Java Developer Projeto operações bancárias

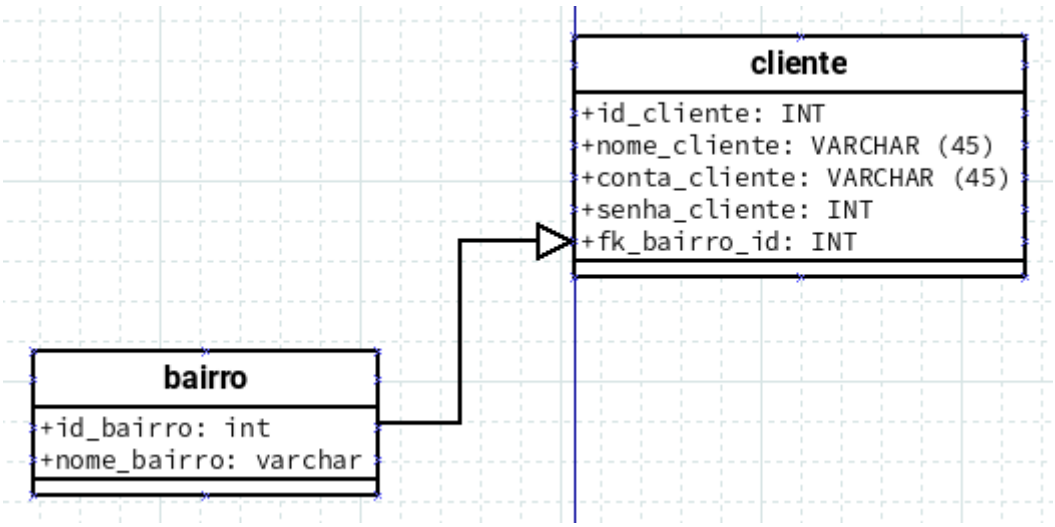
Tópicos

1 - Banco de dados

Nesse projeto utilizaremos um banco de dados para persistir os dados a serem manipulados

1.1 – Diagrama de entidade Relacionamento – DER

O objetivo desse diagrama é demonstrar o modelo de negócio no qual seu projeto foi desenvolvido, da origem dos dados que serão processados e o destino das informações que serão fornecidas.



1.2 –Script MySQL

Use o script para criar e popular seu banco de dados, copiando e colando esse recurso no console de um gerenciador MySQL ou Mariadb.

```sql
CREATE DATABASE db_banco_versatil;
USE db_banco_versatil;

CREATE TABLE bairro(
   id_bairro INT NOT NULL auto_increment,
   nome_bairro VARCHAR(45) NOT NULL
   ) ENGINE = innoDB DEFAULT CHARSET=utf8 ;

CREATE TABLE cliente(
   id_cliente INT NOT NULL auto_increment,
   nome_cliente VARCHAR(45) NOT NULL,
   fk_bairro_id INT NOT NULL,
   PRIMARY KEY(id_cliente),
   FOREING KEY (fk_bairro_id) REFERENCES bairro(id_bairro)
   ) ENGINE = innoDB DEFAULT CHARSET=utf8;
```

2 - Design Pattern MVC

O **MVC** (Model-View-Controller) é um padrão de design de software muito utilizado para a criação de interfaces gráficas. Ele divide a interface em três partes principais:

1. DTO (Objeto de transferência de dados):

- Representa a parte lógica da aplicação, contendo os dados e as regras de negócio.
- É responsável por armazenar, recuperar e manipular dados.

- Não se preocupa com a apresentação dos dados ou com a interação com o usuário.

2. VIEW (Formulários):

- É a parte visual da aplicação, responsável por apresentar os dados ao usuário.
- É composta por elementos gráficos como formulários, botões, menus, etc.
- Não se preocupa com a lógica da aplicação ou com a manipulação de dados.

3. DAO (Objeto de acesso a dados):

- É a ponte entre o modelo e a visão, responsável por controlar a interação entre os dois.
- Recebe as entradas do usuário (através da visão), processa-as e envia comandos para o modelo.
- Atualiza a visão de acordo com as mudanças no modelo.

2.1 – Conexão (DAO)

```java
package DAO;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.swing.JOptionPane;

public class ControleConexaoDAO {

    private static final String HOST = "localhost";
    private static final String USER = "root";
    private static final String PASSWORD = "root";
    private static final String DATA = "db_banco_versatil";

    public Connection conectarDadosDAO(){
        Connection conn = null;
        try{
            String url = "jdbc:mysql://"+HOST+":3306/"+DATA+"?";
            conn = DriverManager.getConnection(url,USER, PASSWORD);
        }catch(SQLException e){
            JOptionPane.showMessageDialog(null,e.getMessage());
        }
        return conn;
    }
}

```

É necessário atender as dependências para estabelecer a conexão com o banco de dados, para isso, adicione as lib’s java database connect (JDBC) e o mysql connector.jar

Escreva a classe de conexão, com o nome conexaoDAO

2.3 – Interfaces

A princípio criaremos uma interface, com o nome DadosBasicosINTER, contratando os métodos assessores, para serem implementados nas classes de transferência de dados, o que garantirá a aplicação do encapsulamento, pilar da orientação a objeto.

```
1   package INTER;
2
    public interface DadosBasicosINTER {
        int getId();
        int getBairro();
        String getNome();
        String getSenha();
8
        void setId(int id);
        void setBairro(int bairro);
        void setNome(String nome);
        void setSenha(String senha);
13  }
14
```

Faremos também uma interface para operações de CRUD

```
1   package INTER;
2
3   import java.util.List;
4
5   public interface ControleOperacoes<T> {
6       void incluir(T objeto);
7       void alterar(T objeto);
8       void excluir(T objeto);
9       T buscarPorId(int id);
10      List<T> buscarTodos();
11  }
12
```

2.4 – Data Transfer Object

Criaremos uma classe que implementará a interface de dados pessoais, visando transferir os dados vindos da VIEW para o banco de dados, através das classes DAO.

```java
package DTO;

import INTER.DadosBasicosINTER;

public class DadosBasicosDTO implements DadosBasicosINTER{

    protected int id = 0;
    protected int bairro = 0;
    protected String nome = "";
    protected String senha = "";

    @Override
    public int getId() {
        return id;
    }

    @Override
    public String getNome() {
        return nome;
    }

    @Override
    public int getBairro() {
        return bairro;
    }

    @Override
    public String getSenha() {
        return senha;
    }

    @Override
    public void setId(int id) {
        this.id = id;
    }

    @Override
    public void setNome(String nome) {
        this.nome = nome;

    }

    @Override
    public void setBairro(int bairro) {
        this.bairro = bairro;
    }

    @Override
    public void setSenha(String senha) {
        this.senha = senha;
    }

}
```

```java
package DTO;

public class ControleBairroDTO extends DadosBasicosDTO {

}

