



Desenvolvimento de Sistemas

Listagem de dados

Em conhecimentos anteriores, falou-se a respeito da manipulação de dados e da importância dela. Sabe-se que, em muitas situações, é necessário conectar a sua aplicação Java a um sistema de banco de dados, seja para consultar alguma informação, seja para armazenar um dado dentro desse banco de dados.

Pensando nisso, surge uma questão: quais são as maneiras possíveis de se fazer buscas e listagem de dados recuperados do banco de dados? Como realizar consultas para um relatório, por exemplo?

Antes de responder a essas questões, que tal fazer uma revisão?



Conectando com banco de dados

Quando você precisa criar uma conexão de algum banco de dados com uma aplicação Java, utiliza um conceito chamado “ponte”, no qual se usa o *driver* JDBC (Java Database Connectivity) para realizar a conexão entre o banco de dados e a sua aplicação.

Confira um breve resumo dos passos para a conexão com banco de dados com JDBC:

1. Crie o projeto que pode ser do tipo **Maven** ou **Ant**.
2. Inclua a biblioteca MySQL Connector J.
 - a. Se o projeto for **Maven**, basta editar o arquivo **pom.xml** incluindo uma notação XML da dependência, informando dados da biblioteca. Esses dados podem ser obtidos acessando o *site* “Maven Central Repository Search” e buscando “mysql conector java” nele.
 - b. Se o projeto for **Ant**, você deve buscar e baixar a biblioteca MySQL Connector J do *site* de *downloads* do MySQL. Depois disso, inclua o arquivo **.jar** como dependência do projeto (clitando com o botão direito do *mouse* em **Libraries** e optando por **Add JAR/Folder** no projeto Java.
3. Ative a classe de *driver* com o código
`Class.forName("com.mysql.cj.jdbc.Driver");`.
4. Crie a conexão (objeto da classe **Connection**) usando o método **DriverManager.getConnection("jdbc:mysql://localhost/nome_do_banco", "usuario", "senha")**.
5. Trate as exceções **SQLException** e **ClassNotFoundException**.

Depois de utilizar a conexão de banco de dados, libere-a para que ela não fique ocupando espaço na memória. Para isso, crie um método para desconectar:

Caso você não libere/feche a conexão, o coletor de lixo do Java (*garbage collector*) fechará a conexão depois de um determinado tempo.

Um exemplo de código de conexão é o seguinte:

```
package exemploconexao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ExemploConexao {

    public static void main(String[] args) {

        ExemploConexao conexao = new ExemploConexao();
        Connection conn = conexao.conectar();
        conexao.desconectar(conn);
    }

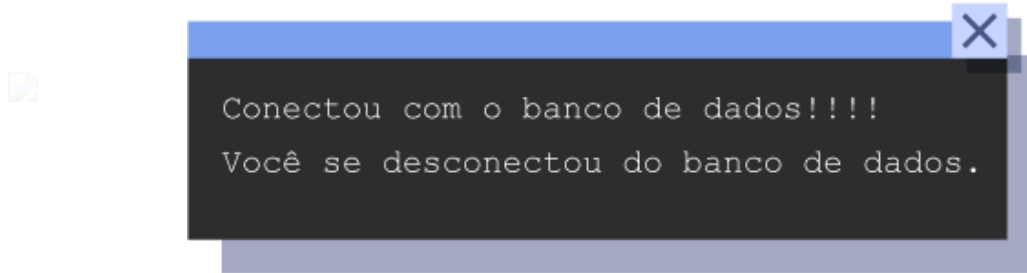
    public Connection conectar() {
        Connection conn = null;
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            conn = DriverManager.getConnection("jdbc:mysql://localhost/teste", "root", "");

            System.out.println("Conectou com o banco de dados!!!!");
        } catch (SQLException ex) {
            System.out.println("Erro: Não foi possível se conectar no banco de dados!");
        } catch (ClassNotFoundException ex) {
            System.out.println("Erro: Driver JDBC não encontrado!");
        }

        return conn;
    }

    public void desconectar(Connection conn) {
        try {
            if (conn != null && !conn.isClosed()) {
                conn.close();
                System.out.println("Você se desconectou do banco de dados.");
            }
        } catch (SQLException ex) {
            System.out.println("Não foi possível desconectar do banco de dados.");
        }
    }
}
```

Caso tudo esteja correto, você terá a seguinte mensagem no console:



Aplicação prática com uso de banco de dados utilizando o *driver* JDBC

Agora, será feita uma aplicação com uso de banco de dados. Nesse projeto, a persistência de dados (inserção, edição, exclusão e pesquisar) será realizada em uma base de dados por meio de interface gráfica.

Que tal praticar?

O primeiro passo será criar uma aplicação Java Ant, o banco de dados e a classe de conexão utilizando o *driver* JDBC.

◆ Crie uma aplicação Java Ant com o nome de “ProjetoJava”.

Depois de criar a aplicação, crie o banco de dados (ainda sem nenhuma tabela), que pode ser chamado de “ExemploJavaBD”.


```
CREATE DATABASE exemplojavabd;
```

Volte à sua aplicação Java e crie a classe de conexão ao banco de dados.

◆ Para organizar este projeto, crie um novo pacote para a classe de conexão.

Com o botão direito do *mouse*, clique em **Source Packages > New > Java Package** e dê ao pacote o nome de **conexao**. Agora, dentro do pacote **conexao**, crie a classe: com o botão direito do *mouse*, clique no pacote **conexão** e em seguida **New > Java Class** e dê à classe o nome de **Conexao**. Lembre-se de que, por convenção, nomes de pacote devem estar com letras minúsculas e nomes de classe devem começar com a primeira letra em maiúscula. Exemplo:

conexao = nome do pacote

 **Conexao** = nome da classe

Na classe de conexão, crie um método que estabelecerá a conexão do projeto com o banco de dados e ele retornará a instância dessa conexão para poder ser utilizada em qualquer outra classe ou formulário dentro do projeto.

O código dessa classe é bem semelhante ao exemplo de conexão com banco de dados exposto anteriormente neste conteúdo.

Para começar o seu bloco de código, crie um método **Public** que retornará um **Connection**. O **Connection** é uma classe que tem dentro da linguagem Java e é responsável por estabelecer e criar uma instância dessa conexão com o banco de dados. Chame esse método de **getConexao()**.

```
public Connection getConexao() {  
}
```

O código estará apontando um erro, que é a falta da importação da classe Java responsável, a **java.sql.Connection**. Caso não apareça a importação para você, crie-a manualmente, abaixo do nome do pacote do projeto:

```
import java.sql.Connection;
```

Não se preocupe se estiver mostrando ainda o erro “*missing return statement*”. Isso será resolvido no decorrer da classe.

