USJT – 2024/1 – Programação de Soluções Computacionais

Professor: Evandro Catelani Ferraz

**Aula: 11 – Interface gráfica e persistência de dados**

**1 Introdução a interfaces gráficas e persistência de dados**

O pacote **javax.swing** e seus sub pacotes possuem classes úteis para a criação de interfaces gráficas para o usuário (GUI: *Graphical User Interface*). Até então, fizemos uso de uma delas a JOptionPane. Neste material estudaremos a criação de interfaces gráficas utilizando diferentes recursos, indo além da simples exibição de caixas de diálogo. Alguns IDEs facilitam bastante o desenvolvimento, principalmente quando a quantidade de código a ser gerada é extensa. Vamos utilizar de agora em diante o Netbeans.

**2 Desenvolvimento**

**2.1 (Criando interfaces gráficas com o NetBeans: Um sistema acadêmico)** Nesta seção passaremos a utilizar o plugin de arrastar e soltar componentes do NetBeans para implementar um sistema acadêmico. O sistema irá permitir operações básicas de acesso a uma base de dados contendo dados de alunos e cursos de uma instituição de ensino.

**2.1.1 (Criando o projeto)** Comece criando um novo projeto/programa com nome de acordo com o contexto.

**2.1.2 (Criando a tela de login)** A primeira tela do sistema permitirá que o usuário insira seus dados de acesso. Para criá-la, clique com o direito no pacote principal da aplicação e escolha **New >> JFrame Form**. Seu nome será **LoginTela**.

**2.1.3 (Campo para login)** O usuário irá digitar seu login em um JTextField. Note que há uma paleta de componentes à direita. Arraste um componente do tipo **Text Field** para a tela. Faça os seguintes ajustes:

- **Largura**: 270

- **Altura**: 54

- Clique com o direito, escolha **Edit Text** e apague o texto que ele exibe por padrão. - Clique com o direito, escolha **Change Variable Name** e digite **loginTextField**. - Mantenha-o selecionado e veja suas propriedades na parte inferior direita da tela. Encontre a propriedade **border**. Escolha **Titled Border** e digite **Digite seu login** no campo **Title**.

**2.1.4 (Campo para senha)** Arraste e solte um componente do tipo **Password Field** logo **abaixo** do **loginTextField**. Faça os seguintes ajustes:

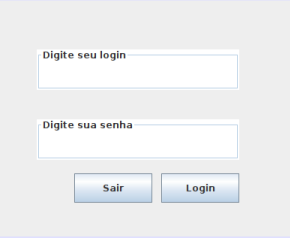
- Posição, largura e altura iguais aos do **loginTextField**.

- Clique com o direito, escolha **Edit Text** e apague o texto que ele exibe por padrão. - Clique com o direito, escolha **Change Variable Name** e digite **senhaPasswordField**.

- Mantenha-o selecionado e veja suas propriedades na parte inferior direita da tela. Encontre a propriedade **border**. Escolha **Titled Border** e digite **Digite sua senha** no campo **Title**.

**2.1.5 (Botões para sair e para fazer login)** Arraste e solte dois componentes **Button** e faça ajustes para que o resultado seja parecido com o que exibe a Figura 2.1.5.1.

Figura 2.1.5.1



- Clique com o direito em cada botão, escolha **Change Variable Name** e altere seus nomes para **sairButton** e **loginButton.**

**2.1.6 (Tratando o evento “clique” dos botões)** Há uma tarefa específica a ser executada quando cada um dos botões for clicado.

- No caso do botão **sairButton**, apenas desejamos encerrar a aplicação. Para isso, basta clicar duas vezes sobre ele e completar o corpo do método que irá aparecer, como mostra a Listagem 2.1.6.1. Note que o registro do observador já foi feito automaticamente.

Listagem 2.1.6.1

|  |
| --- |
| private void sairButtonActionPerformed(java.awt.event.ActionEvent evt) {  this.dispose();  } |

- Para o botão **loginButton**, iremos especificar uma lógica simples, que ainda não acessa a base:

- Primeiro, pegamos o login que o usuário digitou.

- Depois, pegamos a senha. Note que a senha é entregue como um vetor de char. É preciso converter para String.

- Depois, verificamos se ambos são iguais a **admin** (isso vai ser alterado quando passarmos a usar uma base de dados). Em caso positivo, o sistema mostra uma mensagem de boas-vindas. Caso contrário, mostra uma mensagem de usuário inválido. Veja a Listagem 2.1.6.2.

