatro *Buttons* ao formulário *EstadosVisao*.

1 SGBD – Sistema Gerenciador de Banco de Dados.

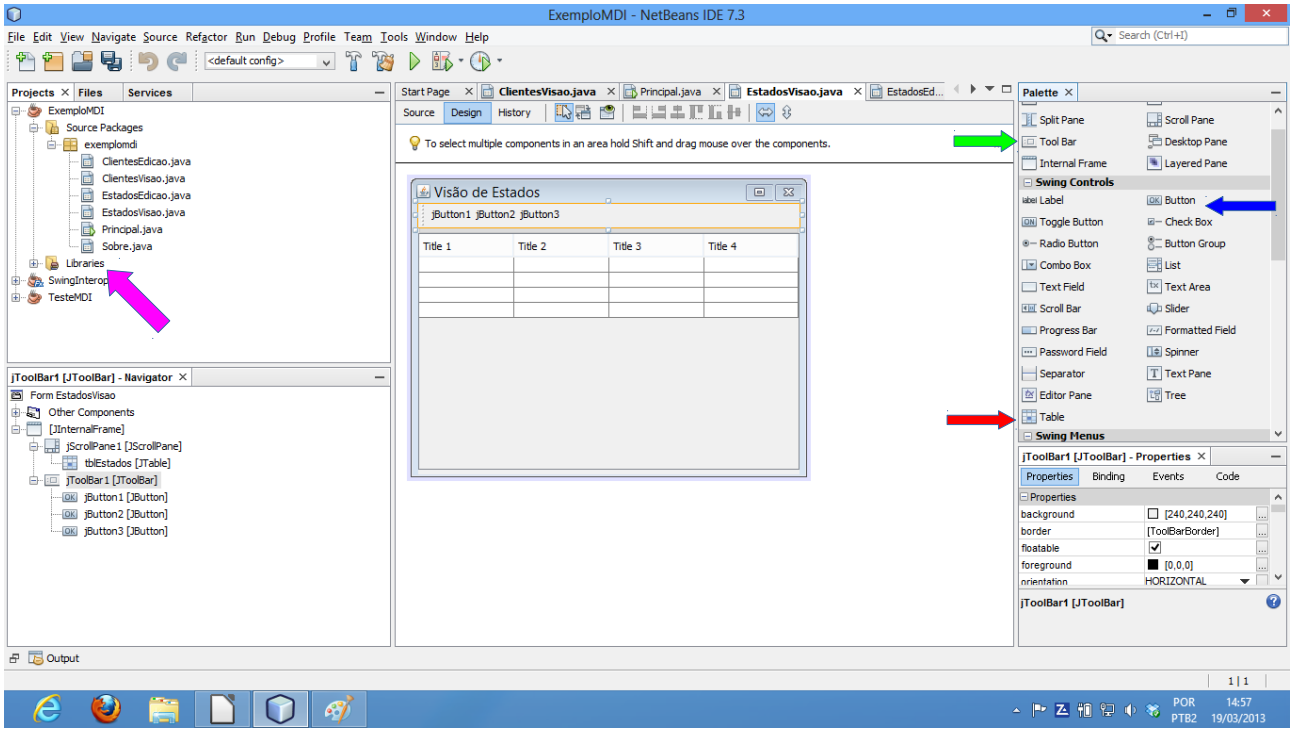

Figura 12: Adição de componentes ToolBar, Table e Buttons ao formulário.

Nomeando os Componentes dos Formulários de Visão

Vamos agora dar nomes aos bois. Abra o formulário *EstadosVisao*.

Clique no *ToolBar*, vá até a aba *Properties* e na propriedade *name* coloque *tlbVisaoEstados*. Na propriedade *floatable* desmarque o *CheckBox*. Vá até a aba interna *Code* e na propriedade *Variable Name* dê o mesmo nome, ou seja, *tlbVisaoEstados*.

Clique no componente *Table* que foi colocado no formulário e volte a aba *Properties*. Na propriedade *name* coloque *tblEstados* e marque a propriedade *autoCreateRowSorter* (esta propriedade criará um mecanismo de ordenação automática por colunas). Pulando para aba interna *Code*, coloque o mesmo nome na propriedade *Variable Name*.

Clique no componente *JButton1* que foi colocado no componente *ToolBar* e, na propriedade *name* coloque *btnNovo*, na propriedade *text* coloque *Novo* e, pulando para aba interna *Code*, coloque na propriedade *Variable Name* o nome *btnNovo*. Siga estes mesmos passos para os outros dois botões, porém, o segundo botão deverá se chamar *btnEditar*, o terceiro *btnExcluir* e o último *btnAtualizar*.

Repita os mesmos passos para a tela *ClientesVisao*.

Componentizando os Formulários de Edição

Agora, abra o formulário *ClienteEdicao* e vamos preenchê-lo com componentes de entrada de dados.

Você deverá pegar 7 componentes do tipo *Text Field* (setas vermelhas na Figura 13) e dispô-los de maneira organizada e elegante no formulário. Para cada componente *Text Field* você deverá acessar a aba *Properties* do componente e apagar o texto contido na propriedade *text* e em seguida pular para a aba *Code* e na propriedade *Variable Name* deverá dar um nome significativo a cada componente (*e. g.* para o componente *Text Field* que abrigará o código do cliente, o nome a ser dado será *tfdCodigo*, o componente que abrigará o nome do cliente *tfdNome* e assim por diante.)

Deverá pegar também um *Combo Box* (setas verdes na Figura 13) e colocá-lo próximo ao *Text Field* que será destinado ao preenchimento do nome do município do cliente. Vá a aba *Properties* e pule diretamente para a aba interna *Code*. Preencha a propriedade *Variable Name* com o seguinte nome *cbxEstado*.

Para cada um dos componentes acima, você deverá colocar um *Label* (setas amarelas na Figura 13), ou seja um rótulo para o usuário saber do que se trata aquele campo. Para cada *Label* colocado, você deverá ir a aba *Properties* e alterar a propriedade *text*, nela coloque uma palavra significativa para que o usuário do *software* saiba o que exatamente ele deverá colocar no campo imediatamente abaixo do *Label* (*e. g.* para o *Label* do campo Código, preencha a propriedade *text* com *Código*, para o *Label* do campo Nome preencha a propriedade *text* com a palavra *Nome* e assim por diante.), preencha também, pulando para a aba interna *Code*, a propriedade *Variable Name*, desta vez, da seguinte forma: para o *Label Código* preencha com *lblCodigo*, para o *Label Nome* preencha *lblNome* e assim por diante.

E por último cole dois componentes *Button* na tela (setas azuis na Figura 13) e nas propriedades *text* você deverá colocar *Salvar* para o primeiro botão e *Fechar* para o segundo botão. Pulando para a aba interna *Code* na propriedade *Variable Name* você deverá colocar *btnSalvar* para o primeiro e *btnFechar* para o segundo botão.

E assim terminamos a modelagem do formulário. Faça o mesmo para o formulário *EstadoEdicao*. Porém nele você só usará três componentes do tipo *Label*, três do tipo *Text Field* e dois do tipo *Button*. Não se esqueça de mudar as propriedades de cada qual da mesma forma que fizemos aqui.

Agora para terminarmos de construir os formulários que comporão este *software* falta apenas o formulário *Sobre*.

Neste formulário você só utilizará componentes do tipo *Label* e só precisará mudar as propriedades *text* de cada qual, como não faremos nada com os *Label's* então não haverá necessidade de mudar seus nomes. Cada *Label* deverá possuir o texto de acordo com a Figura 14 apresentada adiante. Caso você queira, no lugar do nome do professor, coloque o seu nome. Afinal de contas, quem está construindo o *software* é você!