```

2.5 – Data Access Object

O acesso a dados, para realizar o CRUD (inserir, atualizar, deletar e consultar), é feita nas classes DAO, que usa os dados da classe DTO, no banco de dados.

```java
package DAO;

import java.util.List;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.sql.Connection;
import DTO.ControleBairroDTO;
import java.sql.SQLException;
import javax.swing.JOptionPane;
import java.sql.PreparedStatement;

public class ControleBairroDAO {

    Connection conn;
    ArrayList<ControleBairroDTO> listabairro;

    public ControleBairroDAO() {
        this.listabairro = new ArrayList<>();
    }

```

Declarações, imports e construtor]

```java
    public void atualizarBairro(ControleBairroDTO objbairrodto){
        conn = new ControleConexaoDAO().conectarDadosDAO();
        String sql = "UPDATE bairro SET nome_bairro = ? WHERE id_bairro = ?";

        try (PreparedStatement pstm = conn.prepareStatement(sql)){
            pstm.setString(1,objbairrodto.getNome());
            pstm.executeUpdate();
            if (pstm != null){
                pstm.close();
            }
        }catch(SQLException e){
            e.printStackTrace();
        }finally{
            try{
                if (conn != null);
                conn.close();
            }catch(SQLException e){
                JOptionPane.showMessageDialog(null,e.getMessage());
            }
        }

    }

```

Alterações

```java
    public void cadastrarBairro(ControleBairroDTO objbairrodto){
        conn = new ControleConexaoDAO().conectarDadosDAO();
        String sql = "INSERT INTO bairro (nome_bairro) VALUES (?)";

        try (PreparedStatement pstm = conn.prepareStatement(sql)){
            pstm.setString(1,objbairrodto.getNome());
            pstm.executeUpdate();
            if(pstm != null){
                pstm.close();
            }
        }catch(SQLException e){
                e.printStackTrace();
        }finally{
            try{
                if(conn != null){
                    conn.close();
                }
            }catch(SQLException e){
                JOptionPane.showMessageDialog(null, e.getMessage());
            }
        }
    }

```

Inclusões

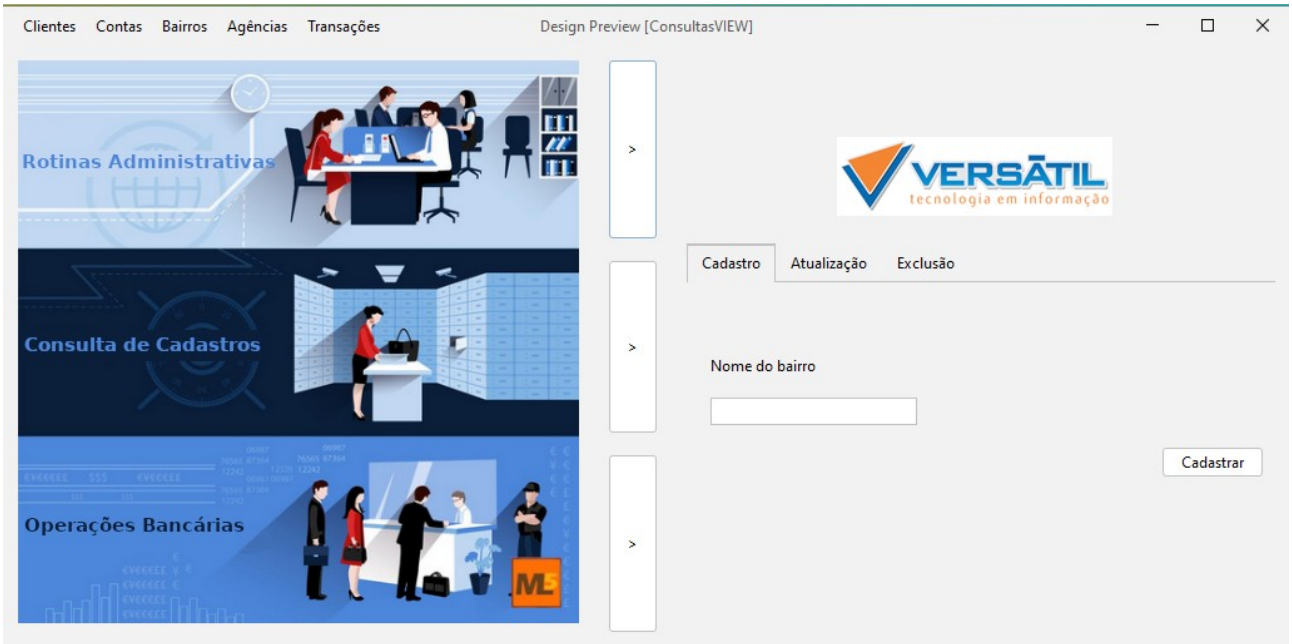
```java
    public void excluirBairro(ControleBairroDTO objbairrodto){
        conn = new ControleConexaoDAO().conectarDadosDAO();
        String sql = "DELET FROM bairro WHERE id_bairro = ?";

        try (PreparedStatement pstm = conn.prepareStatement(sql)){
            pstm.setInt(1,objbairrodto.getId());
            pstm.executeUpdate();
            if (pstm != null){
                pstm.close();
            }
        }catch(SQLException e){
            e.printStackTrace();
        }finally{
            try{
                if (conn != null){
                    conn.close();
                }
            }catch(SQLException e){
                JOptionPane.showMessageDialog(null,e.getMessage());
            }
        }
    }

