

[< VOLTAR](#)

Projeto de GUI conectada a Bases de Dados

Nesta aula você aprenderá como unir dois conceitos importantes: Bancos de dados e interfaces gráficas (GUI), para criar aplicações ainda mais robustas e escaláveis. Isso quer dizer que essa é uma aula prática, ou seja, implementaremos uma interface gráfica, passo a passo, e usaremos um banco de dados para armazenar as informações contidas na tela.

NESTE TÓPICO

- > Definição do que será construído aqui
- > Modelo de dados
- > Projeto de nossa GUI conectada à base de dados
- > Resumo da aula
- > Referências



Definição do que será construído aqui



Como essa aula é essencialmente prática, pois unirá os conceitos de interfaces e banco de dados, é preciso ter um pré-conhecimento bastante sólido sobre alguns conceitos de programação orientada a objetos com Java, pois serão aplicados aqui inevitavelmente. Os principais conceitos aplicados aqui serão:

- Herança
- Polimorfismo
- Encapsulamento
- Conexão com bancos de dados
- Interfaces gráficas de usuário (GUI)
- Conceitos base da linguagem: Tipos de dados, visibilidade, coleções de dados, tratamento de exceções e abstração de dados

Portanto, para concluir essa aula você precisa destes conceitos bem fixados, ou seja, se alguma dúvida permanece, não deixe de rever a aula relacionada.

Vamos definir aqui o que construiremos de projeto. Para praticar construções de GUIs (interfaces gráficas) conectadas a bases de dados, vamos implementar um pequeno sistema acadêmico, contendo três telas, cada uma com uma função específica em relação ao banco de dados.

Modelo de dados

Para implementarmos um projeto completo, com interfaces gráficas e conectado a uma base de dados, precisamos definir como será nosso banco de dados, em relação a estrutura de tabelas.

Para este projeto, vamos criar apenas uma tabela, para facilitar. Tenha certeza que seu servidor de bancos de dados esteja operando normalmente. Recomenda-se o uso do MySQL.

Então vamos lá. Mãos à obra. Crie um schema para armazenar suas tabelas e, em seguida, crie a tabela de clientes com o seguinte script:

```
1. CREATE TABLE `cliente` (  
2.   `id` INT NOT NULL,  
3.   `nome` VARCHAR(45) NULL,  
4.   `telefone` VARCHAR(45) NULL,  
5.   `sexo` VARCHAR(45) NULL,  
6.   `renda` VARCHAR(45) NULL,  
7.   PRIMARY KEY (`id`));
```



Feito isso, você estará pronto para prosseguir com a implementação do projeto em Java.

Projeto de nossa GUI conectada à base de dados

Uma vez implementada a tabela acima, vamos criar um novo projeto no NetBeans, chamado "ProjetoBD". Depois disso, adicione a biblioteca do MySQL ao projeto, conforme visto em:

./videos/370386473.mp4



Depois de adicionar a biblioteca, você está pronto para programar. Como estamos criando um projeto bem definido, orientado a objetos e aplicando boas práticas de programação, vamos criar uma classe para cada responsabilidade no sistema, dividindo-as em pacotes específicos para aquele fim. Vamos criar um projeto baseado no projeto apresentado por Teruel (2016).

Crie, agora, o primeiro pacote que conterá as classes responsáveis por conexões e manipulações na base de dados: **br.uninove.poo.persistencia**

É muito importante que você leia os códigos que estão nessa aula, pois os comentários contidos nos códigos ajudam a entender o que cada linha representa.