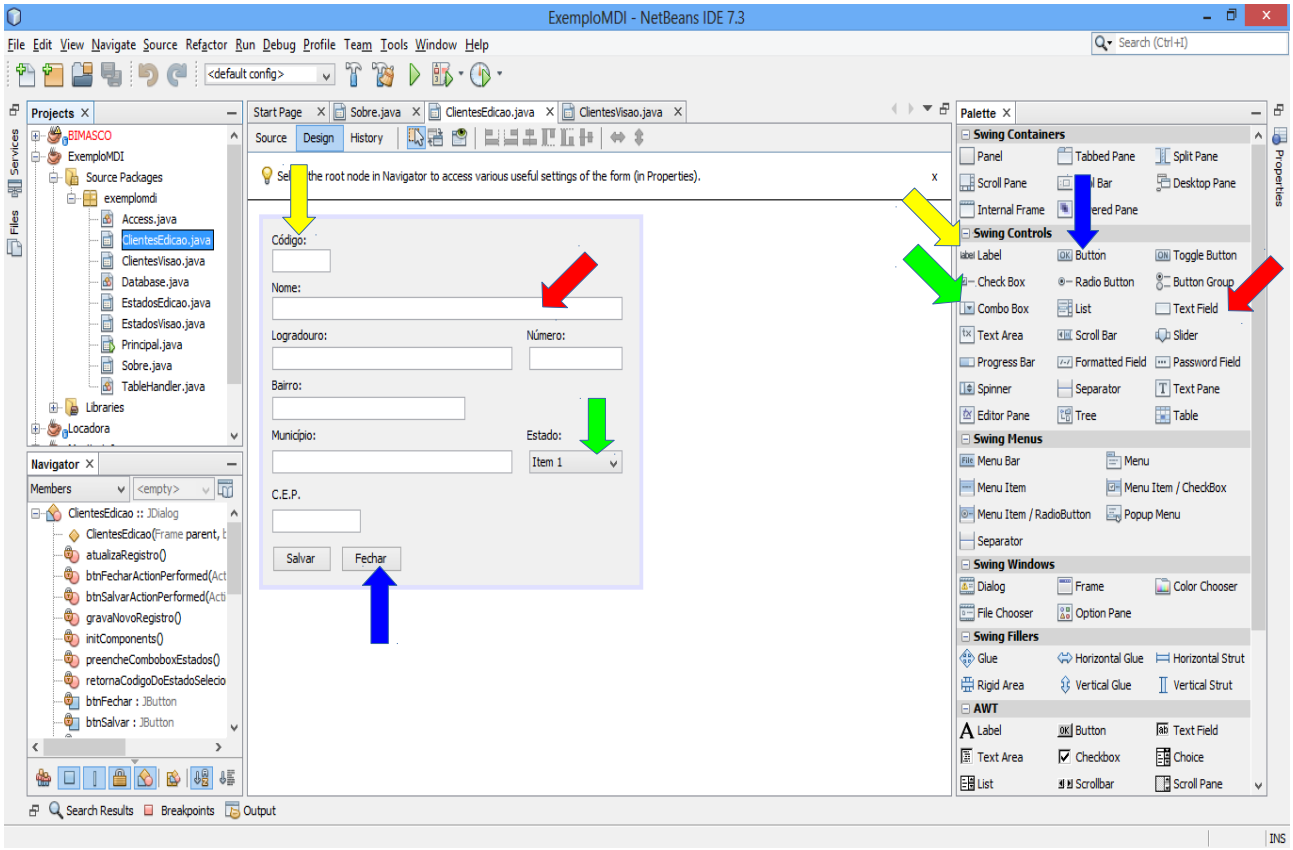

Figura 13: Formulário ClienteEdicao.

Figura 14: Formulário Sobre.

Acessando o Banco de Dados

Para preenchermos o componente *Table* dos formulários precisamos de dados. Estes estão armazenados nas tabelas criadas em nosso SGBD.

Para acessarmos o SGBD precisamos criar uma conexão com o mesmo. Para isto, vamos criar uma classe especial em nosso sistema que se encarregará desta conexão.

Antes de criarmos a conexão com o SGBD precisamos de uma biblioteca para nos auxiliar nesta conexão. Vamos adicioná-la ao nosso projeto. Clique com o botão direito do *mouse* sobre a pasta *Libraries* do projeto (seta rosa na Figura 12). Escolha a opção *Add Library...* e na tela que aparecer (Figura 15), escolha *PostgreSQL JDBC Driver* e clique no botão *Add Library*.

Feito isto, já teremos todas as condições para escrevermos a classe de conexão com o SGBD.

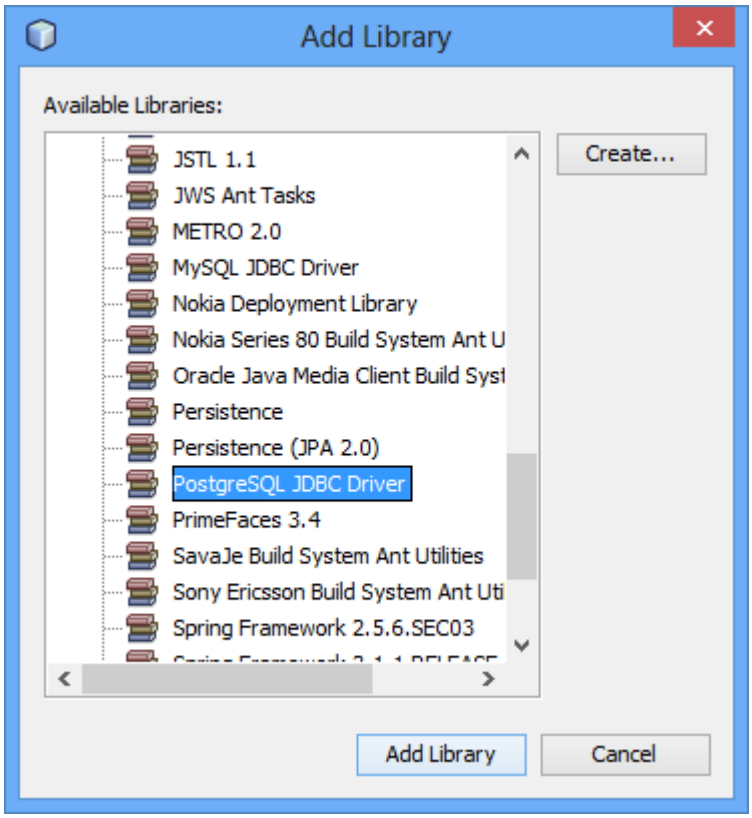

Figura 15: Tela para adição de bibliotecas ao projeto.

Vamos então a criação da classe *Access*. Para isto, clique com o botão direito no pacote *exemplimdi* do projeto do *software* e escolha a opção *New | Java Class...*

Na janela que aparecer (Figura 16) no campo *Class Name* digite *Access* e clique no botão *Finish*.