```

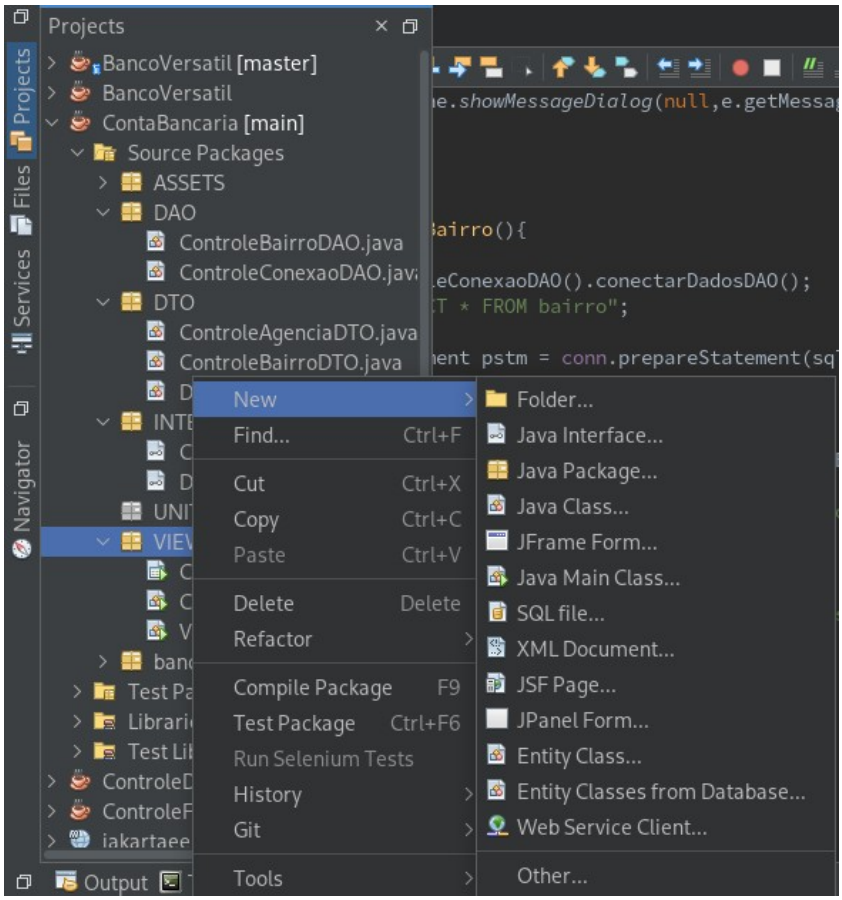
Exclusões

```java
    public List pesquisarBairro(){
        ResultSet rs;
        conn = new ControleConexaoDAO().conectarDadosDAO();
        String sql= "SELECT * FROM bairro";

        try(PreparedStatement pstm = conn.prepareStatement(sql)){
            rs = pstm.executeQuery();

            while(rs.next()){
                ControleBairroDTO objbairrodto = new ControleBairroDTO();
                objbairrodto.setId(rs.getInt("id_bairro"));
                objbairrodto.setNome(rs.getString("nome_bairro"));
                listabairro.add(objbairrodto);
            }
        }catch(SQLException erro){
            JOptionPane.showMessageDialog(null, "BairroDAO Pesquisar" + erro.getMessage());
        }
        return listabairro;
    }
}

```

2.5 – VIEW (JFrame For)

Para construção da camada de visualização, faremos a construção de um java frame, através da paleta java swing, criando a princípio um formulário para cadastro, atualizações e exclusões, um painel, contendo uma label com imagem, botões para as rotinas e operações e uma barra de menu com as entidades envolvidas.





Para programar as ações de formulário, basta clicar duas vezes no botão de comando, as instâncias das classes DAO e DTO podem ser escritas no método construtor da VIEW, para transferir as informações dos fieldtext's para as classes DTO, use os métodos getText(), depois basta chamar os métodos dos métodos DAO, passando o objeto como parâmetros.

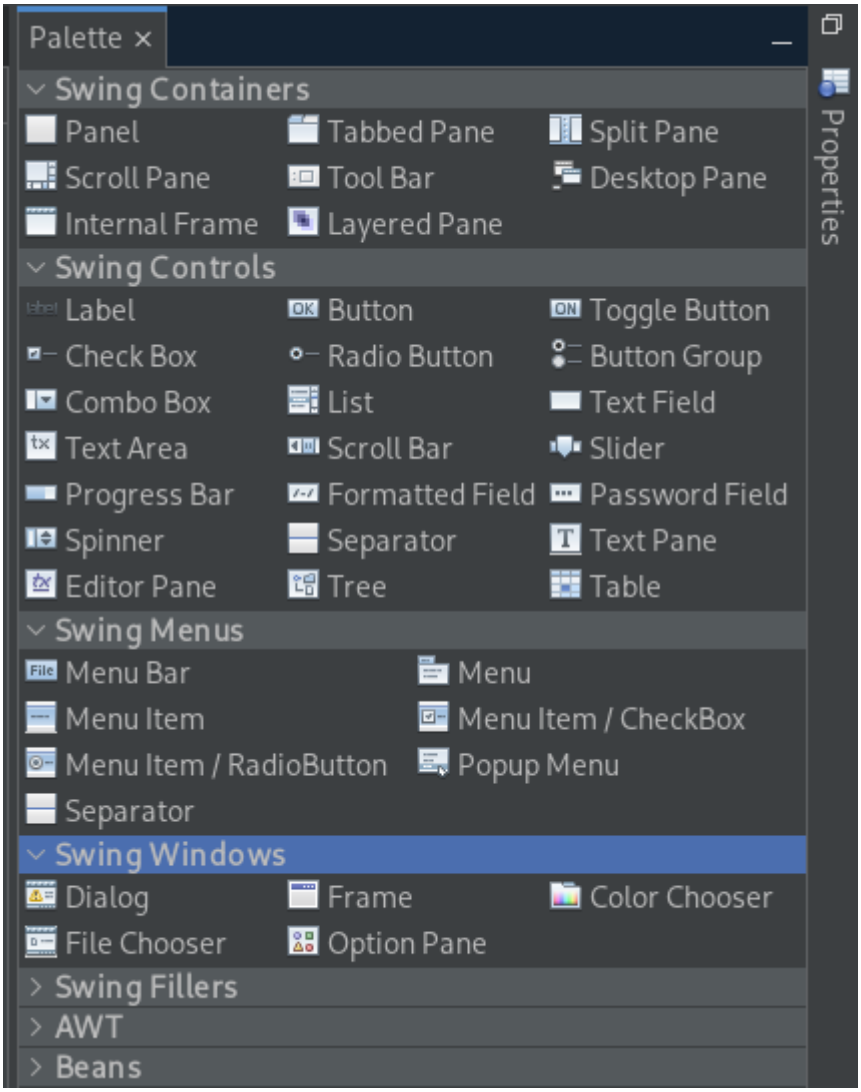
3 – Interface Gráfica

Com uso das ferramentas de design para contrução de interface gráfica do NetBeans, acessáveis no menu Tools, Palette, no submenu Swing/AWT Components, podemos programar a interface de forma visual, usando componentes, que podem, através de objetos chamar métodos, para transferência e acesso a dados.

3.1 - Swing Containers / Swing Controls

Componentes Swing

O pacote Swing é uma evolução do pacote AWT. Além de seus próprios componentes, esse pacote possui quase todos os componentes que já existiam no pacote AWT, porém com uma interface gráfica mais evoluida e com maiores efeitos. Os componentes Swing começam com a letra "J", assim um botão que no AWT se denomina Button, no Swing é JButton. Como dito antes, o pacote Swing possui componentes próprios, esses componentes estão entre telas mais aperfeiçoadas (JTabbedPane, JToolBar, JInternalFrame, JColorChooser...), novos botões e campos para dispor as informações na aplicação (JFormattedTextField, JProgressBar, JSpeener, JPasswordField, JTextPane, JTextArea...) e menus (JMenuBar, JMenu...).