Dentro do método, crie uma cláusula *try-catch*, que, dentro do bloco do *try*, tente estabelecer a conexão; e, quanto ao *catch*, caso ocorra algum erro na hora de conectar, este vá para o *catch* e resulte em mensagem de erro.

```
public Connection getConexao() {  
    try {  
  
    }catch (Exception e) {  
  
    }  
}
```

Nessa parte do projeto, você tentará estabelecer a conexão com o banco de dados. Para isso, você precisa do *driver* JDBC (aconselha-se usar o arquivo em JAR) para realizar essa conexão.

Adicionando o *driver* JDBC ao seu projeto

Esse processo é bem simples: copie o *drive* de conexão JDBC baixado anteriormente (o nome do arquivo será semelhante a **mysql-connector-java-versao.jar**), até a pasta na qual se encontra o seu projeto, que, por padrão, é “NetBeansProjects”. Procure a pasta do seu projeto e crie nela uma pasta **driver** e cole o arquivo copiado. O driver está na pasta **driver**, que foi criada dentro do projeto, e agora será adicionado ao projeto.

No projeto, clique em **Libraries** e depois em **Add JAR/Folder**. Navegue até a pasta na qual está o seu projeto e, dentro da pasta **driver**, adicione o arquivo ao seu projeto. Note que o arquivo JAR foi adicionado ao projeto:

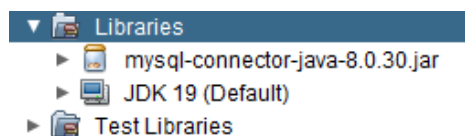


Figura 1 – Trecho correspondente à parte de *libraries* do projeto
Fonte: NetBeans (2022)

Dentro do *try*, é preciso retornar uma conexão, válida ou não, com o banco de dados. Para isso, deve-se utilizar o método apresentado a seguir, que receberá três parâmetros: a URL de conexão, o usuário e a senha do MySQL. Depois de finalizado o método, é preciso usar o **return conn** para que a aplicação retorne ao método:

```
public Connection getConexao() {  
    try {  
        Connection conn = DriverManager.getConnection(  
            "jdbc:mysql://localhost/ exemplojavabd ", // linha de conexao  
            "root", // usuario do mysql  
            "" // senha do mysql  
        );  
        return conn;  
    } catch (Exception e) {  
    }  
}
```

Deve-se importar a classe **DriverManager**, que é: **import java.sql.DriverManager;**.

O *try* tentará realizar a conexão com o banco de dados, mas também é preciso imaginar que essa conexão pode não se estabelecer devido a erro de usuário e senha, erro no nome do banco, entre outros erros. Então, isso pode ser tratado com o *catch*. Dentro do *catch*, coloque uma mensagem que retorne o erro para o usuário da seguinte maneira:

```
System.out.println("Erro ao conectar: " + e.getMessage());  
return null;
```

A variável “**e**” armazenará as informações do erro que foi gerado e você usará o **getMessage** para buscar esse erro dentro da variável.

Devido a esse erro, não haverá nenhum retorno, por isso, use o **null** para que o método retorne algo. Nesse caso, o método retornará a conexão ou o nulo.

Sua classe de conexão ficará da seguinte maneira:


```
package conexao;

import java.sql.Connection;
import java.sql.DriverManager;

public class Conexao {

    public Connection getConexao() {

        try {
            Connection conn = DriverManager.getConnection(
                "jdbc:mysql://localhost/ exemplojavabd ", // linha de conexao
                "root", // usuario do mysql
                ""// senha do mysql
            );
            return conn;

        } catch (Exception e) {
            System.out.println("Erro ao conectar: " + e.getMessage());
            return null;
        }

    }

}
```

Teste agora se o método de conexão deu certo.

No arquivo principal, que é o arquivo gerado com o nome do projeto (nesse caso, é **projetojava.java**), teste para verificar se a comunicação está funcionando.

Para isso, você pode criar uma variável da conexão com o nome de “**c**” e depois chamar o método de conexão **getConexao**.

```
Conexao c = new Conexao();
c.getConexao();
```

Como você estará chamando um método de outra classe, será necessário realizar a sua importação, que será **import conexao.Conexao**;

Construa o seu projeto e depois o execute. Caso não ocorra erro, a sua aplicação está funcionando. Tente trocar o nome do banco de dados e usuário ou senha para que você possa verificar os erros que serão mostrados no console.

Criando a tabela “empresas” no seu banco de dados

Crie agora uma tabela “empresas”, dentro do seu banco de dados “exemplojavabd”. Nela, haverá três colunas: uma coluna do tipo inteiro, contendo o nome de ID, que será a sua chave primária e também será autoincrementada; uma coluna do tipo *varchar*, contendo 50 posições e o nome de “nomeempresa”; e uma coluna do tipo *varchar*, contendo 50 posições e o nome de “areaatuacao”. O código MySQL da tabela será este:

```
CREATE TABLE `empresa` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `nomeempresa` varchar(50) NOT NULL,  
  `areaatuacao` varchar(50) NOT NULL,  
  PRIMARY KEY (id)  
);
```

Criando uma classe para mapear a tabela dentro do projeto

Voltando à aplicação Java, crie agora mais dois pacotes dentro de **Source Packages**.

O primeiro pacote será chamado de **beans** e o segundo pacote será chamado de **dao**, que significa *data access object*, ou seja, “objetos de acesso aos dados”. No pacote “beans” ficarão as entidades do sistema, como a classe Empresa e em “dao” teremos classes responsáveis pela conexão e manipulação do banco de dados, como a classe **EmpresaDAO**, que contará com métodos para inserir, pesquisar, editar e apagar dados de Empresa.

Neste momento, no pacote **beans**, crie uma classe Java com nome **Empresa** que contará com um atributo para cada coluna que consta na tabela. Todos os atributos serão privados e isso significa que eles só poderão ser acessados dentro da própria classe. Para que outras classes e outros métodos tenham acesso a eles, utilize os métodos **GET** e **SET** em cada atributo, ficando a sua classe da seguinte maneira:

```
package beans;

public class Empresa {
    private int id;
    private String nomeempresa;
    private String areaatuacao;

    public int getId() {
        return id;
    }

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

    public String getNomeempresa() {
        return nomeempresa;
    }

    public void setNomeempresa(String nomeempresa) {
        this.nomeempresa = nomeempresa;
    }

    public String getAreaatuacao() {
        return areaatuacao;
    }

    public void setAreaatuacao(String areaatuacao) {
        this.areaatuacao = areaatuacao;
    }
}
```

Criando o método “Inserir” para cadastrar dados na tabela

Agora, crie a classe **EmpresaDAO** dentro do pacote **dao**. Essa classe intermediará a sua aplicação com o banco de dados e, por isso, precisa de um objeto da classe **Conexao**, definida anteriormente. Use a ferramenta de sugestão do NetBeans para realizar **import** dessa classe.