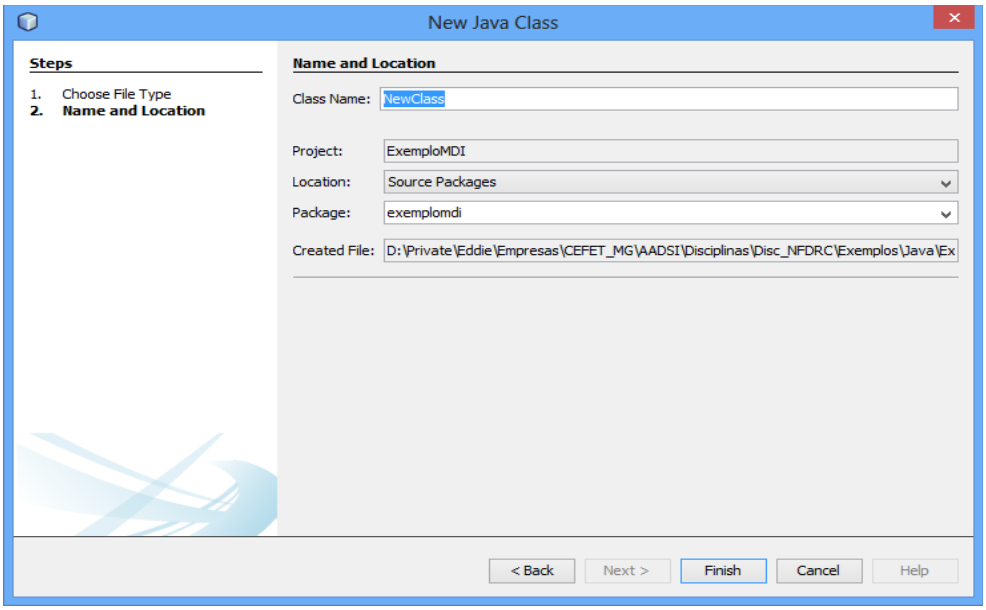

Figura 16: Janela para criação de nova classe.

Após a criação da nova classe, você deverá digitar o código-fonte apresentado na Figura 17.

Nesta classe, cria-se uma estrutura genérica para a montagem da chamada *String de Conexão*, que é uma *String* contendo todas as informações necessárias, na ordem exigida pelo SGBD, para a conexão com o mesmo.

Vale ressaltar que a *String de Conexão* devolvida pelo método *getURL* é de uso exclusivo do SGBD PostgreSQL para conexão do tipo JDBC.

Bom, vale ressaltar também que esta classe *Access* facilitará bastante a nossa vida, pois, não precisaremos lembrar a forma de montagem da *String de Conexão* do PostgreSQL.

Foi criada também, para facilitar mais ainda nossa vida de programador, a classe *Database*. Esta classe, que pode ser vista nas Figuras 18, 19, 20, 21 e 22, trás consigo diversos métodos que trataram de maneira eficiente nossas consultas ao SGBD PostgreSQL. Dê uma olhada e uma rápida estudada na classe *Database* e passe adiante, pois, esta classe será fornecida pelo professor.

```java
package exemplomdi;

/**
 *
 * @author Edwar Saliba Júnior
 */
public class Access {

    private String nomeSGBD;
    private String host;
    private int porta;
    private String nomeBD;
    private String usuario;
    private String senha;

    // Construtor para PostgreSQL.
    public Access(String nomeBD, String senha, String host) {
        this.nomeSGBD = "postgresql";
        if((host == null) || (host.trim().isEmpty()))
            this.host = "localhost";
        else
            this.host = host;
        this.porta = 5432;
        this.nomeBD = nomeBD;
        this.usuario = "postgres";
        this.senha = senha;
    }

    public String getURL() {
        return "jdbc:" + nomeSGBD + "://" + host + ":" + porta + "/" + nomeBD;
    }

    public String getNomeBD() {
        return nomeBD;
    }

    public String getUsuario() {
        return usuario;
    }

    public String getSenha() {
        return senha;
    }
}
```

Figura 17: Código-fonte da classe Access.

```java
package exemplomdi;

import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.util.ArrayList;

/**
 *
 * @author Edwar Saliba Júnior
 */
public final class Database {

    private Access informationDB;
    private Connection connectionDB;
    private Statement queryDB;
    private boolean enableMessages;

    public Database() throws SQLException{
        String dataBaseName = "ExemploMDI",
               password = "123456",
               host = "localhost";

        this.informationDB = new Access(dataBaseName, password, host);
        this.connectionToDB();
        this.enableMessages = true;
    }

    public void connectionToDB() throws SQLException  {
        try {
            this.startDriver();

            connectionDB = DriverManager.getConnection(
                    this.informationDB.getURL(),
                    this.informationDB.getUsuario(),
                    this.informationDB.getSenha());

            if (this.enableMessages)
                System.out.println("Connection with database '" +
                        this.informationDB.getNomeBD() + "' sucess completed.");

            connectionDB.setAutoCommit(false);
            this.queryDB = this.connectionDB.createStatement(
                    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
        } catch (ClassNotFoundException | SQLException e) {
            System.out.println(e.toString());
        }
    }

```

Figura 18: Classe Database - Parte 01/05.

```
52      public void startDriver() throws ClassNotFoundException {
53          Class.forName("org.postgresql.Driver");
54      }
55
56      public void insertValues(String tableName, String fieldsNames[],
57              String fieldsValues[]) throws SQLException {
58          String query = "INSERT INTO \"" + tableName + "\" (";
59          query += returnFieldsNames(fieldsNames) + ")";
60          query += " VALUES(";
61          query += returnValues(fieldsValues, true) + ")";
62
63          if (this.enableMessages)
64              System.out.println(query);
65
66          this.queryDB.executeUpdate(query);
67          this.connectionDB.commit();
68      }
69
70      public void deleteValues(String tableName, String condition) throws SQLException {
71          String query = "DELETE FROM \"" + tableName + "\"";
72
73          if (!condition.equals("")) {
74              query += " WHERE " + condition;
75          }
76
77          if (this.enableMessages)
78              System.out.println(query);
79
80          this.queryDB.execute(query);
81          this.connectionDB.commit();
82      }
83
84      public void updateValues(String tableName, String fields[], String values[],
85              String condition) throws SQLException {
86          String query = "UPDATE \"" + tableName + "\" SET ";
87          query += this.returnSetValues(fields, values);
88
89          if (!condition.equals("")) {
90              query += " WHERE " + condition;
91          }
92
93          if (this.enableMessages)
94              System.out.println(query);
95
96          this.queryDB.executeUpdate(query);
97          this.connectionDB.commit();
98      }
99
100     public ArrayList selection(String table, boolean putQuotationMarksOnTheFields,
101             String fields[], String condition) throws SQLException {
102         String query = "SELECT ";
```

Figura 19: Classe Database - Parte 02/05.

```
103             if(putQuotationMarksOnTheFields)
104                 query += returnFieldsNames(fields);
105             else
106                 query += returnValues(fields, putQuotationMarksOnTheFields);
107             query += " FROM " + table;
108 
109             if ((condition != null) && (!condition.equals(""))) {
110                 query += " WHERE " + condition;
111             }
112 
113             if (this.enableMessages)
114                 System.out.println(query);
115 
116             ResultSet resultSet = this.queryDB.executeQuery(query);
117 
118             ArrayList resultsList = new ArrayList();
119 
120             resultSet.beforeFirst();
121             while (resultSet.next()) {
122                 String[] row = new String[resultSet.getMetaData().getColumnCount()];
123                 for(int i = 0; i < resultSet.getMetaData().getColumnCount(); i++){
124                     row[i] = resultSet.getString(i + 1);
125                 }
126                 resultsList.add(row);
127             }
128             return resultsList;
129         }
130 
131         public ResultSet selection(String table, String fields[],
132                 boolean putQuotationMarksOnTheFields, String condition)
133                 throws SQLException {
134             String query = "SELECT ";
135             if (putQuotationMarksOnTheFields)
136                 query += returnFieldsNames(fields);
137             else
138                 query += returnValues(fields, putQuotationMarksOnTheFields);
139             query += " FROM \"" + table + "\"";
140 
141             if (!condition.equals("")) {
142                 query += " WHERE " + condition;
143             }
144 
145             ResultSet resultSet = this.queryDB.executeQuery(query);
146             resultSet.next();
147 
148             return(resultSet);
149         }
150 
151         public ResultSet selection(String query) throws SQLException {
152             ResultSet resultSet = this.queryDB.executeQuery(query);
153             resultSet.next();
```