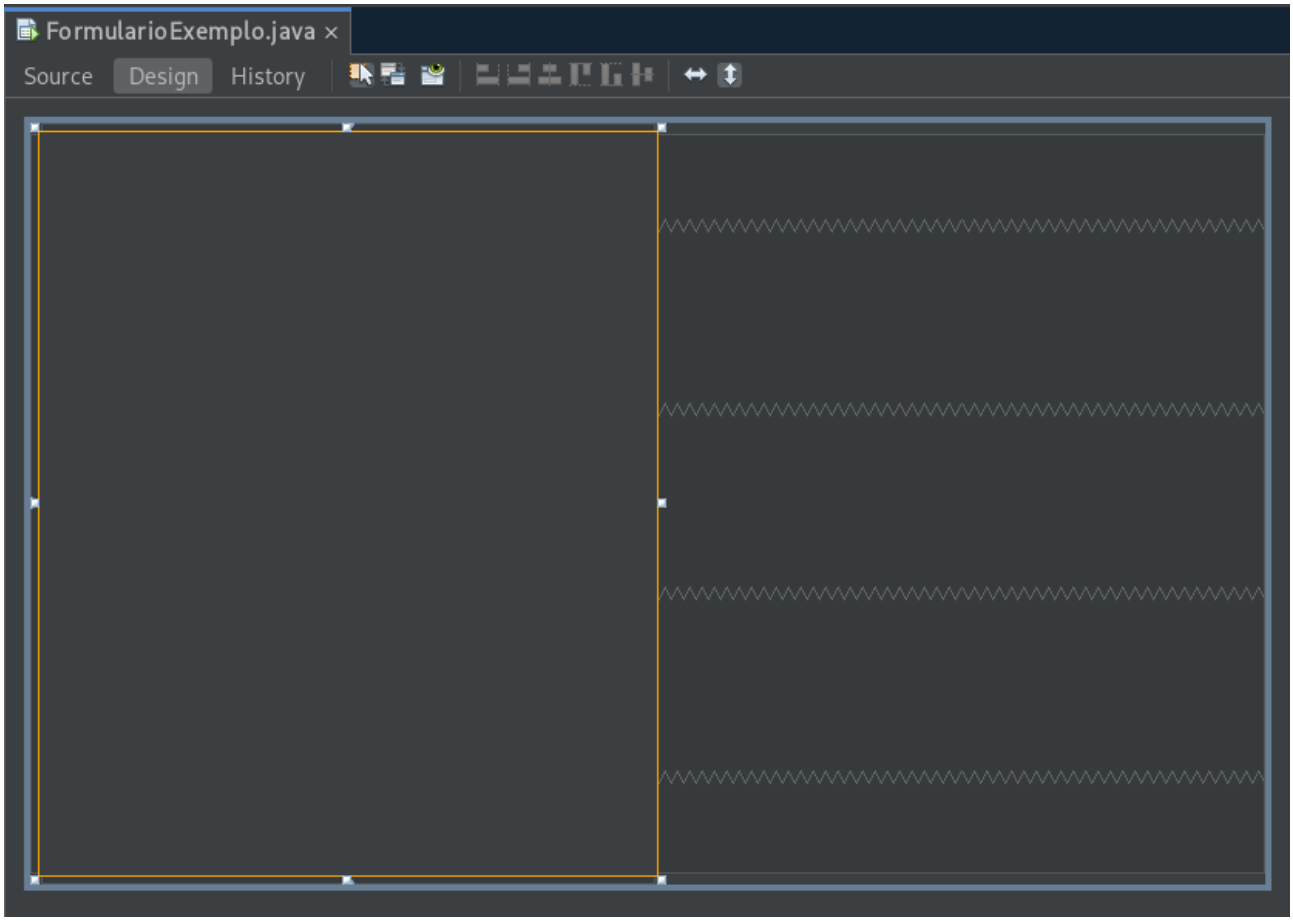


Nesse primeiro artigo vamos conhecer um pouco mais dos componentes relacionados a Contêiners.

3.1.1 - JPanel

O painel é um componente utilizado para fazer subdivisões na tela, ou para separar de forma organizada componentes dispostos na tela.

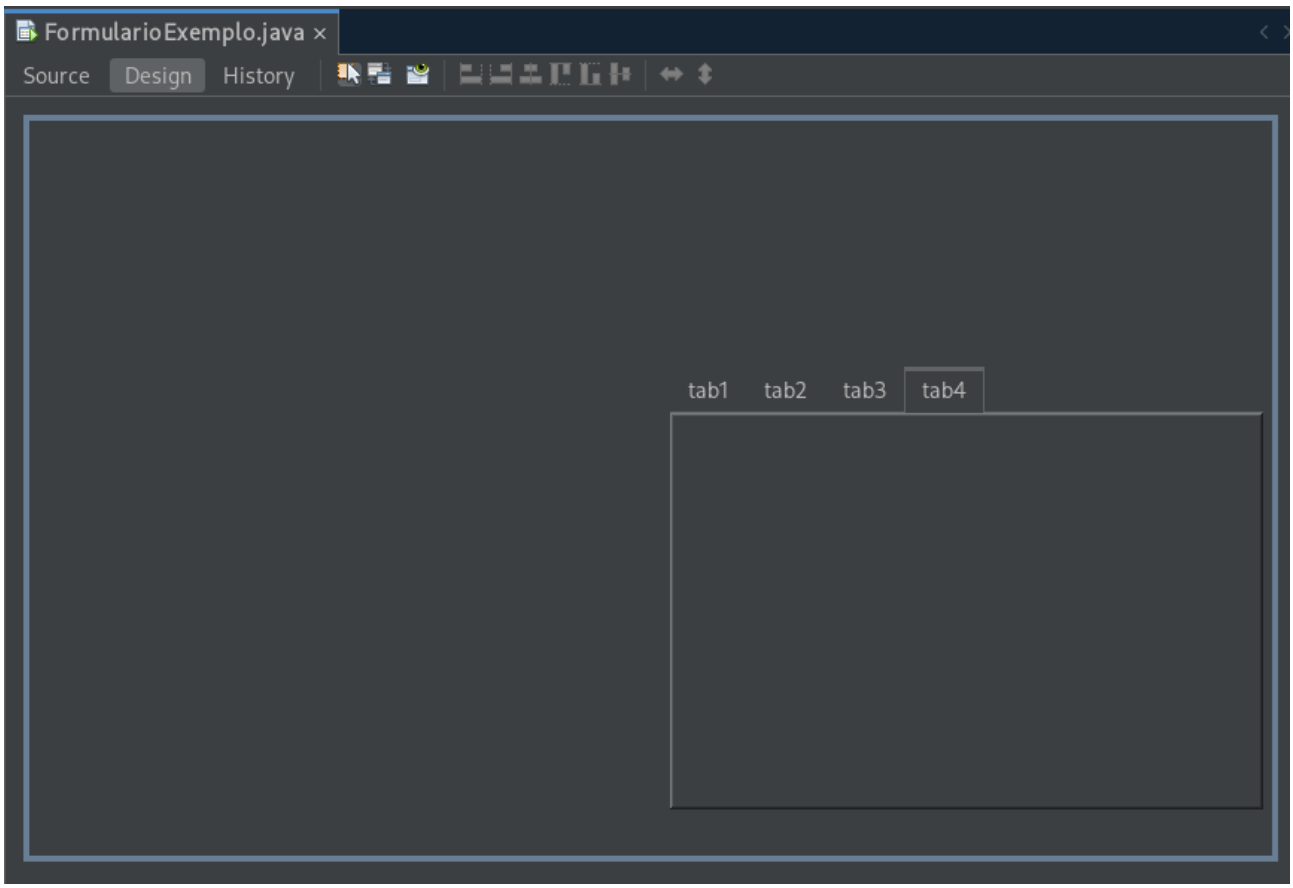
Propriedades mais utilizadas do Jpanel:



- setBackground(Color bg) - Define a cor do painel. A cor pode ser indicada por constantes da classe Color. Exemplo: Color.RED
- setBorder(Border border) - Define a borda do painel. A borda pode possuir título, ser mais espessa, mais densa, entre outras definições e é definida pela classe BorderFactory. Exemplo: BorderFactory.createTitledBorder("Título")
- add(Component comp) - Adiciona um componente no JPainel

3.1.2 - JTabbedPane

O JTabbedPane permite agrupar vários JPanels em um único componente formando uma aba para cada painel.



Propriedade mais utilizadas do JTabbedPane:

- addTab(String title, Component component) - adiciona um aba no componente, o primeiro parâmetro é o título da aba e o segundo é o componente que será parte da aba. Normalmente esse componente é o JPanel e dentro do JPanel vão os demais componentes da janela.
- addTab(String title, Icon icon, Component component) - tem o mesmo efeito que o método acima, porém permite setar um ícone para a aba.
- addTab(String title, Icon icon, Component component, String tip) - acrescenta ao método acima o ToolTipText, ultimo parâmetro do método. A String colococada no local do TipText será visível ao posicionar o cursor do mouse sobre o título da aba.

3.1.3 - Botões e campos de controle de dados

Sabemos que para desenvolver uma tela amigável para o usuário, são necessários diversos recursos visuais, entre eles botões, caixas de texto, locais para apresentar a saída e imagens.

Quanto mais organizada e distribuída for a tela, mais legível e confortável para se trabalhar.

Nesse artigo serão apresentados alguns dos principais componentes que são utilizados em janelas e suas respectivas propriedades.

3.1.4 - JLabel

O JLabel é utilizado para apresentar um texto "fixo" ao usuário. É muito comum utilizar o componente como rótulo para os demais componentes, ou seja, para identificar o que representa

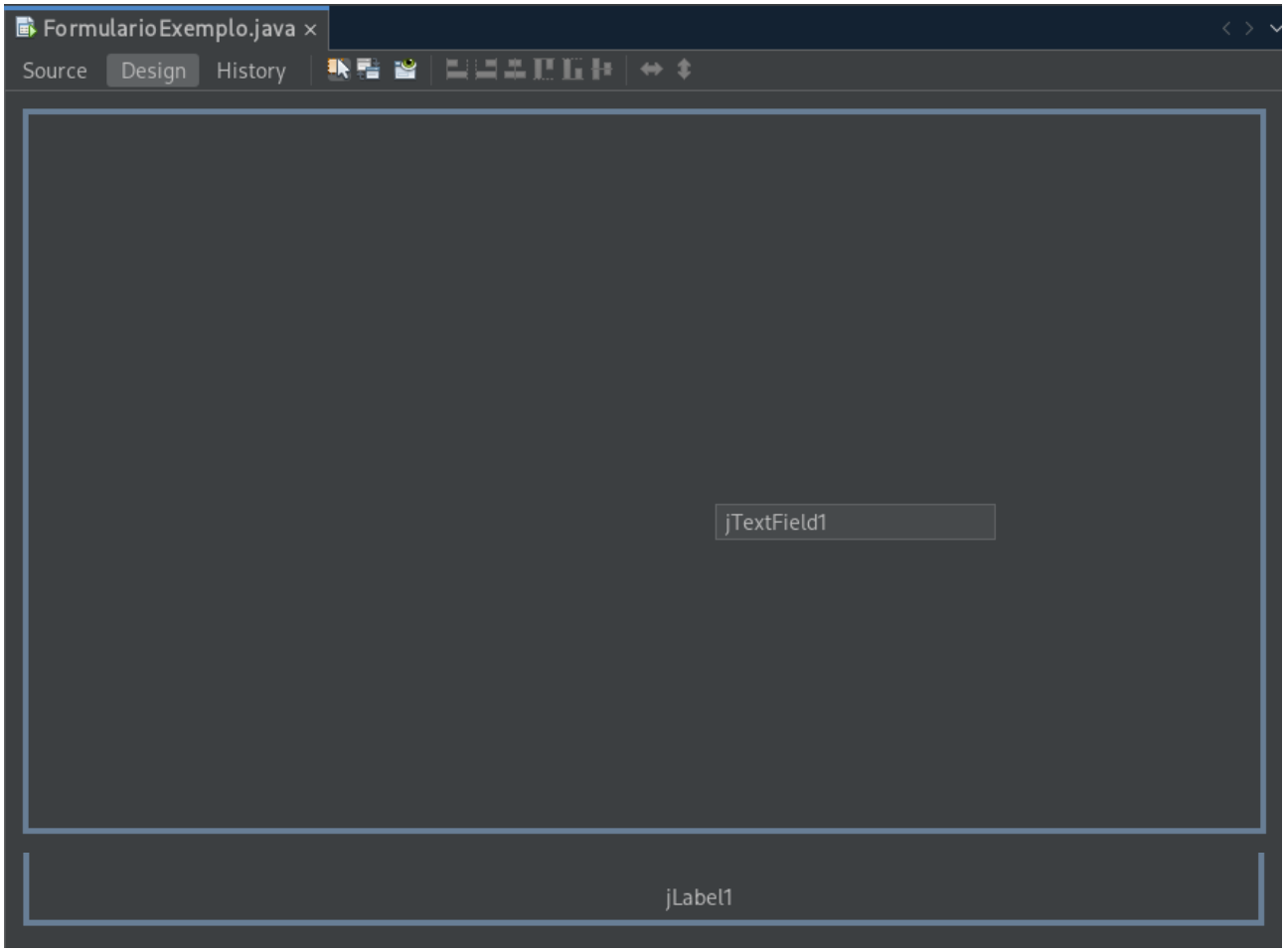
cada campo na tela. Pode ser utilizado também como saída de dados, onde o usuário irá visualizar o resultado final de alguma operação. O texto desse componente só pode ser alterado via código.