Feito isso, vamos implementar nossa primeira classe, a que representa um cliente no banco de dados (e em nossa aplicação). A classe, cliente, então, poderá ficar assim:



```
1. package br.uninove.poo.persistencia;
2.
3. public class Cliente {
4.
5.     private int id;
6.     private String nome;
7.     private String telefone;
8.     private String sexo;
9.     private double renda;
10.
11.     public Cliente() {
12.     }
13.
14.     public Cliente(int id, String nome, String telefone, String sexo, double renda)
15.     {
16.         this.id = id;
17.         this.nome = nome;
18.         this.telefone = telefone;
19.         this.sexo = sexo;
20.         this.renda = renda;
21.     }
22.
23.     public int getId() {
24.         return id;
25.     }
26.
27.     public void setId(int id) {
28.         this.id = id;
29.     }
30.
31.     public String getNome() {
32.         return nome;
33.     }
34.
35.     public void setNome(String nome) {
36.         this.nome = nome;
37.     }
38.
39.     public String getTelefone() {
40.         return telefone;
41.     }
42.
43.     public void setTelefone(String telefone) {
44.         this.telefone = telefone;
45.     }
46.
47.     public String getSexo() {
48.         return sexo;
49.     }
50.
51.     public void setSexo(String sexo) {
52.         this.sexo = sexo;
53.     }
54.
55.     public double getRenda() {
56.         return renda;
57.     }
58.
59.     public void setRenda(double renda) {
60.         this.renda = renda;
61.     }
62. }
```



Depois de implementada a classe “Cliente”, vamos criar a classe que manipula um cliente no banco de dados. Essa será uma classe que conhece objetos da classe cliente e é capaz de gravar, editar, buscar etc., na base de

dados. Estamos falando, agora, da classe ClienteDAO (DAO = Data Access Object - Objeto de Acesso a Dados). A classe ClienteDAO pode ser vista abaixo:



```
1. package br.uninove.poo.persistencia;
2.
3. import java.sql.*;
4. import java.util.ArrayList;
5. import br.uninove.poo.persistencia.Cliente;
6.
7. public class ClienteDAO {
8.
9.     //Não esqueça de atualizar para os dados de SEU BANCO DE DADOS:
10.    String url = "jdbc:mysql://localhost:3306/uninove";
11.    String usuario = "uninove";
12.    String senha = "Senha123";
13.
14.    Connection con;
15.    PreparedStatement st;
16.    private ResultSet rs;
17.
18.    public int conectar() {
19.        try {
20.            Class.forName("com.mysql.jdbc.Driver");
21.            con = DriverManager.getConnection(url, usuario, senha);
22.            return 1;
23.        } catch (ClassNotFoundException ex) {
24.            return 2;
25.        } catch (SQLException ex) {
26.            return 3;
27.        }
28.    }
29.
30.    public int desconectar() {
31.        try {
32.            con.close();
33.            return 1;
34.        } catch (SQLException ex) {
35.
36.            return 0;
37.        }
38.    }
39.
40.    public int salvar(Cliente cli) {
41.        try {
42.            st = con.prepareStatement("insert into cliente "
43.                + "(id, nome, telefone, sexo, renda) "
44.                + "values (?, ?, ?, ?, ?)");
45.            st.setInt(1, cli.getId());
46.            st.setString(2, cli.getNome());
47.            st.setString(3, cli.getTelefone());
48.            st.setString(4, cli.getSexo());
49.            st.setDouble(5, cli.getRenda());
50.            int retorno = st.executeUpdate();
51.            return retorno;
52.        } catch (SQLException ex) {
53.            int c = ex.getErrorCode();
54.            if (c == 1062) {
55.                return 2;
56.            } else {
57.                return 3;
58.            }
59.        }
60.    }
61.
62.    //Método para consultar apenas um cliente cujo id foi informado
63.    public Cliente buscar(int id) {
64.        try {
65.            st = con.prepareStatement("select * from cliente where id = ? ");
66.            st.setInt(1, id);
67.            rs = st.executeQuery();
68.            if (rs.next()) {
69.                Cliente cli = new Cliente();
70.                cli.setId(rs.getInt("id"));
71.                cli.setNome(rs.getString("nome"));
```



```

72.         cli.setTelefone(rs.getString("telefone"));
73.         cli.setSexo(rs.getString("sexo"));
74.         cli.setRenda(rs.getDouble("renda"));
75.         return cli;
76.     } else {
77.         return null;
78.     }
79. } catch (SQLException ex) {
80.     return null;
81. }
82. }
83.
84. //Método para buscar todos os registros cadastrados
85. public ArrayList buscarTudo() {
86.     ArrayList<Cliente> listaClientes = new ArrayList<>();
87.     try {
88.         st = con.prepareStatement("select * from cliente");
89.         rs = st.executeQuery();
90.         while (rs.next()) {
91.             Cliente cli = new Cliente();
92.             cli.setId(rs.getInt("id"));
93.             cli.setNome(rs.getString("nome"));
94.             cli.setTelefone(rs.getString("telefone"));
95.             cli.setSexo(rs.getString("sexo"));
96.             cli.setRenda(rs.getDouble("renda"));
97.             listaClientes.add(cli);
98.         }
99.         return listaClientes;
100.    } catch (SQLException ex) {
101.        return null;
102.    }
103. }
104.
105. //Método para excluir um registro cujo id foi informado
106. public int excluir(int id) {
107.     try {
108.         st = con.prepareStatement("delete from cliente where id = ? ");
109.         st.setInt(1, id);
110.         int r = st.executeUpdate();
111.         return r;
112.     } catch (SQLException ex) {
113.         return 0;
114.     }
115. }
116.
117. // Método para alterar o registro cujo id foi informado
118. public int salvarAlteracao(Cliente cli) {
119.     try {
120.         st = con.prepareStatement("update cliente set nome=?, telefone=?, sexo
121. =?, renda=? where id=?");
122.         st.setString(1, cli.getNome());
123.         st.setString(2, cli.getTelefone());
124.         st.setString(3, cli.getSexo());
125.         st.setDouble(4, cli.getRenda());
126.         st.setInt(5, cli.getId());
127.         int r = st.executeUpdate();
128.         return r;
129.     } catch (SQLException ex) {
130.         return 0;
131.     }
132. }