Sabe-se ainda que **Conexao** conta com método que retorna objeto do tipo *connection*, que será usado para operações CRUD (*create, read, update, delete*) em sua classe. Por isso, também é conveniente um atributo privado para esse objeto (é necessário **import java.sql.Connection**).

```
package dao;

import conexao.Conexao;
import java.sql.Connection;

public class EmpresaDAO {

    private Conexao conexao;
    private Connection conn;

}
```

Agora, crie o construtor da classe. Ele é executado automaticamente sempre que um novo objeto é criado. Dentro dele, você instanciará a conexão, que se dará da seguinte forma:

```
public EmpresaDAO() {
    this.conexao = new Conexao();
    this.conn = this.conexao.getConexao();
}
```

Dentro da classe **EmpresaDAO**, crie um método para inserir uma nova empresa dentro da tabela empresa. Esse método receberá um parâmetro do tipo empresa, que será **public void inserir (Empresa empresa)**. Use a ferramenta de sugestão do NetBeans para realizar **import** da classe **beans.Empresa**.

No método, crie uma variável a partir do **this.conn**, que possibilitará executar comandos SQL dentro do banco de dados. Essa variável será do tipo **PreparedStatement**, na qual se deve adicionar a importação **java.sql.PreparedStatement**. Essa variável deve estar dentro de um *try-catch*, devido ao fato de se poder ou não inserir um dado nela, e, neste caso, será chamada de variável de **stmt**, que será igual ao **this.conn.prepareStatement**.

```
public void inserir(Empresa empresa) {  
    String sql = "INSERT INTO empresa(nomeempresa, areaatuacao) VALUES "  
        + "(?, ?)";  
    try {  
        PreparedStatement stmt = this.conn.prepareStatement(sql);  
  
    } catch (Exception e) {  
        System.out.println("Erro ao inserir empresa: " + e.getMessage());  
    }  
}
```

O método **prepareStatement()** recebe um parâmetro do tipo *string*, que conterá a instrução SQL que se quer executar no banco de dados – neste caso, utilize **INSERT** para gravar dados. No código que está sendo trabalhado aqui, esse parâmetro é informado pela variável *string* chamada "sql", na qual se define a instrução SQL e que deve ser declarada acima do bloco *try*.

Agora, informe quais serão os valores dos parâmetros que você quer passar para as colunas “nomeempresa” e “areaatuacao”. Passe os parâmetros da seguinte forma:

◆ **stmt.setString (1, curso.getNomeEmpresa());**

◆ Onde você quer pegar o seu primeiro parâmetro, correspondente à coluna “nomeempresa”.

◆ **stmt.setString (2, curso.getAreaatuacao());**

◆ Preencha valor para o segundo parâmetro, correspondente à coluna “areaatuacao”.

Por fim, você deve executar essa *query* usando **stmt.execute()**.

```
public void inserir(Empresa empresa) {  
    String sql = "INSERT INTO empresa(nomeempresa, areaatuacao) VALUES "  
        + "(?, ?)";  
    try {  
        PreparedStatement stmt = this.conn.prepareStatement(sql);  
        stmt.setString(1, empresa.getNomeempresa());  
        stmt.setString(2, empresa.getAreaatuacao());  
        stmt.execute();  
  
    } catch (Exception e) {  
        System.out.println("Erro ao inserir empresa: " + e.getMessage());  
    }  
}
```

Para finalizar o método de inserir, você deve considerar que a sua *query* pode estar errada por algum motivo, então trate isso dentro do *catch*, colocando uma mensagem para o usuário e retornando o erro da seguinte forma: **System.out.println("Erro ao inserir empresa: " + e.getMessage());**.

Ao final, você terá o seguinte código para a classe **EmpresaDAO**:

```
package dao;

import beans.Empresa;
import conexao.Conexao;
import java.sql.Connection;
import java.sql.PreparedStatement;

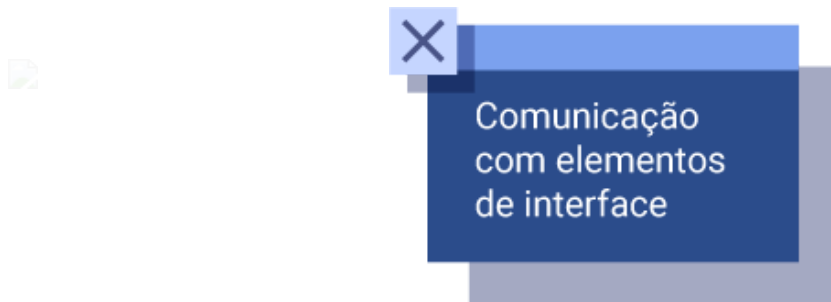
public class EmpresaDAO {

    private Conexao conexao;
    private Connection conn;

    public EmpresaDAO() {
        this.conexao = new Conexao();
        this.conn = this.conexao.getConexao();
    }

    public void inserir(Empresa empresa) {
        String sql = "INSERT INTO empresa(nomeempresa, areaatuacao) VALUES "
            + "(?, ?)";
        try {
            PreparedStatement stmt = this.conn.prepareStatement(sql);
            stmt.setString(1, empresa.getNomeempresa());
            stmt.setString(2, empresa.getAreaatuacao());
            stmt.execute();

        } catch (Exception e) {
            System.out.println("Erro ao inserir empresa: " + e.getMessage());
        }
    }
}
```



Criando o formulário para cadastrar uma empresa na tabela

Agora, crie um formulário para testar o seu método de cadastro. Para isso, crie um novo pacote com nome de **forms** e crie um **JFrame Form** chamado de "FormEmpresa".

Crie um formulário semelhante ao da seguinte imagem:

A imagem mostra uma captura de tela de uma interface web. No topo, o título "Empresa" está centralizado em uma fonte grande e preta. Abaixo do título, há dois campos de entrada. O primeiro é rotulado "Nome da Empresa:" e é um campo de texto branco com uma borda cinza. O segundo é rotulado "Área de atuação:" e é um menu suspenso com o texto "Administração" visível e uma seta para baixo no canto direito. Abaixo desses campos, há um botão retangular com o texto "Salvar" em uma fonte preta.

Figura 2 – Formulário para salvar um nome de empresa e sua área de atuação

Fonte: Senac EAD (2022)

No formulário, estão presentes *labels* (**JLabel**), um campo de texto para nome da empresa, cujo nome de variável deve ser **txtNomeEmpresa**, uma *combobox* com nome de variável **cmbArea** e com valores "Administração", "Contabilidade", "Medicina",

"Informática" e "Outros" como opções do combo. Há também no formulário um botão com nome de variável **btnSalvar**. Ajuste os elementos como desejar, porém, é importante que você altere o nome das variáveis para esses citados, para que consiga acompanhar o projeto.