Propriedades mais utilizadas do JLabel:

- **setText(String text):**define o texto que aparecerá no JLabel.
- **setFont(Font fonte):**permite alterar a fonte do JLabel. A fonte é representada pela classe Font, onde é definido o tamanho, estilo e nome da fonte.
- **setIcon(IconImage icon):**permite adicionar um ícone ao JLabel. Muitas vezes o JLabel possui apenas um ícone ou imagem para deixar a tela mais amigável ao usuário. O ícone é representado pela classe IconImage. Para criar um IconImage deve ser passado como parâmetro o caminho da imagem.
- **setForeground(Color fg):**define a cor da fonte.

3.1.5 - JTextField e JPasswordField

O JTextField é um campo utilizado para a entrada de informações. Normalmente é uma caixa branca onde o usuário pode entrar com algum valor para que seja utilizado pelo algoritmo. O JPassowordField é idêntico ao JTextField com exceção de um fator, a aparência. O JPasswordField, como diz no nome, é um campo de senha. Assim, quando o usuário digita um texto esse texto fica codificado.

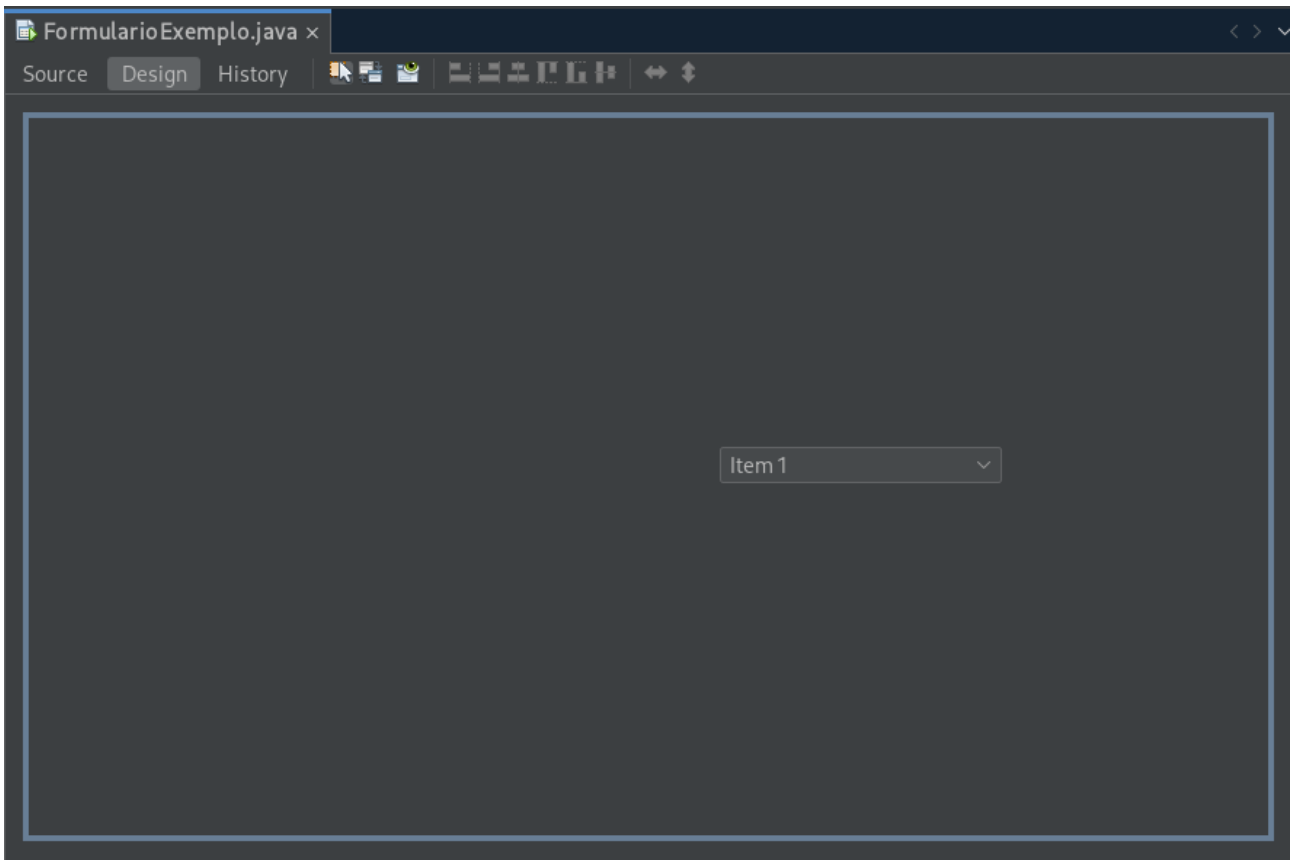


Propriedades mais utilizadas do JTextField/JPasswordField:

- **setText(String text):**adiciona um texto no JTextField.
- **setFont(Font font):**permite alterar a fonte do JTextField. A fonte é representada pela classe Font, onde é definido o tamanho, estilo e nome da fonte.
- **setForeground(Color fg):**define a cor da fonte.
- **setBackground(Color bg):**define a cor do fundo do JTextField.
- **setEditable(boolean b):**se o parâmetro for "true", o usuário pode alterar o texto do JTextField. Se for "false" o usuário não pode alterar o texto.
- **setEnabled(boolean enabled):**se o parâmetro for "false" o componente fica hachurado, ou seja, fica bloqueado para o usuário copiar um texto ou digitar. Se for "true" fica desbloqueado.
- **setFocusable(boolean focusable):**seta o foco para o JTextField. Para que o foco seja setado o parâmetro deve ser true.
- **selectionColor(Color color):**define a cor que o JTextField ficará ao selecionar seu texto. O padrão da cor de seleção é azul.
- **selectedTextColor(Color color):**define a cor que o texto do JTextField ficará ao ser selecionado. O padrão da cor do texto selecionado é branco.