```



Finalmente, vamos implementar nossas telas. Para este exemplo, vamos mostrar a implementação da tela de cadastro de clientes, que está no pacote **br.uninove.poo.view**.

Lembre-se que, se você alterar o nome do pacote ou criar um com outro nome, é preciso ajustar na classe.

Nossa tela de cadastro de cliente pode ser vista abaixo:





```

1. package br.uninove.poo.view;
2.
3. import java.awt.event.ActionEvent;
4. import java.awt.event.ActionListener;
5. import java.text.DecimalFormat;
6. import java.text.ParseException;
7. import javax.swing.*;
8. import javax.swing.text.DefaultFormatterFactory;
9. import javax.swing.text.MaskFormatter;
10. import javax.swing.text.NumberFormatter;
11. import br.uninove.poo.persistencia.Cliente;
12. import br.uninove.poo.persistencia.ClienteDAO;
13.
14. public class TelaCliente extends JFrame implements ActionListener {
15.
16.     JButton btnSalvar;
17.     JLabel lblId, lblNome, lblTelefone, lblSexo, lblRenda;
18.     JTextField txtNome;
19.     JFormattedTextField txtTelefone, txtId, txtRenda;
20.     MaskFormatter mskTelefone, mskId;
21.     ButtonGroup gruSexo;
22.     JRadioButton rdoMasculino, rdoFeminino;
23.
24.     public TelaCliente() {
25.         setTitle("Cadastro");
26.         setDefaultCloseOperation(EXIT_ON_CLOSE);
27.         setLayout(null);
28.         setBounds(400, 200, 600, 280);
29.         lblId = new JLabel("ID:");
30.         lblId.setBounds(10, 10, 50, 30);
31.         add(lblId);
32.         txtId = new JFormattedTextField();
33.         txtId.setFormatterFactory(new DefaultFormatterFactory(new NumberFormatter(ne
w DecimalFormat("#"))));
34.         txtId.setBounds(10, 40, 150, 30);
35.         txtId.requestFocus();
36.         add(txtId);
37.         lblNome = new JLabel("Nome:");
38.         lblNome.setBounds(170, 10, 50, 30);
39.         add(lblNome);
40.
41.         txtNome = new JTextField(40);
42.         txtNome.setBounds(170, 40, 400, 30);
43.         add(txtNome);
44.
45.         lblTelefone = new JLabel("Telefone:");
46.         lblTelefone.setBounds(10, 80, 150, 30);
47.         add(lblTelefone);
48.         try {
49.             mskTelefone = new MaskFormatter("(##)####-####");
50.             mskTelefone.setPlaceholderCharacter('_');
51.         } catch (ParseException ex) {
52.             System.out.println(ex.getMessage());
53.         }
54.         txtTelefone = new JFormattedTextField(mskTelefone);
55.         txtTelefone.setBounds(10, 110, 100, 30);
56.         add(txtTelefone);
57.
58.         lblSexo = new JLabel("Sexo:");
59.         lblSexo.setBounds(170, 80, 150, 30);
60.         add(lblSexo);
61.
62.         rdoMasculino = new JRadioButton("Masculino");
63.         rdoMasculino.setBounds(170, 110, 100, 30);
64.         rdoFeminino = new JRadioButton("Feminino");
65.         rdoFeminino.setBounds(300, 110, 100, 30);
66.         gruSexo = new ButtonGroup();
67.         gruSexo.add(rdoMasculino);
68.         gruSexo.add(rdoFeminino);
69.         add(rdoMasculino);
70.         add(rdoFeminino);