Agora, você precisa criar uma ação dentro do botão de salvar, que captará as informações concedidas pelo usuário e realizará a inserção de um novo registro no banco de dados.

Com o botão direito do *mouse*, clique no botão **Salvar** e em seguida em **Events > Action > actionPerformed**. Essa ação o levará ao código do seu formulário. Tudo que você codificar nesse botão será executado quando o botão **Salvar** for pressionado.

Primeiramente, armazene na nossa classe **Empresa** os dados que foram preenchidos no formulário:

```
String nomeEmpresa = txtNomeEmpresa.getText();  
String areaatuacao = cmbArea.getSelectedItem().toString();
```

Você instanciará um objeto da classe **Empresa**:

```
Empresa empresa = new Empresa();
```

Também precisará definir as variáveis dentro do objeto **Empresa**:

```
empresa.setNomeempresa(nomeEmpresa);  
empresa.setAreaatuacao(areaatuacao);
```

Por fim, armazene esses dados dentro do banco de dados usando o método que foi criado. Para isso, você deve instanciar um objeto da classe **EmpresaDAO** e, usando esse objeto, chamar o método que foi criado, passando o objeto criado como parâmetro:

```
EmpresaDAO empresaDAO = new EmpresaDAO();  
empresaDAO.inserir(empresa);
```

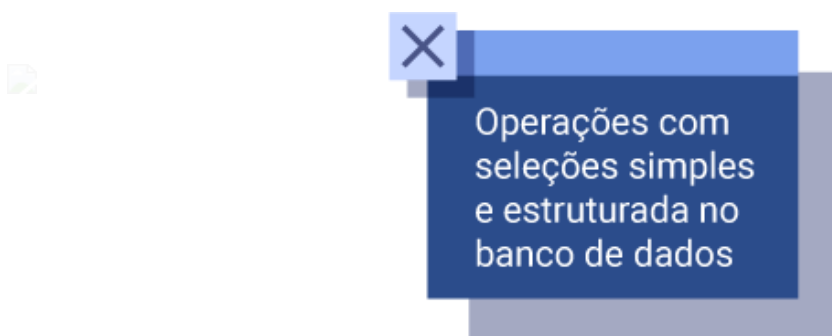
É importante limpar o nome da empresa que foi digitado, depois de ele ser salvo no banco de dados:

```
txtNomeEmpresa.setText("");
```

No final, você terá o seguinte código:

```
private void btnSalvarActionPerformed(java.awt.event.ActionEvent evt) {  
  
    String nomeEmpresa = txtNomeEmpresa.getText();  
    String areaatuacao = cmbArea.getSelectedItem().toString();  
  
    Empresa empresa = new Empresa();  
  
    empresa.setNomeempresa(nomeEmpresa);  
    empresa.setAreaatuacao(areaatuacao);  
  
    EmpresaDAO empresaDAO = new EmpresaDAO();  
    empresaDAO.inserir(empresa);  
  
    txtNomeEmpresa.setText("");  
}
```

Serão necessárias duas importações: **import beans.Empresa** e **import dao.EmpresaDAO**. Execute o seu formulário pressionando **Shift + F6**, realize algum cadastro e veja se haverá erros ou não. Caso não haja erros, verifique se a sua tabela está com o dado cadastrado.



Buscando dados e apresentando no formulário

Agora, crie um método para poder buscar a empresa por ID dentro da classe **EmpresaDAO**. A lógica será: você fornece um código (**id**) e o método retorna todas as informações (nome da empresa e área de atuação).

Abaixo do método **inserir**, dentro da classe **EmpresaDAO**, crie o método **getEmpresa**. O nome será **public Empresa getEmpresa(int id)**. Aqui não será utilizado o *void*, pois você quer um retorno, que é a empresa com seus dados. O parâmetro será um **id**, para que retornem os dados.

Para realizar esse método, segue-se o padrão semelhante ao de **inserir**, porém usando o **SELECT** agora:

```
public Empresa getEmpresa (int id){  
    String sql = "SELECT * FROM empresa WHERE id = ?";  
}
```

Query que fará a seleção dos dados na tabela “empresa” conforme o **id** que o usuário digitar.

Você usará o bloco *try-catch*, pois, como você terá uma conexão com o banco de dados, terá a chance de que seja retornado um erro, que será mostrado dentro do *catch*. Isso porque, por ordem de execução, o compilador executa tudo que está no *try* e, caso haja algum erro com a conexão ou com os campos da tabela, o *catch* será retornado.

Primeiramente, você deve usar o **PreparedStatement**, que controla e executa a instrução SQL criada, juntamente ao **ResultSet**, que conterá o conjunto de dados retornado pela sua consulta. Em seguida, você passará o parâmetro da consulta, que é o **id**.

```
public Empresa getEmpresa (int id){
    String sql = "SELECT * FROM empresa WHERE id = ?";
    try {

        PreparedStatement stmt = this.conn.prepareStatement(sql);
        stmt.setInt(1, id);
        ResultSet rs = stmt.executeQuery();

        Empresa empresa = new Empresa();

        //tratando o erro, caso ele ocorra
    } catch (Exception e) {
        System.out.println("erro: " + e.getMessage());
        return null;
    }
}
```

Você está instanciando um objeto da classe **ResultSet** e, usando o objeto **stmt**, executará a *query*. Não se esqueça de realizar a importação da classe **java.sql.ResultSet**.

```
ResultSet rs = stmt.executeQuery();
```

Agora, você deve instanciar um objeto da classe **Empresa**, o qual obterá os dados do objeto do **ResultSet (rs)** e armazenará no objeto. Será preciso mover o ponteiro do objeto atual para a próxima linha, a partir da posição atual, e, para isso, utilize o **next()**.

```
public Empresa getEmpresa (int id){  
    String sql = "SELECT * FROM empresa WHERE id = ?";  
    try {  
  
        PreparedStatement stmt = this.conn.prepareStatement(sql);  
        stmt.setInt(1, id);  
        ResultSet rs = stmt.executeQuery();  
  
        Empresa empresa = new Empresa();  
  
        rs.next();  
  
        //tratando o erro, caso ele ocorra  
    } catch (Exception e) {  
        System.out.println("erro: " + e.getMessage());  
        return null;  
    }  
}
```

Agora, atribua os dados do objeto **rs** para dentro do objeto da classe **Empresa** e também defina o que cada atributo mostrará, vinculando-os à sua respectiva coluna dentro da tabela do banco de dados. Em seguida, defina o retorno para o objeto empresa, pois seu método necessita de um retorno.