Figura 20: Classe Database - Parte 03/05.

```
154
155             return(resultSet);
156         }
157
158         public String returnValues(String values[], boolean putQuotationMarks) {
159             String vals = "";
160
161             if(putQuotationMarks){
162                 for (int i = 0; i < values.length - 1; i++) {
163                     vals += "\'" + values[i] + "\', ";
164                 }
165
166                 vals += "\'" + values[values.length - 1] + "\'";
167             }
168             else{
169                 for (int i = 0; i < values.length - 1; i++) {
170                     vals += values[i] + ", ";
171                 }
172
173                 vals += values[values.length - 1];
174             }
175
176             return vals;
177         }
178
179         public String returnFieldsNames(String values[]) {
180             String vals = "\"";
181
182             for (int i = 0; i < values.length - 1; i++) {
183                 vals += values[i] + "\", \"";
184             }
185
186             vals += values[values.length - 1] + "\"";
187
188             return vals;
189         }
190
191         public String returnSetValues(String fields[], String values[]) {
192
193             String vals = "";
194
195             for (int i = 0; i < values.length - 1; i++) {
196                 vals += "\"" + fields[i] + "\" = " +
197                         (values[i].equals("") ? "\'\'" : "\'" + values[i] + "\'") + ", ";
198             }
199
200             vals += "\"" + fields[fields.length - 1] + "\" = " +
201                     (values[values.length - 1].equals("") ? "\'\'" : "\'" +
202                     values[values.length - 1] + "\'");
203
204             return vals;
```

Figura 21: Classe Database - Parte 04/05.

```
205         }
206
207         public void closeDBConnection() throws SQLException {
208             this.connectionDB.close();
209         }
210
211         @Override
213         public void finalize() throws SQLException, Throwable{
214             super.finalize();
215             closeDBConnection();
216         }
    }
```

Figura 22: Classe Database - Parte 05/05.

Precisaremos também de uma outra classe que facilitará bastante nossa vida enquanto programadores Java. Esta classe chamar-se *TableHandler* (Figuras 23 e 24) e nos ajudará na manipulação dos componentes *JTable*. Não se preocupe! Apesar de pequena, o professor fornecerá esta classe também.

```
1   package exemplomdi;
2
3   import javax.swing.JOptionPane;
4   import javax.swing.JTable;
5   import javax.swing.table.DefaultTableModel;
6
7   /**
8    *
9    * @author Edwar Saliba Júnior
10   */
11  public class TableHandler {
12      /**
13       * Adiciona campos em um componente "Table". Parâmetros: table - componente
14       * table onde deverá ser acrescentada uma nova linha com os valores contidos
15       * no parâmetro fields.
16       * @param table
17       * @param fields
18       */
19      public static void add(JTable table, Object[] fields) {
20          DefaultTableModel model = (DefaultTableModel) table.getModel();
21          int line = table.getSelectedRow();
22          table.removeEditor();
23
24          if ((line == -1) || ((line + 1) == model.getRowCount())) {
25              model.addRow(fields);
26          }
```

Figura 23: Classe TableHandler - Parte 01/02.

```
27          else {
28              if ((line + 1) < model.getRowCount()) {
29                  model.addRow(fields);
30                  Object transfer;
31                  for (int i = (model.getRowCount() - 2); i > line; i--) {
32                      for (int k = 0; k < model.getColumnCount(); k++) {
33                          transfer = model.getValueAt(i, k);
34                          model.setValueAt(transfer, (i + 1), k);
35                      }
36                  }
37
38                  for (int i = 0; i < model.getColumnCount(); i++) {
                        model.setValueAt(new String(""), (line + 1), i);
40                  }
41              }
42          }
43      }
44
45      /**
46       * Apaga a linha que estiver selecionada em um componente Table. O Parâmetro
47       * table espera receber o componente table que terá a linha apagada.
48       * @param table
49       */
50      public static void delete(JTable table) {
51          DefaultTableModel model = (DefaultTableModel) table.getModel();
52          int line = table.getSelectedRow();
53          if (line != -1) {
54              table.removeEditor();
55              model.removeRow(line);
56              if ((line + 1) <= model.getRowCount()) {
57                  table.getSelectionModel().addSelectionInterval(line, line);
58              }
59          }
60          else
61              JOptionPane.showMessageDialog(null, "Por favor selecione uma linha.");
62      }
63  }
```

Figura 24: Classe TableHandler - Parte 02/02.

Agora que já temos as classes de acesso ao SGBD e a classe que nos auxiliará com os componentes *JTable* criados, então está na hora de trabalharmos nos formulários que criamos.

Criando os Métodos e Eventos Necessários nos Formulários

Vamos abrir o formulário *Principal*.

Vamos criar um novo atributo neste formulário para que possamos acessar o SGBD PostgreSQL.

Abaixo dos atributos já criados, digite a linha de comando:

```
private Database dbObj;
```

E no método construtor da classe instancie o objeto.

```
dbObj = new Database();
```

Seu código-fonte deve estar semelhante ao apresentado na Figura 25.

```
8   *
9   * @author Edwar Saliba Júnior
10  */
11  public class Principal extends javax.swing.JFrame {
12
13      private ClientesVisao clientesVisao;
14      private EstadosVisao estadosVisao;
15      private Sobre sobre;
16      private Database dbObj;
17
18      /**
19       * Creates new form Principal
20       */
21      public Principal() throws SQLException {
22          initComponents();
23          setExtendedState(JFrame.MAXIMIZED_BOTH);
24          dbObj = new Database();
25      }
26
27      /**
28       * This method is called from within the constructor to initialize the form.
29       * WARNING: Do NOT modify this code. The content of this method is always
30       * regenerated by the Form Editor.
31       */
32      @SuppressWarnings("unchecked")
```

Figura 25: Formulário Principal - Novo atributo e instanciação do mesmo.

Agora, apesar dos formulários ainda não estarem prontos para isto, mas em breve estarão, nós vamos modificar a chamada dos métodos construtores na instanciação dos formulários *clientesVisao* e *estadosVisao*, para que estes contemplem o novo objeto criado para acesso ao SGBD.

Então, certifique-se que os métodos *mnuClientesActionPerformed* e *mnuEstadosActionPerformed* do seu formulário *Principal* estejam semelhantes aos métodos

mostrados nas Figuras 26 e 27.

```
110     private void mnuClientesActionPerformed(java.awt.event.ActionEvent evt) {
111         try {
112             // Testa se o formulário já existe.
113             if (clientesVisao == null) {
114                 // Cria o formulário.
115                 clientesVisao = new ClientesVisao(dbObj);
116                 // Mostra o formulário.
117                 clientesVisao.setVisible(true);
118                 // Liga o formulário criado ao formulário Principal.
119                 getContentPane().add(clientesVisao);
120                 // Maximiza o formulário.
121                 clientesVisao.setMaximum(true);
122             } else {
123                 if (!clientesVisao.isVisible()) {
124                     clientesVisao.setVisible(true);
125                     getContentPane().add(clientesVisao);
126                     clientesVisao.setMaximum(true);
127                 }
128             }
129         } catch (PropertyVetoException ex) {
130             System.out.println("Erro. A seguinte exceção foi gerada: " + ex);
131         } finally {
132             clientesVisao = null;
133         }
134     }
```

Figura 26: Evento mnuClientesActionPerformed.