Listagem 2.1.6.2

|  |
| --- |
| private void loginButtonActionPerformed(java.awt.event.ActionEvent evt) {  //pega o login e a senha do usuário  String login = loginTextField.getText();  String senha = new String(senhaPasswordField.getPassword());    //verifica se o usuário é válido  if (login.equals("admin") && senha.equals("admin"))  JOptionPane.showMessageDialog (null, "Bem vindo!");  else  JOptionPane.showMessageDialog(null, "Usuário inválido");  } |

**2.1.7 (Ajustes finos no código gerado pelo NetBeans)** Para demonstrar a possibilidade de personalização no código gerado pelo NetBeans, vamos **centralizar** a tela e adicionar um **título** à moldura. Não podemos editar o código gerado por ele. Podemos, porém, adicionar código ao construtor.

- Encontre o construtor e adicione as linhas destacadas na Listagem 2.1.7.1.

Listagem 2.1.7.1

|  |
| --- |
| public LoginTela() {  super ("Sistema Acadêmico");  initComponents();  this.setLocationRelativeTo(null);  } |

**2.2 (Implementando a funcionalidade de login)** Os dados de usuários do sistema serão armazenados em uma base relacional gerenciada pelo MySQL. Como sabemos, a fim de obter esses dados, a aplicação Java precisa estabelecer uma conexão com o MySQL Server, o que pode ser feito utilizando a API JDBC.

- Começamos criando um database para o sistema. No Workbench, uma vez conectado com o MySQL Server, use

**CREATE DATABASE nome\_do\_seu\_db;**

**USE nome\_do\_seu\_db;**

para criar o novo database e informar ao MySQL Server que as próximas instruções deverão ter impacto sobre ele.

- A seguir, crie uma tabela para armazenar os dados de usuários com:

**CREATE TABLE tb\_usuario (id INT PRIMARY KEY AUTO\_INCREMENT, login VARCHAR(200), senha VARCHAR(200));**

- Faça a inserção de um usuário com:

**INSERT INTO tb\_usuario (login, senha) VALUES (‘admin’, ‘admin’);**

- Segundo o princípio conhecido como **alta coesão**, cada classe que criamos deve ter um único propósito, uma única razão de ser. Sendo assim, criaremos uma classe cuja única responsabilidade será a de gerenciar conexões com o banco. Veja a Listagem 2.2.1.

Listagem 2.2.1

|  |
| --- |
| import java.sql.Connection;  import java.sql.DriverManager;  public class ConexaoBD {  /\* Digite aqui as informações referentes ao SEU banco de dados.  Os valores “seu\_banco”, “seu\_usuario” e “sua\_senha” devem ser substituídos, respectivamente, pelo nome do seu banco, seu usuário e sua senha do MySQL Workbench \*/      private static String host = "localhost";      private static String porta = "3306";      private static String db = "seu\_banco";      private static String usuario = "seu\_usuario";      private static String senha = "sua\_senha";        public static Connection obtemConexao (){          try{              Connection c = DriverManager.getConnection("jdbc:mysql://" + host + ":" + porta + "/" + db + "?useTimezone=true&serverTimezone=UTC", usuario,senha);              return c;          }          catch (SQLException e){              return null;          }      }  } |

- Lembre-se de abrir o arquivo **pom.xml** e especificar que o driver do MySQL deve ser baixado pelo Maven. O ajuste a ser feito é exibido na Listagem 2.2.2.

Listagem 2.2.2

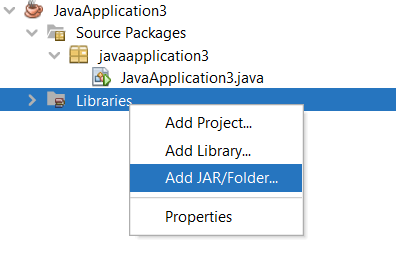
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>br.com.bossini</groupId>  <artifactId>pessoal\_sistema\_academico\_com\_netbeans\_para\_montar\_pdf</ artifactId>  <version>1.0-SNAPSHOT</version>  <packaging>jar</packaging>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding> <maven.compiler.source>14</maven.compiler.source>  <maven.compiler.target>14</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.20</version>  </dependency>  </dependencies>  </project> |

- Caso o projeto desenvolvido **não seja do tipo Maven**, devemos incluir o driver do MySQL manualmente. O arquivo de driver deve ser incluído na pasta Libraries (ou Bibliotecas, em português) do projeto. O arquivo de driver se encontra disponível no material desta aula, no link:

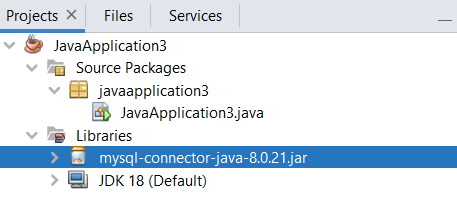
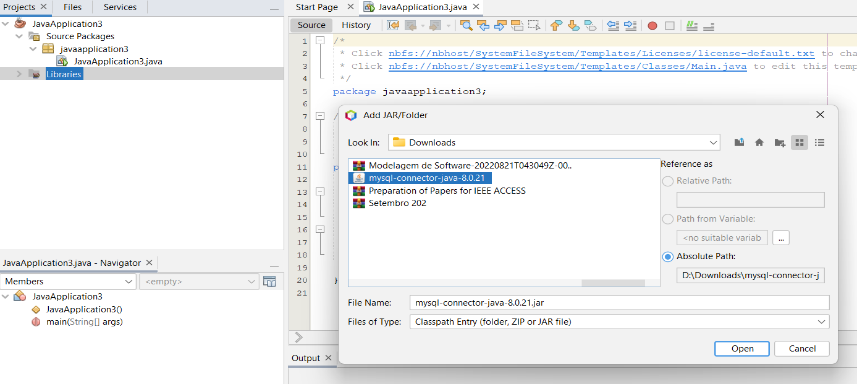
<https://drive.google.com/drive/folders/1Ev7LeNRI-yZk_NqhQYlpWeVJPb-weDBm>

Após o download, clique com o botão direito na pasta Libraries do projeto e escolha a opção Add JAR/Folder. Selecione então o arquivo do driver mysql-connector, como é mostrado nas figuras abaixo.

Passo 1 – Clique com o botão direito na pasta Libraries e escolha a opção Add JAR/Folder.



Passo 2 – Selecione o arquivo de driver.



- Um sistema desenvolvido com linguagem que tem suporte à orientação a objetos é uma representação simplificada do mundo real (lembra do **mini-mundo**?). Assim, vamos criar uma classe para representar o que é um usuário do sistema. No momento, usuários possuem apenas duas coisas de interesse: login e senha. Veja a definição da classe que descreve o que é um usuário na Listagem 2.2.3.

Listagem 2.2.3

|  |
| --- |
| public class Usuario {    private String login;  private String senha;  public Usuario(String login, String senha) {  this.login = login;  this.senha = senha;  }  public String getLogin() {  return login;  }  public void setLogin(String login) {  this.login = login;  }    public String getSenha() {  return senha;  }  public void setSenha(String senha) {  this.senha = senha;  }  } |

- A seguir, vamos criar uma classe que será responsável por todas as operações de persistência de dados da aplicação. Assim, crie a classe da Listagem 2.2.4. No momento, o único método que ela possui se encarrega de verificar se um determinado usuário existe ou não na base de dados.

**Nota:** DAO é um acrônimo para **Data Access Object**, ou seja, Objeto de Acesso aos Dados. Trata-se de um dos padrões de desenvolvimento de software mais antigos. Uma classe DAO tem a finalidade de encapsular código de acesso a bases de dados. É comum que um projeto possua muitas classes DAO, cada qual apropriada para a manipulação de diferentes tipos de objetos.

Listagem 2.2.4

|  |
| --- |
| public class DAO {  public boolean existe (Usuario usuario) throws Exception{  String sql = "SELECT \* FROM tb\_usuario WHERE login = ? AND senha = ?";  try (Connection conn = ConexaoBD.obtemConexao();  PreparedStatement ps = conn.prepareStatement(sql)){  ps.setString(1, usuario.getLogin());  ps.setString(2, usuario.getSenha());  try (ResultSet rs = ps.executeQuery()){  return rs.next();  }  }  }  } |

- O método acionado quando o botão de login é clicado será cliente do método existe. Assim ele passa a validar os dados do usuário em função do que existe realmente na base. Veja a Listagem 2.2.5.

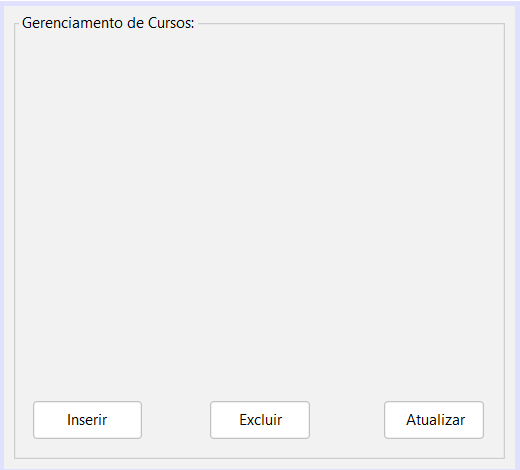
Listagem 2.2.5

|  |
| --- |
| private void loginButtonActionPerformed(java.awt.event.ActionEvent evt) {  //pega o login do usuário  String login = loginTextField.getText();  //pega a senha do usuário como char[] e converte para String  String senha = new String (senhaPasswordField.getPassword());    try{  //verifica se o usuário é válido  Usuario usuario = new Usuario (login, senha);  DAO dao = new DAO();  if (dao.existe(usuario)){  JOptionPane.showMessageDialog (null, "Bem vindo, " + usuario.getLogin() + "!");  }  else{  JOptionPane.showMessageDialog(null, "Usuário inválido"); }  }  catch (Exception e){  JOptionPane.showMessageDialog (null, "Problemas técnicos. Tente novamente mais tarde");  }  } |