O método completo ficará da seguinte forma:

```
public Empresa getEmpresa (int id){
    String sql = "SELECT * FROM empresa WHERE id = ?";
    try {

        PreparedStatement stmt = this.conn.prepareStatement(sql);
        stmt.setInt(1, id);
        ResultSet rs = stmt.executeQuery();

        Empresa empresa = new Empresa();

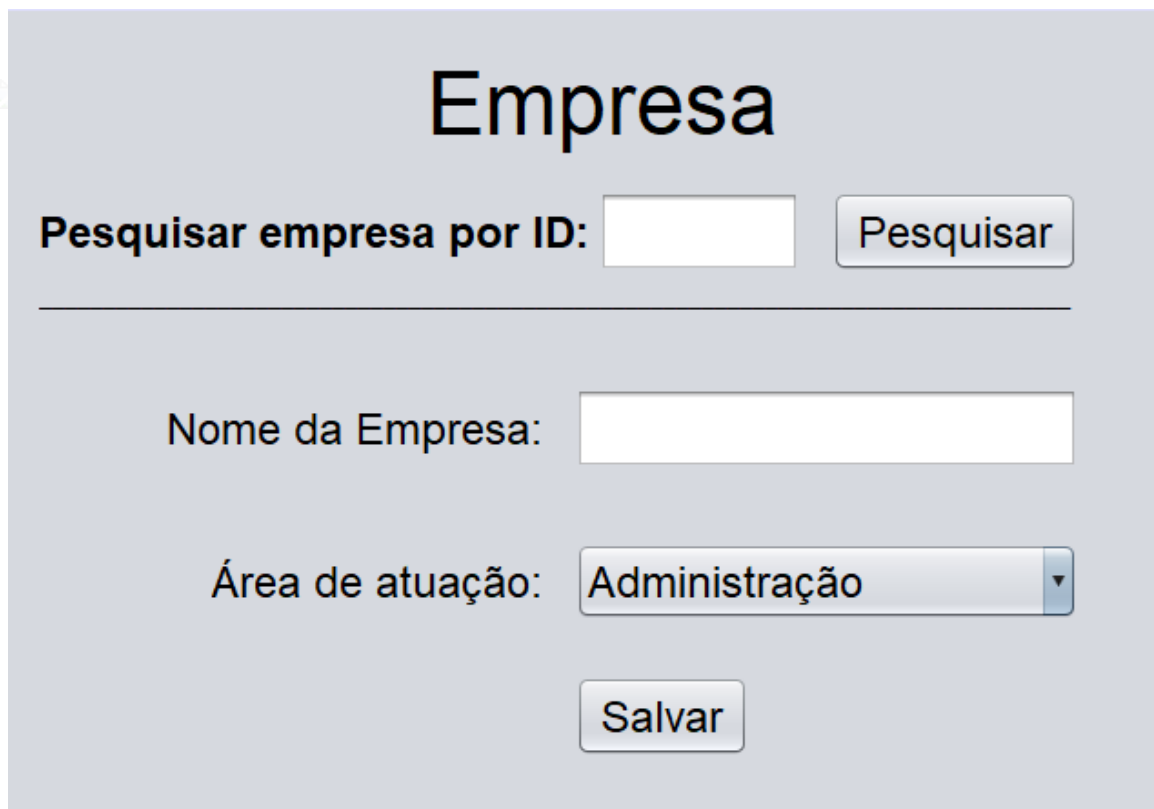
        rs.next();
        empresa.setId(id);
        empresa.setNomeempresa(rs.getString("nomeempresa"));
        empresa.setAreaatuacao(rs.getString("areaatuacao"));

        return empresa;

        //tratando o erro, caso ele ocorra
    } catch (Exception e) {
        System.out.println("erro: " + e.getMessage());
        return null;
    }
}
```

Invocando o método “getEmpresa()” dentro do formulário

Agora, volte ao seu formulário e coloque um *label*, um campo de texto (nome da variável **txtIdPesquisa**) e um botão (nome da variável **btnPesquisar**), conforme exemplo:



O formulário, intitulado "Empresa", possui um campo de busca para "Pesquisar empresa por ID:" com um botão "Pesquisar" ao lado. Abaixo, há um campo para "Nome da Empresa:" e um menu suspenso para "Área de atuação:" com a opção "Administração" selecionada. Um botão "Salvar" está localizado na base do formulário.

Figura 3 – Formulário para pesquisar e salvar uma empresa

Fonte: Senac EAD (2022)

A busca no banco de dados será realizada quando o usuário clicar no botão **Pesquisar**. Então, você deve configurar esse método usando o **actionPerformed**. Clique no botão **Pesquisar** com o botão direito do *mouse* e em seguida **Events > Action > actionPerformed**.

Primeiramente, você deve armazenar ("pegar") o **id** que o usuário digitou. Para isso, crie uma variável do tipo **int** e, usando o **GET**, armazene o que o usuário digitou. Depois, você deve instanciar um objeto da classe **EmpresaDAO**. É necessário usá-lo, pois o método **getEmpresa** está dentro da classe e você deve acessá-lo. Após isso, chame o método **getEmpresa**, que retornará um objeto do tipo **Empresa**, o qual tem como parâmetro o **id** que o usuário digitou:

```
private void btnPesquisarActionPerformed(java.awt.event.ActionEvent evt) {  
    int idPesquisa = Integer.parseInt(txtIdPesquisa.getText());  
  
    EmpresaDAO empresaDAO = new EmpresaDAO();  
    Empresa empresa = empresaDAO.getEmpresa(idPesquisa);  
}
```

Para verificar se o curso existe, use as funções **IF** e **ELSE**, nas quais você passará como critério o seguinte: no “se” (**IF**), caso o objeto **empresa** seja igual a nulo, será mostrada a mensagem “Curso não encontrado”; no “senão” (**ELSE**), a função preencherá o formulário com os dados do curso retornado.

```
private void btnPesquisarActionPerformed(java.awt.event.ActionEvent evt) {  
    int idPesquisa = Integer.parseInt(txtIdPesquisa.getText());  
  
    EmpresaDAO empresaDAO = new EmpresaDAO();  
    Empresa empresa = empresaDAO.getEmpresa(idPesquisa);  
  
    if (empresa == null) {  
        JOptionPane.showMessageDialog(this, "Curso não encontrado!");  
    }  
    else {  
        txtNomeEmpresa.setText(empresa.getNomeempresa());  
        cmbArea.setSelectedItem(empresa.getAreaatuacao());  
    }  
}
```

Execute o seu formulário pressionando **Shift + F6**. Busque algum ID existente na tabela e outro não existente para verificar se tudo está funcionando. Caso tudo esteja OK, seu formulário já está com a opção de cadastrar e a pesquisa funcionando! Vá em frente!