Observe a seta vermelha na Figura 26 que mostra a chamada do método construtor da classe *ClientesVisao* já com o parâmetro *dbObj* que é o mecanismo de acesso e manipulação do SGBD.

Como já explicado anteriormente, o método construtor da classe *ClientesVisao* ainda não está preparado para receber este parâmetro. Portanto, você deve estar, neste momento, recebendo um aviso ou uma mensagem de erro em sua IDE. Não se preocupe, em breve faremos a alteração do método construtor da classe *ClientesVisao* e esta mensagem de erro desaparecerá automaticamente.

```
140     private void mnuEstadosActionPerformed(java.awt.event.ActionEvent evt) {
141
142         try {
143             // Testa se o formulário já existe.
144             if (estadosVisao == null) {
145                 // Cria o formulário.
146                 estadosVisao = new EstadosVisao(dbObj);
147                 // Mostra o formulário.
148                 estadosVisao.setVisible(true);
149                 // Liga o formulário criado ao formulário Principal.
150                 getContentPane().add(estadosVisao);
151                 // Maximiza o formulário.
152                 estadosVisao.setMaximum(true);
153             } else {
154                 if (!estadosVisao.isVisible()) {
155                     estadosVisao.setVisible(true);
156                     getContentPane().add(estadosVisao);
157                     estadosVisao.setMaximum(true);
158                 }
159             }
160         } catch (PropertyVetoException ex) {
161             System.out.println("Erro. A seguinte exceção foi gerada: " + ex);
162         } finally {
163             estadosVisao = null;
164         }
165     }
```

Figura 27: Evento mnuEstadosActionPerformed.

Da mesma forma que no evento mostrado na Figura 26, o evento *mnuEstadosActionPerformed* também está modificando a chamada do método construtor da classe *EstadosVisao*, o que poderá também gerar uma aviso ou uma mensagem de erro, mas isto será corrigido mais tarde. Portanto, mais uma vez, não se preocupe.

Ok! Terminamos por aqui as inovações no formulário *Principal*.

Modificando e Criando os Eventos no Formulário ClienteEdicao

Observe a classe *ClienteEdicao* apresentada nas Figuras 28, 29, 30 e 31. Compare com a que está no seu *software* e implemente o que estiver faltando.

```
1   package exemplomdi;
2
3   import java.sql.ResultSet;
4   import java.sql.SQLException;
5   import javax.swing.JOptionPane;
6
7   /**
8    *
9    * @author Edwar Saliba Júnior
10   */
11  public class ClientesEdicao extends javax.swing.JDialog {
12      private Object campos[];
13      private Database objDB;
14      private boolean novaEntradaDeDados;
15
16      /**
17       * Creates new form ClientesEdicao
18       */
19      public ClientesEdicao(java.awt.Frame parent, boolean modal, Database db,
20              Object[] camp) throws SQLException {
21          super(parent, modal);
22          initComponents();
23          objDB = db;
24          campos = camp;
25          novaEntradaDeDados = true;
26
27          preencheComboboxEstados();
28
29          /* Se o formulário for aberto para alteração, preenche os campos com os
30           * seus respectivos valores.
31           */
32          if(campos != null){
33              novaEntradaDeDados = false;
34              tfdCodigo.setText((String)campos[0]);
35              tfdCodigo.setEditable(false);  // Desabilita campo.
36              tfdNome.setText((String)campos[1]);
37              tfdLogradouro.setText((String)campos[2]);
38              tfdNumero.setText((String)campos[3]);
39              tfdBairro.setText((String)campos[4]);
40              tfdMunicipio.setText((String)campos[5]);
41              ffdCEP.setText((String)campos[6]);
42              cbxEstado.setSelectedItem((String)campos[7]);
43          }
44
45          // Coloca o cursor no primeiro componente habilitado quando o formulário
46          // for criado.
47          if(tfdCodigo.isEnabled())
48              tfdCodigo.requestFocus();
49          else
50              tfdNome.requestFocus();
```

Figura 28: Classe ClienteEdicao - Parte 01/04.

```
51
52              setDefaultCloseOperation(DISPOSE_ON_CLOSE);
53          }
54
55          /**
56           * This method is called from within the constructor to initialize the form.
57           * WARNING: Do NOT modify this code. The content of this method is always
58           * regenerated by the Form Editor.
59           */
60          @SuppressWarnings("unchecked")
61          Generated Code
215
216         private void btnSalvarActionPerformed(java.awt.event.ActionEvent evt) {
217             // Salva os dados digitados ou modificados no SGBD.
218             try {
219                 if(novaEntradaDeDados)
220                     gravaNovoRegistro();
221                 else
222                     atualizaRegistro();
223
224                 JOptionPane.showMessageDialog(this, "Dados salvos com sucesso!");
225                 btnFecharActionPerformed(evt);
226             } catch (SQLException ex) {
227                 JOptionPane.showMessageDialog(this, "Erro ao tentar acessar o Banco" +
228                         " de Dados.\n\nMensagem: " + ex);
229             }
230         }
231
232         private void btnFecharActionPerformed(java.awt.event.ActionEvent evt) {
233             setVisible(false);
234             dispose();
235         }
236
237         private void gravaNovoRegistro() throws SQLException{
238             String[] fields = { "cli_cod", "cli_nom", "cli_lgd", "cli_num",
239                                 "cli_bai", "cli_mun", "cli_cep", "est_cod" },
240                      values = { tfdCodigo.getText().trim(),
241                                 tfdNome.getText().trim(),
242                                 tfdLogradouro.getText().trim(),
243                                 tfdNumero.getText().trim(),
244                                 tfdBairro.getText().trim(),
245                                 tfdMunicipio.getText().trim(),
246                                 ffdCEP.getText().trim(),
247                                 retornaCodigoDoEstadoSelecionado().toString() };
248
249             objDB.insertValues("Cliente", fields, values);
250         }
251
252         private void atualizaRegistro() throws SQLException{
253             String[] fields = { "cli_nom", "cli_lgd", "cli_num",
```

Figura 29: Classe ClienteEdicao - Parte 02/04.

```
254                             "cli_bai", "cli_mun", "cli_cep", "est_cod" },
255                     values = { tfdNome.getText().trim(),
256                             tfdLogradouro.getText().trim(),
257                             tfdNumero.getText().trim(),
258                             tfdBairro.getText().trim(),
259                             tfdMunicipio.getText().trim(),
260                             ffdCEP.getText().trim(),
261                             retornaCodigoDoEstadoSelecionado().toString() };
262
263             objDB.updateValues("Cliente", fields, values, "cli_cod = " +
264                     tfdCodigo.getText().trim());
265
266         }
267
268         private void preencheComboboxEstados() throws SQLException{
269             ResultSet rs = objDB.selection("SELECT est_nom " +
270                                            "  FROM \"Estado\"");
271             cbxEstado.removeAllItems();
272             if(rs != null){
273                 while(rs.next()){
274                     cbxEstado.addItem(rs.getString("est_nom"));
275                 }
276             }
277         }
278
279         private String retornaCodigoDoEstadoSelecionado() throws SQLException{
280             String cod;
281             String nome;
282             nome = cbxEstado.getItemAt(cbxEstado.getSelectedIndex()).toString().trim();
283             ResultSet r = objDB.selection("select est_cod " +
284                                           "  from \"Estado\" " +
285                                           " where est_nom = '" + nome + "'");
286             r.first();
287            // if(r.next()){
288                 cod = r.getString("est_cod");
289            // }
290
291             return cod;
292         }
293
294         // Variables declaration - do not modify
295         private javax.swing.JButton btnFechar;
296         private javax.swing.JButton btnSalvar;
297         private javax.swing.JComboBox cbxEstado;
298         private javax.swing.JFormattedTextField ffdCEP;
299         private javax.swing.JLabel lblBairro;
300         private javax.swing.JLabel lblCEP;
301         private javax.swing.JLabel lblCodigo;
302         private javax.swing.JLabel lblEstado;
303         private javax.swing.JLabel lblLogradouro;
```

Figura 30: Classe ClienteEdicao - Parte 03/04.

```
304         private javax.swing.JLabel lblMunicipio;
305         private javax.swing.JLabel lblNome;
306         private javax.swing.JLabel lblNumero;
307         private javax.swing.JTextField tfdBairro;
308         private javax.swing.JTextField tfdCodigo;
309         private javax.swing.JTextField tfdLogradouro;
310         private javax.swing.JTextField tfdMunicipio;
311         private javax.swing.JTextField tfdNome;
312         private javax.swing.JTextField tfdNumero;
313         // End of variables declaration
314     }
```

Figura 31: Classe ClienteEdicao - Parte 04/04.