**2.3 Implementando a Interface Gráfica para o CRUD.**

Crie um novo **JFrame** com o nome **CRUDTela**. Coloque um elemento **Panel** e modifique sua propriedade **border** incluindo um título com borda. A tela deve possuir, inicialmente, a aparência mostrada na Listagem 2.3.1:

Listagem 2.3.1



Dessa forma, devemos incluir 3 botões. Edite o texto de cada botão para Inserir, Excluir e Atualizar, respectivamente. Em seguida, altere a nome de variável dos 3 botões para **btnInserirCurso**, **btnExcluirCurso** e **btnAtualizarCurso**, respectivamente. Lembre-se que para trocar o nome de variável de um elemento na tela, devemos clicar com o direito e selecionar a opção “Change Variable Name”.

Queremos também definir um título e centralizar nossa nova tela. Fazemos isso implementando o código no construtor do JFrame, como mostrado na Listagem 2.3.2.

Listagem 2.3.2

|  |
| --- |
| public CRUDTela() {  super("Gerenciador de Cursos");  initComponents();  this.setLocationRelativeTo(null);  } |

Devemos também vincular nossas classes JFrame criadas até o momento, **LoginTela** e **CRUDTela**. Neste contexto, queremos que a tela de CRUD apareça quando o botão login for clicado e um usuário válido for encontrado. Dessa forma, devemos incluir o código que vincula as duas telas no evento do botão de login de **LoginTela**. A Listagem 2.3.3 mostra essa implementação:

Listagem 2.3.3

|  |
| --- |
| private void loginButtonActionPerformed(java.awt.event.ActionEvent evt) {  //pega o login do usuário  String login = loginTextField.getText();  //pega a senha do usuário como char[] e converte para String  String senha = new String (senhaPasswordField.getPassword());    try{  //verifica se o usuário é válido  Usuario usuario = new Usuario (login, senha);  DAO dao = new DAO();  if (dao.existe(usuario)){  JOptionPane.showMessageDialog (null, "Bem vindo, " + usuario.getLogin() + "!");  CRUDTela ct = new CRUDTela();  ct.setVisible(true);  this.dispose();  }  else{  JOptionPane.showMessageDialog(null, "Usuário inválido"); }  }  catch (Exception e){  JOptionPane.showMessageDialog (null, "Problemas técnicos. Tente novamente mais tarde");  }  } |

- Execute sua aplicação e teste o que programados até o momento.

- Evidentemente, os cursos serão armazenados na base de dados. Por isso, acesse a base com o Workbench e crie uma tabela apropriada para o armazenamento de cursos. Utilize o código:

**CREATE TABLE tb\_curso (id INT PRIMARY KEY AUTO\_INCREMENT,**

**nome VARCHAR (200) NOT NULL, tipo VARCHAR (200) NOT NULL);**

- A seguir, faça a inserção de um curso para que tenhamos um primeiro dado de teste. Para isso, utilize o código:

**INSERT INTO tb\_curso (nome, tipo) VALUES ('Ciência da Computação',**

**'Bacharelado');**

- Devemos também criar uma classe **Curso** para representar nossos cursos de forma apropriada. A Listagem 2.3.4 mostra essa implementação:

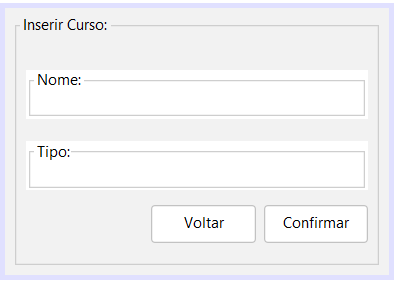
Listagem 2.3.4

|  |
| --- |
| public class Curso {  private int id;  private String nome;  private String tipo;  public Curso(int id, String nome, String tipo) {  this.id = id;  this.nome = nome;  this.tipo = tipo;  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getNome() {  return nome;  }  public void setNome(String nome) {  this.nome = nome;  }  public String getTipo() {  return tipo;  }  public void setTipo(String tipo) {  this.tipo = tipo;  }  } |