Editando dados por meio do formulário

Faça agora, dentro da classe **EmpresaDAO**, o método para conseguir editar os registros buscados do banco de dados.

Para que o método de editar funcione, o método de pesquisar por ID deve estar funcionando!

Vá para o arquivo **EmpresaDAO** e, depois do método de inserir, inclua o novo método, que será: **public void editar(Empresa empresa)**. Semelhantemente ao método de inserir, esse método receberá um parâmetro “Empresa empresa”.

A lógica é bem semelhante ao método de editar, porém, agora, você usará os códigos referentes ao editar/atualizar (*update*).

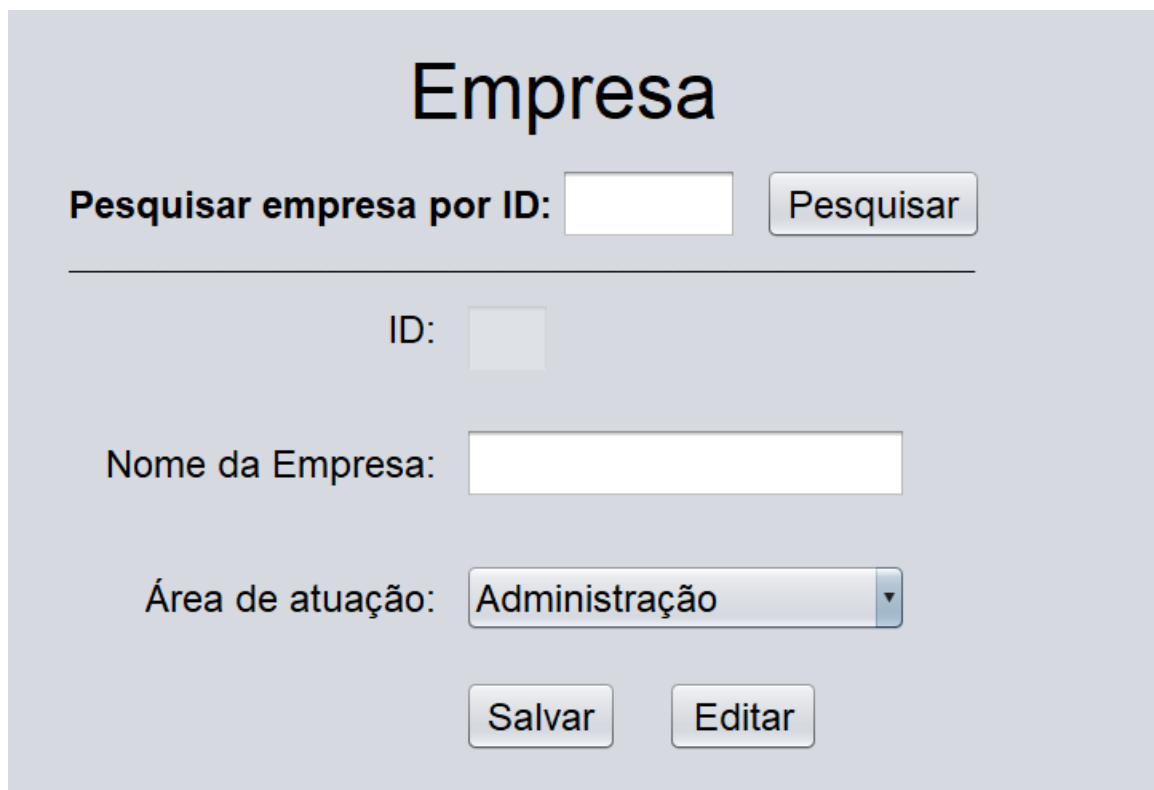
Explicando a *query*:

```
String sql = "UPDATE empresa SET nomeempresa=?, areaatuacao=? WHERE id=?";
```

A tabela empresa será atualizada, definindo-se os novos “nomeempresa” e “areaatuacao”, conforme o ID que o usuário informar. O código comentado ficará da seguinte maneira:

```
public void editar (Empresa empresa){
    //string sql com o código de update para o banco de dados
    String sql = "UPDATE empresa SET nomeempresa=?, areaatuacao=? WHERE id=?";
    try {
        //esse trecho é igual ao método inserir
        PreparedStatement stmt = conn.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
        //Setando os parâmetros
        stmt.setString(1, empresa.getNomeempresa());
        stmt.setString(2, empresa.getAreaatuacao());
        stmt.setInt(3, empresa.getId());
        //Executando a query
        stmt.execute();
        //tratando o erro, caso ele ocorra
    } catch (Exception e) {
        System.out.println("Erro ao editar empresa: " + e.getMessage());
    }
}
```

Agora, volte ao **FormEmpresa** e faça algumas alterações. Inclua um campo para mostrar o ID e também acrescente um botão para editar, ficando da seguinte maneira:



O formulário, intitulado "Empresa", contém os seguintes elementos:

- Um campo de pesquisa rotulado "Pesquisar empresa por ID:" seguido de um campo de entrada de texto e um botão "Pesquisar".
- Uma linha horizontal separadora.
- Um campo rotulado "ID:" seguido de um campo de entrada de texto.
- Um campo rotulado "Nome da Empresa:" seguido de um campo de entrada de texto.
- Um campo rotulado "Área de atuação:" seguido de uma lista suspensa com o valor "Administração" selecionado.
- Dois botões, "Salvar" e "Editar", alinhados horizontalmente na base.

Figura 4 – Formulário com as opções **Pesquisar**, **Salvar** e **Editar** empresa

Fonte: Senac EAD (2022)

- ◆ Campo de texto do ID: nome da variável é **txtID**
 - ◆ Esse campo não pode ser alterado, somente visualizado, então clica-se sobre ele com o botão direito do *mouse* em **Properties** e desmarca-se a opção **Enabled**.
- ◆ Botão **Editar**: nome da variável é **btnEditar**

Você precisará fazer um ajuste também no código do botão **Pesquisar**, pois ele só está retornando os campos de nome da empresa e de área de atuação. Você quer que ele mostre também o número do ID correspondente à pesquisa.

Clique duas vezes no botão **Pesquisar** para ir até o código e adicione a seguinte linha na parte em que o código preenche o formulário com os dados: **txtId.setText(String.valueOf(empresa.getId()));**.