```



```

71.
72.         lblRenda = new JLabel("Renda:");
73.         lblRenda.setBounds(450, 80, 150, 30);
74.         add(lblRenda);
75.
76.         txtRenda = new JFormattedTextField();
77.         txtRenda.setFormatterFactory(new DefaultFormatterFactory(new NumberFormatter
(new DecimalFormat("#0.00"))));
78.         txtRenda.setBounds(450, 110, 100, 30);
79.         txtRenda.setToolTipText("Utilize vírgula para separar as casas decimais");
80.         add(txtRenda);
81.
82.         btnSalvar = new JButton("Salvar");
83.         btnSalvar.setBounds(10, 170, 150, 30);
84.         btnSalvar.addActionListener(this);
85.         add(btnSalvar);
86.     }
87.
88.     public static void main(String args[]) {
89.         TelaCliente tela = new TelaCliente();
90.         tela.setVisible(true);
91.     }
92.
93.     @Override
94.     public void actionPerformed(ActionEvent e) {
95.         String mensagem;
96.         if (e.getSource() == btnSalvar) {
97.             int id;
98.             double renda;
99.             String nome, telefone, sexo;
100.            id = Integer.parseInt(txtId.getText());
101.            nome = txtNome.getText();
102.            telefone = txtTelefone.getText();
103.            if (rdoMasculino.isSelected()) {
104.                sexo = rdoMasculino.getText();
105.            } else if (rdoFeminino.isSelected()) {
106.                sexo = rdoFeminino.getText();
107.            } else {
108.                sexo = "";
109.            }
110.            renda = Double.parseDouble(txtRenda.getText().replace(",", "."));
111.            Cliente cli = new Cliente(id, nome, telefone, sexo, renda);
112.            ClienteDAO dao = new ClienteDAO();
113.            int r = dao.conectar();
114.            if (r == 2) {
115.                mensagem = "Driver de conexão não foi encontrado";
116.            } else if (r == 3) {
117.                mensagem = "Os dados de conexão com o banco de dados estão incorreto
s";
118.            } else {
119.                int x = dao.salvar(cli);
120.                if (x == 1) {
121.                    mensagem = "Dados gravados com sucesso";
122.                    limparCampos();
123.                } else if (x == 2) {
124.                    mensagem = "Você está tentando cadastrar um ID que já existe";
125.                } else {
126.                    mensagem = "Dados gravados com sucesso";
127.                }
128.                dao.desconectar();
129.            }
130.            JOptionPane.showMessageDialog(null, mensagem);
131.        }
132.    }
133.
134.    private void limparCampos() {
135.        txtId.setText("");
136.        txtNome.setText("");
137.        txtTelefone.setText("");