**2.4 Implementando o botão de Inserção.**

Crie um novo **JFrame** com o nome **InserirTela**. Coloque um elemento **Panel** e modifique sua propriedade **border** incluindo um título com borda. A tela deve possuir, inicialmente, a aparência mostrada na Listagem 2.4.1:

Listagem 2.4.1



Dessa forma, devemos criar um **Panel** e incluir 2 campos de texto e 2 botões. Edite o texto de cada botão para Confirmar e Voltar, respectivamente. Em seguida, altere a nome de variável dos 2 botões para **btnConfirmarInserir** e **btnVoltarInserir**. Para os campos de texto, apague o texto inicial em cada um deles e mude os nomes de variável para **txtNomeInserir** e **txtTipoInserir**. Lembre-se que para trocar o nome de variável de um elemento na tela, devemos clicar com o direito e selecionar a opção “Change Variable Name”.

Adicione também bordas com título (na propriedade border > Titled Border) para o Panel e para os 2 campos de texto criados.

Queremos também definir um título e centralizar nossa nova tela. Fazemos isso implementando o código no construtor do JFrame, como mostrado na Listagem 2.4.2.

Listagem 2.4.2

|  |
| --- |
| public InserirTela() {  super("Inserir Cursos");  initComponents();  this.setLocationRelativeTo(null);  } |

Em seguida, iremos programar primeiro a funcionalidade do botão “**Voltar”**. Clicando duas vezes no botão, temos acesso ao método que implementa o evento de clique deste botão, ou seja, o que acontecerá quando o botão for clicado pelo usuário. Queremos que ao clicar no botão “Voltar” o usuário retorne à tela de CRUD. Devemos então escrever o código destacado em amarelo na Listagem 2.4.3.

Listagem 2.4.3

|  |
| --- |
| private void btnVoltarInserirActionPerformed(java.awt.event.ActionEvent evt) {  CRUDTela t = new CRUDTela();  t.setVisible(true);  this.dispose();  } |

No JFrame **CRUDTela**, queremos que o JFrame **InserirTela** seja chamado quando o usuário clicar no botão “Inserir”. Devemos então incluir o código destacado em amarelo na Listagem 2.4.4, que deve ser escrito no método de evento do botão “Inserir”. Lembre-se que para acessar o método de evento de um botão, basta clicar duas vezes nesse botão que o Netbeans nos direcionará automaticamente ao método, criando o método automaticamente caso ele já não exista.

Listagem 2.4.4

|  |
| --- |
| private void btnInserirCursoActionPerformed(java.awt.event.ActionEvent evt) {  InserirTela t = new InserirTela();  t.setVisible(true);  this.dispose();  } |

Para implementar a função de **Inserção** na base de dados, devemos escrever esse método na classe que controle as instruções SQL (querys) que acessam a base de dados da nossa aplicação. Já temos uma classe criada com essa finalidade, a classe **DAO** (Data Acess Object). Dessa forma, incluiremos o método **inserir()** na classe DAO, como mostrado na Listagem 2.4.5.

Listagem 2.4.5

|  |
| --- |
| public void inserir (Curso curso) throws Exception{  String sql = "INSERT INTO tb\_curso (nome,tipo) values (?,?)";  try (Connection conn = ConexaoBD.obtemConexao();  PreparedStatement ps = conn.prepareStatement(sql)){  ps.setString(1, curso.getNome());  ps.setString(2, curso.getTipo());  ps.execute();  }  } |

Após a criação do método **inserir()**, podemos retornar ao JFrame **InserirTela** e programar o evento do botão “Confirmar”. Clicamos duas vezes neste botão e em seu método escrevemos o código apresentado na Listagem 2.4.6. Com isso, concluímos a implementação da inserção de um curso na nossa base de dados.

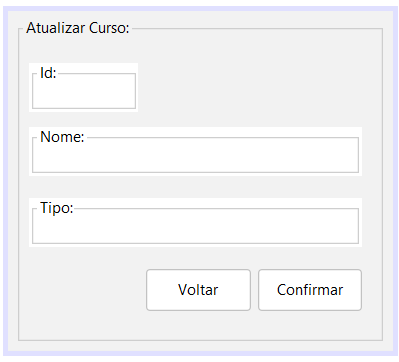
Listagem 2.4.6

|  |
| --- |
| private void btnConfirmarInserirActionPerformed(java.awt.event.ActionEvent evt) {  String nome = txtNomeInserir.getText();  String tipo = txtTipoInserir.getText();    try{  Curso curso = new Curso(0,nome,tipo);  DAO dao = new DAO();  dao.inserir(curso);  JOptionPane.showMessageDialog(null,"Curso inserido!!");  }catch(Exception e){  JOptionPane.showMessageDialog(null,"Não foi possível inserir.");  }  } |