Modificando e Criando os Eventos no Formulário EstadoEdicao

Observe a classe *EstadoEdicao* apresentada nas Figuras 32, 33 e 34. Compare com a que está no seu *software* e implemente o que estiver faltando.

```
1   package exemplomdi;
2
3   import java.sql.SQLException;
4   import javax.swing.JOptionPane;
5
6   /**
7    *
8    * @author Edwar Saliba Júnior
9    */
10  public class EstadosEdicao extends javax.swing.JDialog {
11      private Object campos[];
12      private Database objDB;
13      private boolean novaEntradaDeDados;
14
15      /**
16       * Creates new form EstadosEdicao
17       */
18      public EstadosEdicao(java.awt.Frame parent, boolean modal, Database db,
19              Object[] camp) {
20          super(parent, modal);
21          initComponents();
22          objDB = db;
23          campos = camp;
24          novaEntradaDeDados = true;
25
26          /* Se o formulário for aberto para alteração, preenche os campos com os
```

Figura 32: Classe EstadoEdicao - Parte 01/02.

```
27          * seus respectivos valores.
28          */
29         if(campos != null){
30             novaEntradaDeDados = false;
31             tfdCodigo.setText((String)campos[0]);
32             tfdCodigo.setEnabled(false);
33             tfdNome.setText((String)campos[1]);
34             tfdSigla.setText((String)campos[2]);
35         }
36
37         // Coloca o cursor no primeiro componente habilitado quando o formulário
38         // for criado.
39         if(tfdCodigo.isEnabled())
40             tfdCodigo.requestFocus();
41         else
42             tfdNome.requestFocus();
43     }
44
45     /**...*/
50     @SuppressWarnings("unchecked")
51     Generated Code
131
132    private void btnSalvarActionPerformed(java.awt.event.ActionEvent evt) {
133        // Salva os dados digitados ou modificados no SGBD.
134        try {
135            if(novaEntradaDeDados)
136                gravaNovoRegistro();
137            else
138                atualizaRegistro();
139
140            JOptionPane.showMessageDialog(this, "Dados salvos com sucesso!");
141            btnFecharActionPerformed(evt);
142        } catch (SQLException ex) {
143            JOptionPane.showMessageDialog(this, "Erro ao tentar acessar o Banco" +
144                    " de Dados.\n\nMensagem: " + ex);
145        }
146    }
147
148    private void gravaNovoRegistro() throws SQLException{
149        String[] fields = { "est_cod", "est_nom", "est_sgl" },
150                 values = { tfdCodigo.getText().trim(),
151                            tfdNome.getText().trim(),
152                            tfdSigla.getText().trim() };
153
154        objDB.insertValues("Estado", fields, values);
155    }
156
157    private void atualizaRegistro() throws SQLException{
158        String[] fields = { "est_nom", "est_sgl" },
159                 values = { tfdNome.getText().trim(),
```

Figura 33: Classe EstadoEdicao - Parte 02/03.

```
160                             tfdSigla.getText().trim() };
161
162             objDB.updateValues("Estado", fields, values, "est_cod = " +
163                     tfdCodigo.getText().trim());
164         }
165
166         private void btnFecharActionPerformed(java.awt.event.ActionEvent evt) {
167             setVisible(false);
168             dispose();
169         }
170
171         // Variables declaration - do not modify
172         private javax.swing.JButton btnFechar;
173         private javax.swing.JButton btnSalvar;
174         private javax.swing.JLabel lblCodigo;
175         private javax.swing.JLabel lblNome;
176         private javax.swing.JLabel lblSigla;
177         private javax.swing.JTextField tfdCodigo;
178         private javax.swing.JTextField tfdNome;
179         private javax.swing.JTextField tfdSigla;
180         // End of variables declaration
181     }
```

Figura 34: Classe EstadoEdicao - Parte 03/03.

Pronto! Estamos com mais um formulário terminado. Partamos para o próximo.

Modificando e Criando os Eventos no Formulário ClientesVisao

Observe a classe *ClientesVisao* apresentada nas Figuras 35, 36, 37, 38 e 39. Compare com a que está no seu *software* e implemente o que estiver faltando.

```
1   package exemplomdi;
2
3   import java.sql.SQLException;
4   import java.util.ArrayList;
5   import javax.swing.JOptionPane;
6   import javax.swing.ListSelectionModel;
7   import javax.swing.table.DefaultTableModel;
8   import javax.swing.table.TableColumnModel;
9   import javax.swing.table.TableModel;
10  import javax.swing.table.TableRowSorter;
11
12  /**
13   *
14   * @author Edwar Saliba Júnior
15   */
16  public class ClientesVisao extends javax.swing.JInternalFrame {
17
18      private Database objDB;
19      private ClientesEdicao dlgClientesEdicao;
20      // Filtro de dados da JTable.
21      private DefaultTableModel dtm;
22      private TableRowSorter<TableModel> sorterTable;
23      // Nome dos campos.
24      private String clienteCodigo;
25      private String clienteNome;
26      private String clienteLogradouro;
27      private String clienteNumero;
28      private String clienteBairro;
29      private String clienteMunicipio;
30      private String clienteCEP;
31      private String estadoNome;
32
33      /**
34       * Creates new form Clientes
35       */
36      public ClientesVisao(Database objDB) {
37          this.objDB = objDB;
38
39          initComponents();
40
41          // Ajusta a largura das colunas.
42          TableColumnModel colModel;
43          // Permite selecionar apenas uma linha de cada vez.
44          this.tblClientes.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
45
46          this.clienteCodigo = "Código";
47          this.clienteNome = "Nome";
48          this.clienteLogradouro = "Logradouro";
49          this.clienteNumero = "Número";
50          this.clienteBairro = "Bairro";
```