O código ficará assim:

```
//preenchendo o formulário com os dados do curso retornado
txtId.setText(String.valueOf(empresa.getId())); //linha adicionada
txtNomeEmpresa.setText(empresa.getNomeempresa());
cmbArea.setSelectedItem(empresa.getAreaatuacao());
```

Agora, configure o botão **Editar** clicando nele com o botão direito do *mouse* e clicando em seguida em **Events > Action > actionPerformed**. A aplicação o levará para o local indicado para colocar o código. O processo dele será semelhante ao **btnSalvar**, o qual você pode, se se sentir à vontade, copiar e fazer nele as devidas alterações. O código ficará assim:


```
private void btnEditarActionPerformed(java.awt.event.ActionEvent evt) {
    //Pegar o código que o usuário digitou no campo de texto, converte para inteiro
    e salvar da variavel id
    int id = Integer.parseInt(txtId.getText());
    String nomeempresa = txtNomeEmpresa.getText();
    String areadeatuacao = cmbArea.getSelectedItem().toString();

    Empresa empresa = new Empresa();
    empresa.setId(id);
    empresa.setNomeempresa(nomeempresa);
    empresa.setAreaatuacao(areadeatuacao);

    EmpresaDAO empresaDAO = new EmpresaDAO();
    empresaDAO.editar(empresa);

    //limpando os campos
    txtId.setText("");
    txtNomeEmpresa.setText("");
}
```

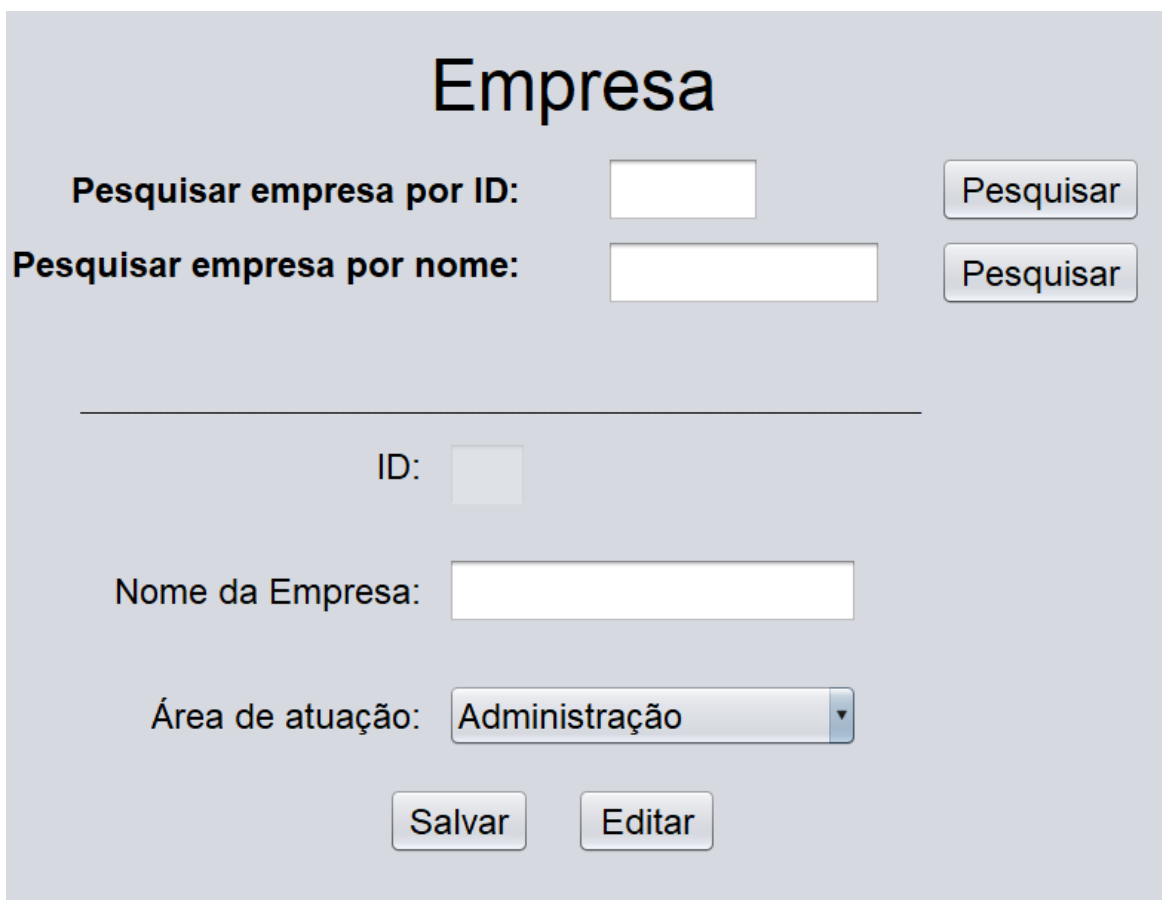
Execute o seu formulário pressionando **Shift + F6**, realize a busca de algum ID existente na tabela, faça alguma alteração no nome da empresa ou na área de atuação e clique em **Editar**. Depois, pesquise novamente esse mesmo registro para ver se foi gravada a alteração que você realizou.

Você já tem em seu formulário as opções de cadastrar, pesquisar e editar. Vá em  frente!

Criando outro método para pesquisa

Dentro do **FormEmpresa** já foi criado um método para realizar pesquisa pelo ID fornecido. Agora, você criará um método de pesquisa no qual o usuário informará o nome da empresa.

Ajuste seu formulário para que ele fique semelhante ao desta imagem:



O formulário, intitulado "Empresa", apresenta duas opções de pesquisa no topo. A primeira, "Pesquisar empresa por ID:", é acompanhada por um campo de entrada de texto e um botão "Pesquisar". A segunda, "Pesquisar empresa por nome:", também possui um campo de entrada de texto e um botão "Pesquisar". Abaixo dessas opções, há uma linha horizontal separadora. Segue-se o campo "ID:" com um pequeno campo de entrada. Em seguida, o campo "Nome da Empresa:" com um campo de entrada de texto mais largo. Abaixo disso, o campo "Área de atuação:" é seguido por um menu suspenso com "Administração" selecionado. No rodapé do formulário, há dois botões: "Salvar" e "Editar".

Figura 5 – Formulário com as opções **Pesquisar empresa por ID**, **Pesquisar empresa por nome**, **Salvar** e **Editar**

Fonte: Senac EAD (2022)

Foram adicionados um label com a frase “Pesquisar empresa por nome”, um campo de texto com nome da variável **txtIdPesquisaNome** e um botão com o texto **Pesquisar** e nome da variável **btnPesquisarEmpresa**.

Primeiramente, na classe **EmpresaDAO**, crie o método para pesquisar por nome. A lógica será bem semelhante à lógica do método de editar, porém a *query* terá mais cláusulas.