**2.5 Implementando o botão de Atualização.**

Crie um novo **JFrame** com o nome **AtualizarTela**. Coloque um elemento **Panel** e modifique sua propriedade **border** incluindo um título com borda. A tela deve possuir, inicialmente, a aparência mostrada na Listagem 2.5.1:

Listagem 2.5.1



Dessa forma, devemos criar um **Panel** e incluir 3 campos de texto e 2 botões. Edite o texto de cada botão para Confirmar e Voltar, respectivamente. Em seguida, altere a nome de variável dos 2 botões para **btnConfirmarAtualizar** e **btnVoltarAtualizar**. Para os campos de texto, apague o texto inicial em cada um deles e mude os nomes de variável para **txtIdAtualizar, txtNomeAtualizar** e **txtTipoAtualizar**. Lembre-se que para trocar o nome de variável de um elemento na tela, devemos clicar com o direito e selecionar a opção “Change Variable Name”.

Adicione também bordas com título (na propriedade border > Titled Border) para o Panel e para os 3 campos de texto criados.

Queremos também definir um título e centralizar nossa nova tela. Fazemos isso implementando o código no construtor do JFrame, como mostrado na Listagem 2.5.2.

Listagem 2.5.2

|  |
| --- |
| public AtualizarTela() {  super("Atualizar Cursos:");  initComponents();  this.setLocationRelativeTo(null);  } |

Em seguida, iremos programar primeiro a funcionalidade do botão “**Voltar”**. Clicando duas vezes no botão, temos acesso ao método que implementa o evento de clique deste botão, ou seja, o que acontecerá quando o botão for clicado pelo usuário. Queremos que ao clicar no botão “Voltar” o usuário retorne à tela de CRUD. Devemos então escrever o código destacado em amarelo na Listagem 2.5.3.

Listagem 2.5.3

|  |
| --- |
| private void btnVoltarAtualizarActionPerformed(java.awt.event.ActionEvent evt) {  CRUDTela t = new CRUDTela();  t.setVisible(true);  this.dispose();  } |

No JFrame **CRUDTela**, queremos que o JFrame **AtualizarTela** seja chamado quando o usuário clicar no botão “Atualizar”. Devemos então incluir o código destacado em amarelo na Listagem 2.5.4, que deve ser escrito no método de evento do botão “Atualizar”. Lembre-se que para acessar o método de evento de um botão, basta clicar duas vezes nesse botão que o Netbeans nos direcionará automaticamente ao método, criando o método automaticamente caso ele já não exista.

Listagem 2.5.4

|  |
| --- |
| private void btnAtualizarCursoActionPerformed(java.awt.event.ActionEvent evt) {  AtualizarTela t = new AtualizarTela();  t.setVisible(true);  this.dispose();  } |

Para implementar a função de **Atualização** na base de dados, devemos escrever esse método na classe que controla nossas instruções SQL (querys), a classe **DAO**. Incluiremos o método **atualizar()** na classe DAO, como mostrado na Listagem 2.5.5:

Listagem 2.5.4

|  |
| --- |
| public void atualizar(Curso curso) throws Exception{  String sql = "UPDATE tb\_curso SET nome = ?, tipo = ? WHERE id = ?";  try (Connection conn = ConexaoBD.obtemConexao();  PreparedStatement ps = conn.prepareStatement(sql)){  ps.setString(1, curso.getNome());  ps.setString(2, curso.getTipo());  ps.setInt(3, curso.getId());  ps.execute();  }  } |

Após a criação do método **atualizar()**, podemos retornar ao JFrame **AtualizarTela** e programar o evento do botão “Confirmar”. Clicamos duas vezes neste botão e em seu método escrevemos o código apresentado na Listagem 2.5.5. Com isso, concluímos a implementação da atualização de um curso na nossa base de dados.

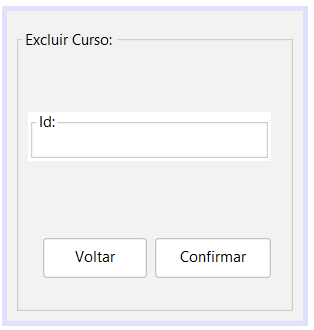
Listagem 2.5.5

|  |
| --- |
| private void btnConfirmarAtualizarActionPerformed(java.awt.event.ActionEvent evt) {    int id = Integer.parseInt(txtIdAtualizar.getText());  String nome = txtNomeAtualizar.getText();  String tipo = txtTipoAtualizar.getText();    try{  Curso curso = new Curso(id,nome,tipo);  DAO dao = new DAO();  dao.atualizar(curso);  JOptionPane.showMessageDialog(null,"Curso atualizado!!");    }catch(Exception e){  JOptionPane.showMessageDialog(null,"Não foi possível atualizar.");  }  } |