Figura 35: Classe ClientesVisao - Parte 01/05.

```
51          this.clienteMunicipio = "Cidade";
52          this.estadoNome = "Estado";
53          this.clienteCEP = "C.E.P.";
54
55          // Cria o modelo da tabela.
56          this.tblClientes.setModel(new DefaultTableModel(new Object[][]{},
57                  new String[]{this.clienteCodigo,
58              this.clienteNome,
59              this.clienteLogradouro,
60              this.clienteNumero,
61              this.clienteBairro,
62              this.clienteMunicipio,
63              this.clienteCEP,
64              this.estadoNome}) {
65          });
66
67          // Ordenação e filtro do JTable.
68          this.dtm = (DefaultTableModel) this.tblClientes.getModel();
69          this.sorterTable = new TableRowSorter<>(this.tblClientes.getModel());
70          this.tblClientes.setRowSorter(sorterTable);
71
72          colModel = this.tblClientes.getColumnModel();
73
74          // Colunas visíveis.
75          this.tblClientes.getColumnModel().getColumn(colModel.getColumnIndex(
76                  this.clienteCodigo)).setPreferredWidth(15);
77          this.tblClientes.getColumnModel().getColumn(colModel.getColumnIndex(
78                  this.clienteNome)).setPreferredWidth(100);
79          this.tblClientes.getColumnModel().getColumn(colModel.getColumnIndex(
80                  this.clienteLogradouro)).setPreferredWidth(60);
81          this.tblClientes.getColumnModel().getColumn(colModel.getColumnIndex(
82                  this.clienteNumero)).setPreferredWidth(10);
83          this.tblClientes.getColumnModel().getColumn(colModel.getColumnIndex(
84                  this.clienteBairro)).setPreferredWidth(40);
85          this.tblClientes.getColumnModel().getColumn(colModel.getColumnIndex(
86                  this.clienteMunicipio)).setPreferredWidth(80);
87          this.tblClientes.getColumnModel().getColumn(colModel.getColumnIndex(
88                  this.clienteCEP)).setPreferredWidth(8);
89          this.tblClientes.getColumnModel().getColumn(colModel.getColumnIndex(
90                  this.estadoNome)).setPreferredWidth(2);
91
92          preencheVisaoDeClientes();
93      }
94
95      private void preencheVisaoDeClientes() {
96          ((DefaultTableModel) this.tblClientes.getModel()).setRowCount(0);
97          this.tblClientes.updateUI();
98
99          try {
100             ArrayList list = this.objDB.selection(
```

Figura 36: Classe ClientesVisao - Parte 02/05.

```
101                     "\"Cliente\" c, \"Estado\" e",
102                     false,
103                     new String[]{"c.\"cli_cod\"",
104                                  "c.\"cli_nom\"",
105                                  "c.\"cli_lgd\"",
106                                  "c.\"cli_num\"",
107                                  "c.\"cli_bai\"",
108                                  "c.\"cli_mun\"",
109                                  "c.\"cli_cep\"",
110                                  "e.\"est_nom\""},
111                                  "c.\"est_cod\" = e.\"est_cod\""
112                                  + " ORDER BY c.\"cli_nom\"");
113
114             for (int i = 0; i < list.size(); i++) {
115                 TableHandler.add(this.tblClientes, (Object[]) list.get(i));
116             }
117         } catch (SQLException ex) {
118             JOptionPane.showMessageDialog(this, "Erro ao tentar acessar o Banco" +
119                     " de Dados.\n\nMensagem: " + ex);
120         }
121     }
122
123     /**
124      * This method is called from within the constructor to initialize the form.
125      * WARNING: Do NOT modify this code. The content of this method is always
126      * regenerated by the Form Editor.
127      */
128     @SuppressWarnings("unchecked")
129     Generated Code
229
230     private void btnNovoActionPerformed(java.awt.event.ActionEvent evt) {
231         try{
232             if (dlgClientesEdicao == null) {
233                 try {
234                     dlgClientesEdicao = new ClientesEdicao(null, true, objDB,
235                             null);
236                 } catch (SQLException ex) {
237                     JOptionPane.showMessageDialog(this, "Erro ao tentar acessar" +
238                             " o Banco de Dados.\n\nMensagem: " + ex);
239                 }
240             }
241
242             this.dlgClientesEdicao.setVisible(true);
243         } finally {
244             dlgClientesEdicao = null;
245         }
246     }
247
248     private void btnEditarActionPerformed(java.awt.event.ActionEvent evt) {
249         try {
```

Figura 37: Classe ClientesVisao - Parte 03/05.

```
250             if (this.tblClientes.getSelectedRow() != - 1) {
251                 Object[] values = obtemLinhaDoComponenteTable();
252                 try {
253                     dlgClientesEdicao = new ClientesEdicao(null, true, objDB,
254                             values);
255                 } catch (SQLException ex) {
256                     JOptionPane.showMessageDialog(this, "Erro ao tentar acessar" +
257                             " o Banco de Dados.\n\nMensagem: " + ex);
258                 }
259 
260                 this.dlgClientesEdicao.setVisible(true);
261             } else {
262                 JOptionPane.showMessageDialog(this, "Antes de tentar editar, " +
263                         "selecione uma linha na tabela.");
264             }
265         } finally {
266             dlgClientesEdicao = null;
267         }
268     }
269 
270     private void btnExcluirActionPerformed(java.awt.event.ActionEvent evt) {
271         String valor;
272 
273         if (this.tblClientes.getSelectedRow() != - 1) {
274             Object[] ob = obtemLinhaDoComponenteTable();
275             if (JOptionPane.showConfirmDialog(this, "Confirma exclusão?",
276                     "Confirmação de Exclusão", JOptionPane.YES_NO_OPTION,
277                     JOptionPane.QUESTION_MESSAGE) == JOptionPane.YES_OPTION) {
278                 valor = (String) ob[0];
279                 int codigo = Integer.parseInt(valor);
280                 try {
281                     objDB.deleteValues("Cliente", "cli_cod = " + codigo);
282                     JOptionPane.showMessageDialog(this, "Dados excluídos com " +
283                             "sucesso!");
284                     btnAtualizarActionPerformed(evt);
285                 } catch (SQLException ex) {
286                     JOptionPane.showMessageDialog(this, "Erro ao tentar acessar" +
287                             " o Banco de Dados.\n\nMensagem: " + ex);
288                 }
289             }
290         } else {
291             JOptionPane.showMessageDialog(this, "Antes de tentar excluir, " +
292                     "selecione uma linha na tabela.");
293         }
294     }
295 
296     private void btnAtualizarActionPerformed(java.awt.event.ActionEvent evt) {
297         preencheVisaoDeClientes();
298     }
299 
```

Figura 38: Classe ClientesVisao - Parte 04/05.

```
300     private Object[] obtemLinhaDoComponenteTable() {
301         DefaultTableModel model = (DefaultTableModel) this.tblClientes.getModel();
302         TableColumnModel colModel = this.tblClientes.getColumnModel();
303         int row = this.tblClientes.getSelectedRow();
304         Object[] ob;
305 
306         ob = createEditableData(model.getValueAt(row, colModel.getColumnIndex(
307                 this.clienteCodigo)).toString(),
308              model.getValueAt(row, colModel.getColumnIndex(
309                 this.clienteNome)).toString(),
310              model.getValueAt(row, colModel.getColumnIndex(
311                 this.clienteLogradouro)).toString(),
312              model.getValueAt(row, colModel.getColumnIndex(
313                 this.clienteNumero)).toString(),
314              model.getValueAt(row, colModel.getColumnIndex(
315                 this.clienteBairro)).toString(),
316              model.getValueAt(row, colModel.getColumnIndex(
317                 this.clienteMunicipio)).toString(),
318              model.getValueAt(row, colModel.getColumnIndex(
319                 this.clienteCEP)).toString(),
320              model.getValueAt(row, colModel.getColumnIndex(
321                 this.estadoNome)).toString());
322 
323         return ob;
324     }
325 
326     private Object[] createEditableData(String codigo, String nome,
327             String logradouro, String numero, String bairro, String municipio,
328             String cep, String sigla) {
329         return (new Object[]{codigo, nome, logradouro, numero, bairro, municipio,
330             cep, sigla});
331     }
332     // Variables declaration - do not modify
333     private javax.swing.JButton btnAtualizar;
334     private javax.swing.JButton btnEditar;
335     private javax.swing.JButton btnExcluir;
336     private javax.swing.JButton btnNovo;
337     private javax.swing.JScrollPane jScrollPane1;
338     private javax.swing.JTable tblClientes;
339     // End of variables declaration
340 }
```

Figura 39: Classe ClientesVisao - Parte 05/05.