3.1.5 - JComboBox

O JComboBox é utilizado como seletor de dados, onde são configurados determinados campos pelo programador e o usuário pode escolher um deles. Um exemplo prático seria um campo para escolher o estado, onde o usuário só poderia escolher um dos estados definidos. A caixão de seleção também trabalha com índices.

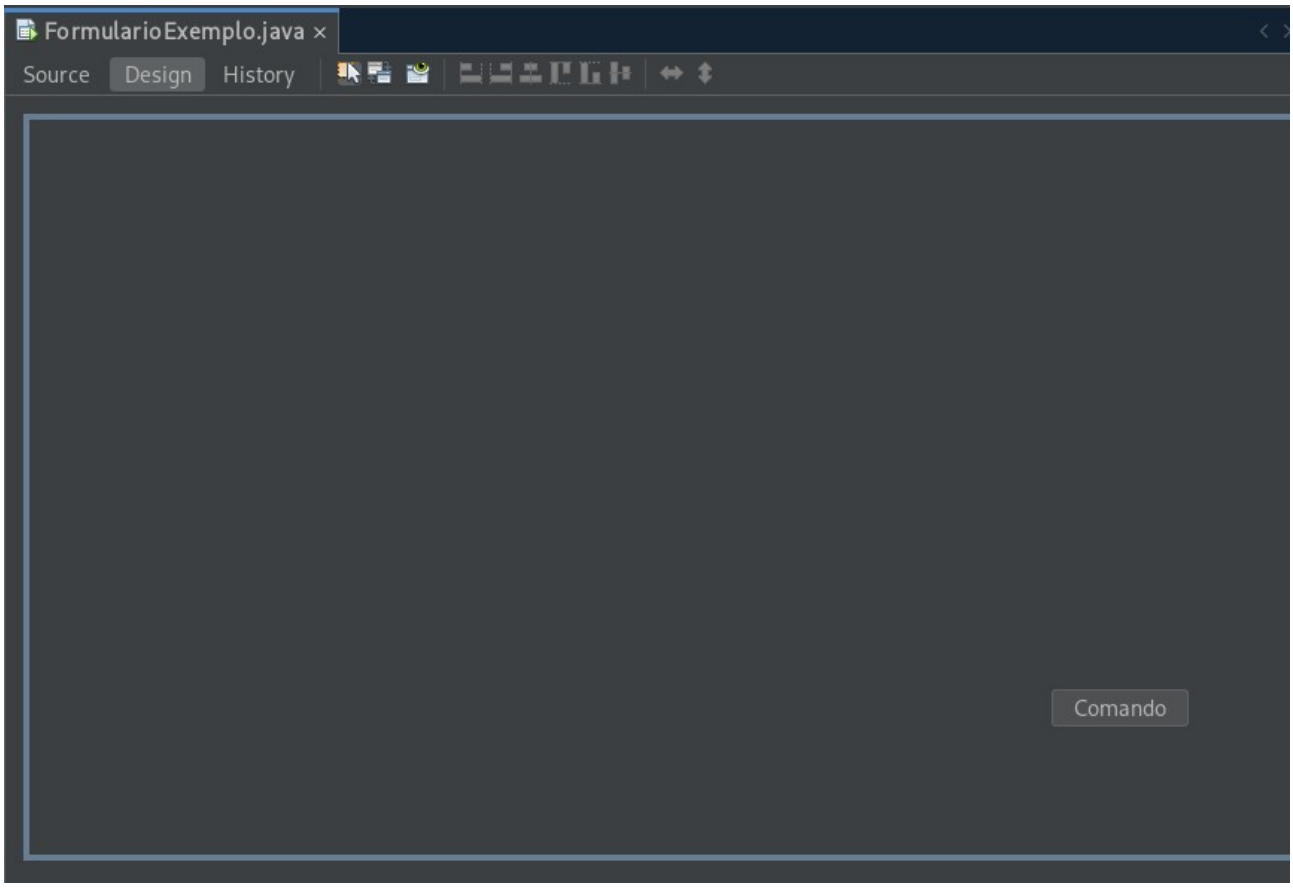


Propriedades mais utilizadas do JComboBox:

- **addItem(Object anObject):**utilizado para adicionar um item/campo no JComboBox. O objeto passado no parâmetro normalmente é uma String.
- **setSelectedIndex(int anIndex):**define qual o item do JComboBox começa selecionado.
- **setFont(Font font):**permite alterar a fonte. A fonte é representada pela classe Font, onde é definido o nome, estilo e tamanho da fonte.

3.1.6 - JButton

O JButton é o componente correspondente ao botão. É utilizado para que seja efetuada alguma tarefa ou função ao ser clicado.



Propriedades mais importantes do JButton:

- **setText(String string):**define o texto do botão.
- **setFont(Font font):**permite alterar a fonte do JButton. A fonte é representada pela classe Font, onde é definido o tamanho, estilo e nome da fonte.
- **setBackground(Color bg):**define a cor do fundo do botão.
- **setForeground(Color fg):**define a cor da fonte do botão.
- **setEnabled(Boolean enabled):**quando o parâmetro é “true” permite-se que o usuário clique no botão, quando é “false” o botão fica bloqueado.
- **setIcon(Icon defaultIcon):**seta um ícone para o botão. O ícone pode ser representado pela classe ImageIcon e deve ser passado o caminho da imagem de onde está o ícone.

3.2 – Programação da VIEW

3.2.1 – Programação de operações de CRUD na VIEW

Ao clicar duas vezes em um JButton podemos programar o evento onClick, com operações de CRUD, usaremos com exemplo a entidade bairro, como exemplo:

3.2.2 – INSERT

```
362     private void btncadbairroActionPerformed(java.awt.event.ActionEvent evt) {
363         ControleBairroDTO objbairrodto = new ControleBairroDTO();
364         ControleBairroDAO objbairrodao = new ControleBairroDAO();
365         String nomebairro = txtcadbairro.getText();
366 
367         objbairrodto.setNome(nomebairro);
368         objbairrodao.cadastrarBairro(objbairrodto);
369         listarBairro();
370 
371         txtcadbairro.setText("");
372         lblstatus.setText("Cadastrado com sucesso");
373     }
```