**2.6 Implementando o botão de Exclusão.**

Crie um novo **JFrame** com o nome **ExcluirTela**. Coloque um elemento **Panel** e modifique sua propriedade **border** incluindo um título com borda. A tela deve possuir, inicialmente, a aparência mostrada na Listagem 2.6.1:

Listagem 2.6.1



Dessa forma, devemos criar um **Panel** e incluir 1 campo de texto e 2 botões. Edite o texto de cada botão para Confirmar e Voltar, respectivamente. Em seguida, altere a nome de variável dos 2 botões para **btnConfirmarExcluir** e **btnVoltarExcluir**. Para o campo de texto, apague o texto inicial e mude o nome de variável para **txtIdExcluir**. Lembre-se que para trocar o nome de variável de um elemento na tela, devemos clicar com o direito e selecionar a opção “Change Variable Name”.

Adicione também bordas com título (na propriedade border > Titled Border) para o Panel e para o campo de texto criado.

Queremos também definir um título e centralizar nossa nova tela. Fazemos isso implementando o código no construtor do JFrame, como mostrado na Listagem 2.6.2.

Listagem 2.6.2

|  |
| --- |
| public ExcluirTela() {  super("Excluir Cursos:");  initComponents();  this.setLocationRelativeTo(null);  } |

Em seguida, iremos programar primeiro a funcionalidade do botão “**Voltar”**. Clicando duas vezes no botão, temos acesso ao método que implementa o evento de clique deste botão, ou seja, o que acontecerá quando o botão for clicado pelo usuário. Queremos que ao clicar no botão “Voltar” o usuário retorne à tela de CRUD. Devemos então escrever o código destacado em amarelo na Listagem 2.6.3.

Listagem 2.6.3

|  |
| --- |
| private void btnVoltarExcluirActionPerformed(java.awt.event.ActionEvent evt) {  CRUDTela t = new CRUDTela();  t.setVisible(true);  this.dispose();  } |

No JFrame **CRUDTela**, queremos que o JFrame **ExcluirTela** seja chamado quando o usuário clicar no botão “Excluir”. Devemos então incluir o código destacado em amarelo na Listagem 2.6.3, que deve ser escrito no método de evento do botão “Excluir”. Lembre-se que para acessar o método de evento de um botão, basta clicar duas vezes nesse botão que o Netbeans nos direcionará automaticamente ao método, criando o método automaticamente caso ele já não exista.

Listagem 2.6.4

|  |
| --- |
| private void btnExcluirCursoActionPerformed(java.awt.event.ActionEvent evt) {  ExcluirTela t = new ExcluirTela();  t.setVisible(true);  this.dispose();  } |

Para implementar a função de **Exclusão** na base de dados, devemos escrever esse método na classe que controla nossas instruções SQL (querys), a classe **DAO**. Incluiremos o método **excluir()** na classe DAO, como mostrado na Listagem 2.6.5:

Listagem 2.6.5

|  |
| --- |
| public void excluir(Curso curso) throws Exception{  String sql = "DELETE FROM tb\_curso WHERE id=?";  try (Connection conn = ConexaoBD.obtemConexao();  PreparedStatement ps = conn.prepareStatement(sql)){  ps.setInt(1, curso.getId());  ps.execute();  }  } |

Após a criação do método **excluir()**, podemos retornar ao JFrame **ExcluirTela** e programar o evento do botão “Confirmar”. Clicamos duas vezes neste botão e em seu método escrevemos o código apresentado na Listagem 2.6.6. Com isso, concluímos a implementação da exclusão de um curso na nossa base de dados.

Listagem 2.6.6

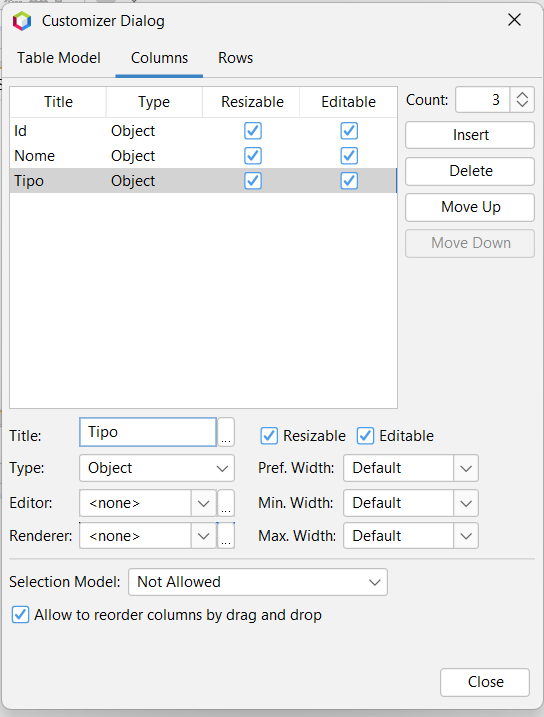
|  |
| --- |
| private void btnConfirmarExcluirActionPerformed(java.awt.event.ActionEvent evt) {    int id = Integer.parseInt(txtIdExcluir.getText());    try{  Curso curso = new Curso(id, "", "");  DAO dao = new DAO();  dao.excluir(curso);  JOptionPane.showMessageDialog(null,"Curso Excluído!!");    }catch(Exception e){  JOptionPane.showMessageDialog(null,"Não foi possível excluir.");  }  } |