Modificando e Criando os Eventos no Formulário EstadosVisao

Observe a classe *EstadosVisao* apresentada nas Figuras 40, 41, 42 e 43. Compare com a que está no seu *software* e implemente o que estiver faltando.

```java
package exemplomdi;

import java.sql.SQLException;
import java.util.ArrayList;
import javax.swing.JOptionPane;
import javax.swing.ListSelectionModel;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumnModel;
import javax.swing.table.TableModel;
import javax.swing.table.TableRowSorter;

/**
 *
 * @author Edwar Saliba Júnior
 */
public class EstadosVisao extends javax.swing.JInternalFrame {
    private Database objDB;
    private EstadosEdicao dlgEstadosEdicao;

    // Filtro de dados da JTable.
    private DefaultTableModel dtm;
    private TableRowSorter<TableModel> sorterTable;

    // Nome dos campos.
    private String estadoCodigo;
    private String estadoNome;
    private String estadoSigla;

    /**
     * Creates new form EstdosVisao
     */
    public EstadosVisao(Database objDB) {
        this.objDB = objDB;

        initComponents();

        // Ajusta a largura das colunas.
        TableColumnModel colModel;
        // Permite selecionar apenas uma linha de cada vez.
        this.tblEstados.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        this.estadoCodigo = "Código";
        this.estadoNome = "Nome";
        this.estadoSigla = "Sigla";

        // Cria o modelo da tabela.
        this.tblEstados.setModel(new DefaultTableModel(new Object [][] { },
                new String [] { this.estadoCodigo,
                                this.estadoNome,
                                this.estadoSigla }){});
```

Figura 40: Classe EstadosVisao - Parte 01/04.

```
51
52          // Ordenação e filtro do JTable.
53          this.dtm = (DefaultTableModel)this.tblEstados.getModel();
54          this.sorterTable = new TableRowSorter<>(this.tblEstados.getModel());
55          this.tblEstados.setRowSorter(sorterTable);
56
57          colModel = this.tblEstados.getColumnModel();
58
59          // Colunas visíveis.
60          this.tblEstados.getColumnModel().getColumn(colModel.getColumnIndex(
61                  this.estadoCodigo)).setPreferredWidth(15);
62          this.tblEstados.getColumnModel().getColumn(colModel.getColumnIndex(
63                  this.estadoNome)).setPreferredWidth(80);
64          this.tblEstados.getColumnModel().getColumn(colModel.getColumnIndex(
65                  this.estadoSigla)).setPreferredWidth(25);
66
67          preencheVisaoDeEstados();
68      }
69
70      private void preencheVisaoDeEstados(){
71          ((DefaultTableModel)this.tblEstados.getModel()).setRowCount(0);
72          this.tblEstados.updateUI();
73
74          try {
75              ArrayList list = this.objDB.selection(
76                      "\"Estado\" e",
77                      false,
78                      new String[]{ "e.\"est_cod\"",
79                                    "e.\"est_nom\"",
80                                    "e.\"est_sgl\"" },
81                                    "1 = 1" +
82                                    " ORDER BY e.\"est_nom\"");
83
84              for(int i = 0; i < list.size(); i++)
85                  TableHandler.add(this.tblEstados, (Object[]) list.get(i));
86          } catch (SQLException ex) {
87              JOptionPane.showMessageDialog(this, "Erro ao tentar acessar o Banco" +
88                      " de Dados. \n\nMensagem: " + ex);
89          }
90      }
91
92      /**
93       * This method is called from within the constructor to initialize the form.
94       * WARNING: Do NOT modify this code. The content of this method is always
95       * regenerated by the Form Editor.
96       */
97      @SuppressWarnings("unchecked")
98      Generated Code
200
201     private void btnNovoActionPerformed(java.awt.event.ActionEvent evt) {
```

Figura 41: Classe EstadosVisao - Parte 02/04.

```
202         if(dlgEstadosEdicao == null){
203             dlgEstadosEdicao = new EstadosEdicao(null, true, objDB, null);
204         }
205
206         this.dlgEstadosEdicao.setVisible(true);
207     }
208
209     private void btnEditarActionPerformed(java.awt.event.ActionEvent evt) {
210         if(this.tblEstados.getSelectedRow() != - 1){
211             Object[] valores = obtemLinhaDoComponenteTable();
212
213             dlgEstadosEdicao = new EstadosEdicao(null, true, objDB, valores);
214
215             this.dlgEstadosEdicao.setVisible(true);
216         }
217         else
218             JOptionPane.showMessageDialog(this, "Antes de tentar editar, " +
219                     "selecione uma linha na tabela.");
220     }
221
222     private void btnExcluirActionPerformed(java.awt.event.ActionEvent evt) {
223         String valor;
224         int codigo;
225
226         if(this.tblEstados.getSelectedRow() != - 1){
227             Object[] ob = obtemLinhaDoComponenteTable();
228             if(JOptionPane.showConfirmDialog(this, "Confirma exclusão?",
229                     "Confirmação de Exclusão", JOptionPane.YES_NO_OPTION,
230                     JOptionPane.QUESTION_MESSAGE) == JOptionPane.YES_OPTION){
231                 valor = (String)ob[0];
232                 codigo = Integer.parseInt(valor);
233                 try {
234                     objDB.deleteValues("Estado", "est_cod = " + codigo);
235                     JOptionPane.showMessageDialog(this, "Dados excluídos com " +
236                             "sucesso!");
237                     btnAtualizarActionPerformed(evt);
238                 } catch (SQLException ex) {
239                     JOptionPane.showMessageDialog(this, "Erro ao tentar acessar" +
240                             " o Banco de Dados.\n\nMensagem: " + ex);
241                 }
242             }
243         }
244         else{
245             JOptionPane.showMessageDialog(this, "Antes de tentar excluir, " +
246                     "selecione uma linha na tabela.");
247         }
248     }
249
250     private void btnAtualizarActionPerformed(java.awt.event.ActionEvent evt) {
251         preencheVisaoDeEstados();
```

Figura 42: Classe EstadosVisao - Parte 03/04.

```
252     }
253
254     private Object[] obtemLinhaDoComponenteTable(){
255         DefaultTableModel model = (DefaultTableModel) this.tblEstados.getModel();
256         TableColumnModel colModel = this.tblEstados.getColumnModel();
257         int row = this.tblEstados.getSelectedRow();
258         Object[] ob;
259
260         ob = createEditableData(model.getValueAt(row, colModel.getColumnIndex(
261                                 this.estadoCodigo)).toString(),
262                             model.getValueAt(row, colModel.getColumnIndex(
263                                 this.estadoNome)).toString(),
264                             model.getValueAt(row, colModel.getColumnIndex(
265                                 this.estadoSigla)).toString());
266
267         return ob;
268     }
269
270     private Object[] createEditableData(String codigo, String nome, String sigla){
271         return(new Object []{ codigo, nome, sigla });
272     }
273
274     // Variables declaration - do not modify
275     private javax.swing.JButton btnAtualizar;
276     private javax.swing.JButton btnEditar;
277     private javax.swing.JButton btnExcluir;
278     private javax.swing.JButton btnNovo;
279     private javax.swing.JScrollPane jScrollPane1;
280     private javax.swing.JTable tblEstados;
281     private javax.swing.JToolBar tlbVisaoEstados;
282     // End of variables declaration
283 }
```

Figura 43: Classe EstadosVisao - Parte 04/04.

Se você chegou até aqui, parabéns! Você acabou de terminar seu *software* feito em Java. Teste-o e constate se está tudo funcionando perfeitamente.