3.2.3 – UPDATE

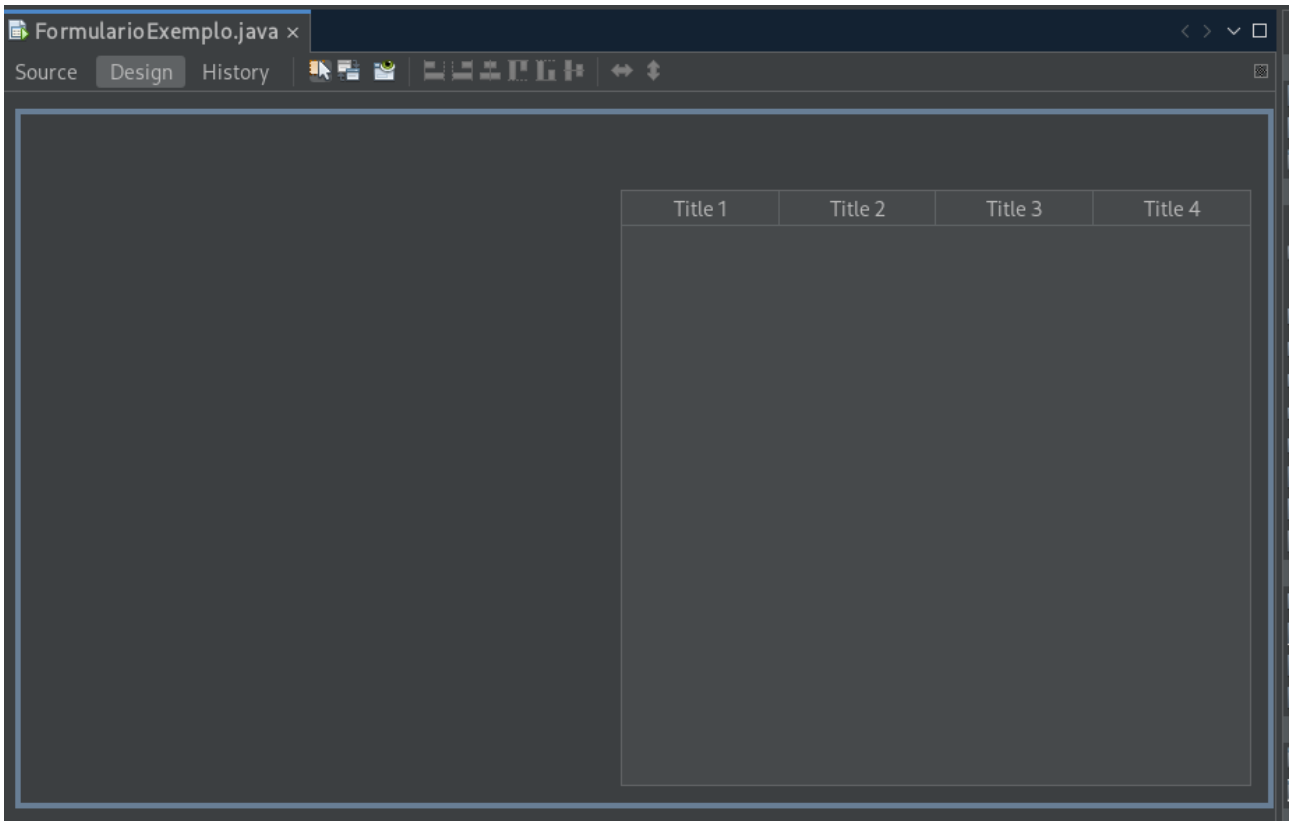
```
375     private void btnupbairroActionPerformed(java.awt.event.ActionEvent evt) {
376         ControleBairroDTO objbairrodto = new ControleBairroDTO();
377         ControleBairroDAO objbairrodao = new ControleBairroDAO();
378 
379         String idbairro;
380         String selecionado = (String) cmbupbairro.getSelectedItem();
381         String novobairro = txtupbairro.getText();
382 
383         idbairro = mapaidpornome.get(selecionado);
384 
385         objbairrodto.setId(Integer.parseInt(idbairro));
386         objbairrodto.setNome(novobairro);
387 
388         objbairrodao.atualizarBairro(objbairrodto);
389         listarBairro();
390 
391         txtcadbairro.setText("");
392         lblstatus.setText("Alterado com sucesso");
393     }
```

3.2.4 – DELETE

```
395     private void btndelbairroActionPerformed(java.awt.event.ActionEvent evt) {
396         ControleBairroDTO objbairrodto = new ControleBairroDTO();
397         ControleBairroDAO objbairrodao = new ControleBairroDAO();
398 
399         String idbairro;
400         String selecionado = (String) cmbdelbairro.getSelectedItem();
401 
402         idbairro = mapaidpornome.get(selecionado);
403 
404         objbairrodto.setId(Integer.parseInt(idbairro));
405         objbairrodao.excluirBairro(objbairrodto);
406         listarBairro();
407 
408         lblstatus.setText("Excluído com sucesso");
409     }
```

3.2.5 – Tabela

O Jtable é um componen te visual udtilizado para vizualizar dados em forma de grid, com cabeçalho, colunas e linhas e é um dos componentes mais complexos do pacote Swing. O Jtable é componente MVC (Model, View, Controller), deste modo, o seu trabalho é dividido em três partes.



- Model: Cuida dos dados da tabela, ou seja, é quem controla e distribui os mesmos. É implementado pela interface TableModel.
- View: Cuida da apresentação da tabela. É implementado pela interface CellRenderer. A apresentação é dada célula a célula da tabela.
- Controller: Controla a apresentação dos dados na camada view.

Embora, as tabelas sejam componentes extremamente complexos, a JTable encapsula grande parte desta complexidade. Existem algumas maneiras de trabalhar com o componente JTable, neste tutorial será apresentado como construir uma tabela simples populada através de um array bidimensional de objetos, em seguida, uma tabela populada através de dados presentes em um banco de dados utilizando o modelo default da JTable DefaultTableModel e para finalizar uma demonstração de como criar um modelo próprio de tabela sobrescrevendo os métodos da classe AbstractTableModel.

3.2.6 – Default Table Model

```
485     private List listarBairro(){
486         ControleBairroDAO objbairrodao = new ControleBairroDAO();
487 
488         List<ControleBairroDTO> lista = objbairrodao.pesquisarBairro();
489 
490         try {
491             DefaultTableModel model = (DefaultTableModel) tabbairro.getModel();
492             model.setRowCount(0);
493 
494             mapaidpornome = new HashMap<>();
495             mapaidpornome.clear();
496 
497             cmbupbairro.removeAllItems();
498             cmbdelbairro.removeAllItems();
499 
500             for (ControleBairroDTO bairro : lista) {
501                 model.addRow(new Object[]{
502                     bairro.getId(),
503                     bairro.getNome()
504                 });
505 
506                 mapaidpornome.put(bairro.getNome(),String.valueOf(bairro.getId())
507 
508                 cmbupbairro.addItem(bairro.getNome());
509                 cmbdelbairro.addItem(bairro.getNome());
510             }
511 
512         } catch (Exception e) {
513             JOptionPane.showMessageDialog(null, "listar Valores VIEW: " + e);
514         }
515         return lista;
516     }
517 }
```