**2.7 Implementando uma função de Consulta.**

Por fim, implementaremos uma funcionalidade de **Consulta** em nosso projeto. Para isso, incluiremos no JFrame **CRUDTela** um componente chamado **JTable** que, como o nome sugere, consiste em uma tabela. Utilizaremos essa tabela para apresentar os dados de resultado (Result Grid) retornados pela query de consulta (SELECT).

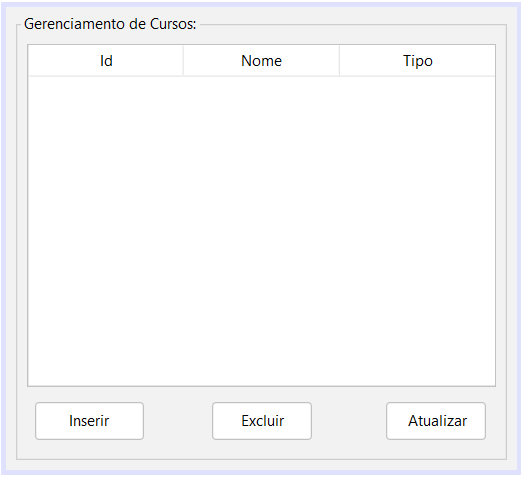
Trocamos o nome de variável da tabela para **tableCursos**. Devemos também incluir nomenclaturas para cada coluna dessa tabela, faremos isso clicando com o botão direito na tabela e selecionando a opção “Table Contents”. Em seguida, selecionamos a aba “Columns” e incluímos 3 colunas, para os atributos Id, Nome e Tipo, respectivamente. A Listagem 2.7.1 mostra um exemplo da aba “Columns”, onde foram adicionados os atributos desejados.

Listagem 2.7.1



Na Listagem 2.7.2, temos a tela de CRUD atualizada com a inclusão da tabela:

Listagem 2.7.2



Nesta funcionalidade de Consulta, desejamos que a tabela receba os dados de todos os Cursos cadastrados em nosso banco. Dessa forma, sempre que a **CRUDTela** for construída, queremos que a tabela **tableCursos** nos mostre o Id, Nome e Tipo de todos os cursos cadastrados em nossa base de dados.

Começamos implementando essa funcionalidade com a adição do método **consultaCursos**(), que deve ser escrito no código fonte (aba Source) da CRUDTela**.** A Listagem 2.7.3 mostra o código necessário para a implementação desse método.

Listagem 2.7.3

|  |
| --- |
| private void consultaCursos() throws Exception{  // Cria um objeto para representar a tabela que incluímos na tela  DefaultTableModel model = (DefaultTableModel)tableCursos.getModel();  model.setRowCount(0);    String sql = "SELECT \* FROM tb\_curso";  try (Connection conn = ConexaoBD.obtemConexao();  PreparedStatement ps = conn.prepareStatement(sql)){  ResultSet rs = ps.executeQuery();    // Enquanto existir uma linha no result grid, representado por rs, essa linha será  // adicionada na tabela, representada por model.  while(rs.next()){  model.addRow(new String[]{rs.getString(1), rs.getString(2), rs.getString(3)});  }  }  } |

Certifique-se de importar as classes DefaultTableModel, Connection PreparedStatement e ResultSet. Em seguida, queremos que o método consultaTabela() seja chamado toda vez que a tela CRUDTela for criada. Devemos então adicionar este método ao método construtor da CRUDTela, como mostra a Listagem 2.7.4.

Listagem 2.7.4

|  |
| --- |
| public CRUDTela() {  super("Gerenciamento de Cursos");  initComponents();  this.setLocationRelativeTo(null);    try {  consultaCursos();  } catch (Exception ex) {  JOptionPane.showMessageDialog(null,"Não foi possí­vel buscar os dados do banco.");  }  } |