```



```

138.         gruSexo.clearSelection();
139.         txtRenda.setText("");
140.         txtId.requestFocus();
141.     }
142. }

```

Pronto. Nossa aplicação com interface gráfica e acesso ao banco de dados está pronta. Vamos executá-la! A tela principal (de cadastro de clientes) ficará com essa aparência:

Execução do projeto - GUI conectada a uma base de dados

Depois de executar, essa interface gráfica já está pronta para receber informações e salva. Na tela mostrada acima, os dados já estão preenchidos. Depois de apertarmos o botão salvar, um **insert** é executado e as informações são inseridas no banco. Veja, por exemplo como está nossa tabela criada depois de executar o cadastro acima:



```

[mysql> select * from cliente;
+----+-----+-----+-----+-----+
| id | nome           | telefone | sexo   | renda |
+----+-----+-----+-----+-----+
| 1  | Josefino da Silva Sauro | (11)9090-9090 | Masculino | 12000.45 |
+----+-----+-----+-----+-----+
1 row in set (0,00 sec)

```

Resultado do select na tabela

Pronto. Agora você está pronto para unir dois conceitos: Interfaces gráficas e bancos de dados. Note que na classe ClienteDAO, há diversos métodos de consulta e atualização prontos. Para praticar, não deixe de criar interfaces gráficas que sejam capazes de listar, atualizar, apagar etc., clientes deste modelo.

Resumo da aula

Nesta aula você aprendeu a unir dois conceitos importantíssimos em Java: Interfaces gráficas (GUI) e acesso a bancos de dados. É muito importante você não esquecer que:

- A organização do código é muito importante para você não se perder e para facilitar futuras alterações e manutenções no código;
- As interfaces gráficas devem ficar, por boas práticas de programação, em um local (pacote) específico para este fim;
- As classes de negócio podem representar as entidades que você possui na base de dados;
- Cada classe de negócio associada a uma entidade na base de dados deve possuir sua própria classe associada de acesso a base. Neste exemplo usamos a nomenclatura ?DAO?, ou seja, objetos de acesso aos dados. É uma nomenclatura muito comum usada no mercado;
- Unir os conceitos de GUI e bancos de dados não é difícil, tudo que você precisa é organizar o código para que cada coisa fique em seu lugar.

Pronto. Chegamos ao final de um importante conceito de orientação a objetos em Java. Neste ponto você já está pronto para criar aplicações robustas e corporativas e, até mesmo, concorrer as melhores vagas no mercado. Para isso, não deixe de estudar e aplicar bastante todos os conceitos apresentados [até] aqui. Bons estudos e boa programação.

Não deixe de realizar os exercícios abaixo, que são ótimos para fixação de importantes conceitos sobre orientação a objetos em Java.



Quiz

Exercício Final

Projeto de GUI conectada a Bases de Dados

INICIAR ➤

Referências

Teruel, E. C., 2015, Programação Orientada a Objetos com Java - sem mistérios - 1ª Ed., Editora Uninove

Deitei P. e Deitel H., 2010, Java : Como programar, 8ª Edição, Pearson Pretice Hall

Schildt, H., 2015, Schildt, Java para iniciantes : crie, compile e execute programas Java rapidamente, Bookman



Avalie este tópico



ANTERIOR

Construção de GUI em Java



Índice

Biblioteca

(<https://www.uninove.br/conhec-a->

a-

uninove/biblioteca/sobre-

a-

biblioteca/apresentacao/)

Portal Uninove

(<http://www.uninove.br>)

Mapa do Site

Ajuda?

(<https://ava.un>

idCurso=)

Projeto de um Apple

® Todos os direitos reservados