Explicando a *query*:

```
String sql = "SELECT * FROM empresa WHERE nomeempresa LIKE ?";
```

O registro da tabela “empresa” (caso haja), no qual o nome da empresa que o usuário informar seja o mesmo que está cadastrado, sendo ele maiúsculo ou minúsculo, será pesquisado e selecionado. Por exemplo, se o usuário cadastrou “SENAC” e pesquisou “senac”, o valor será retornado.

```
public Empresa getEmpresaNome (String nomeempresa){
    String sql = "SELECT * FROM empresa WHERE nomeempresa LIKE ?";
    try {
        PreparedStatement stmt = this.conn.prepareStatement(sql);

        stmt.setString(1, nomeempresa);

        ResultSet rs = stmt.executeQuery();

        Empresa empresa = new Empresa();
        rs.next();

        empresa.setId(rs.getInt("id"));
        empresa.setNomeempresa(nomeempresa);
        empresa.setNomeempresa(rs.getString("nomeempresa"));
        empresa.setAreaatuacao(rs.getString("areaatuacao"));

        return empresa;

        //tratando o erro, caso ele ocorra
    } catch (Exception e) {
        System.out.println("erro: " + e.getMessage());
        return null;
    }
}
```

Neste momento, você configurará o botão **Pesquisar**, que foi criado por último, clicando nele com o botão direito do *mouse* e em seguida em **Events > Action > actionPerformed**. A aplicação levará você para o local indicado para colocar o código. O processo dele será semelhante ao do **btnEditar**, o qual, caso você se sinta à vontade, poderá copiar e fazer nele as devidas alterações. O código ficará assim:

```
private void btnPesquisarEmpresaActionPerformed(java.awt.event.ActionEvent evt) {  
    String nomepesquisa = txtIdPesquisaNome.getText();  
  
    EmpresaDAO empresaDAO = new EmpresaDAO();  
    Empresa empresa = empresaDAO.getEmpresaNome(nomepesquisa);  
  
    if (empresa == null) {  
        JOptionPane.showMessageDialog(this, "Empresa não encontrado!");  
    }  
    else {  
        txtId.setText(String.valueOf(empresa.getId()));  
        txtNomeEmpresa.setText(empresa.getNomeempresa());  
        cmbArea.setSelectedItem(empresa.getAreaatuacao());  
    }  
}
```

Excluindo dados por meio do formulário

Crie um método para excluir um registro no banco de dados e, respectivamente, inclua o botão no formulário.

Primeiramente, na classe **EmpresaDAO**, crie o método para exclusão. Crie o novo método abaixo do método de editar. Seu nome será **public void excluir (int id)**, seguindo o mesmo padrão dos métodos **inserir** e **editar**. O parâmetro recebido será um **id**, pois será excluído algum registro por meio do seu **id**.

Explicando a *query*:

```
String sql = "DELETE FROM empresa WHERE id = ?";
```

O registro será deletado do registro da tabela “empresa”, conforme o **id** que o usuário informar.

O código comentado será o seguinte:

```
public void excluir (int id){

    String sql = "DELETE FROM empresa WHERE id = ?";
    try {
        //esse trecho é igual ao método editar e inserir
        PreparedStatement stmt = this.conn.prepareStatement(sql);
        stmt.setInt(1, id);

        //Executando a query
        stmt.execute();
        //tratando o erro, caso ele ocorra
    } catch (Exception e) {
        System.out.println("Erro ao excluir empresa: " + e.getMessage());
    }

}
```

No seu **FormEmpresa**, crie um botão para deletar. O nome da variável do botão será **btnExcluir**, ficando conforme o exemplo:



Empresa

Pesquisar empresa por ID: **Pesquisar**

ID:

Nome da Empresa:

Área de atuação:

Salvar **Editar** **Excluir**

Figura 6 – Formulário com as opções **Pesquisar**, **Salvar**, **Editar** e **Excluir**

Fonte: Senac EAD (2022)

As ações de pesquisa e edição devem estar funcionando para que a ação de excluir seja executada sem erro. Caso seu programa esteja com algum erro, verifique o seu código.

Configure agora o botão **Excluir** clicando nele com o botão direito do *mouse* e, em seguida, clicando em **Events** > **Action** > **actionPerformed**. O código será bem simples e semelhante ao código dos demais botões:


```
private void btnExcluirActionPerformed(java.awt.event.ActionEvent evt) {  
    //Pegar o código que o usuário digitou no campo de texto, converte para inteiro  
    e salvar da variavel id  
    int id = Integer.parseInt(txtId.getText());  
  
    EmpresaDAO empresaDAO = new EmpresaDAO();  
    empresaDAO.excluir(id);  
  
    //limpando os campos  
    txtId.setText("");  
    txtNomeEmpresa.setText("");  
}
```

Execute o seu formulário, pressionando **Shift + F6**, realize a busca de algum ID existente na tabela ou cadastre uma nova empresa; pesquise a empresa criada e depois a exclua.

Até agora, você já criou os métodos e botões para a opção de inserir, pesquisar por ID, editar e excluir. Seu próximo passo será realizar uma pesquisa mais dinâmica e avançada.

Criando método de pesquisa para buscar todos os cursos

Você agora criará um método em **EmpresaDAO** para retornar todas as empresas cadastradas no banco de dados. Então, abaixo do método **getEmpresa()**, crie este método:

```
public List<Empresa> getEmpresa()
```

Nesse método, utilize uma lista para preencher os dados na tabela que será adicionada mais à frente em seu **Form**. Esse método não tem um retorno específico, pois você quer todos os dados, por isso, não é necessário passar nenhum parâmetro dentro dos parênteses. Você terá que adicionar a importação referente à lista, que é: **import java.util.List;**

O primeiro passo é criar a *query* de seleção de todos os dados da tabela “empresa”:

```
public List<Empresa> getEmpresa() {  
    String sql = "SELECT * FROM empresa";  
}
```

Como usado anteriormente, dentro de um *try-catch*, o **PreparedStatement** controla e executa a instrução SQL que foi criada, juntamente ao **ResultSet** que conterá o conjunto de dados retornado pela sua consulta:

```
public List<Empresa> getEmpresa() {  
    String sql = "SELECT * FROM empresa";  
  
    try {  
        PreparedStatement stmt = this.conn.prepareStatement(sql);  
        ResultSet rs = stmt.executeQuery();  
  
    } catch (Exception e) {  
        return null;  
    }  
  
}
```

Agora, você precisa criar um objeto do tipo **List** (não esqueça de realizar a importação da classe **ArrayList**) para percorrer o **ResultSet** e salvar as informações dentro de um objeto do tipo **Empresa**, e depois salvar esse objeto dentro da lista. Use a estrutura *while*, que verificará se há ou não uma próxima posição dentro do *array*:

```
List<Empresa> listaEmpresas = new ArrayList<>();  
while (rs.next()) { //rs.next() retorna verdadeiro caso exista uma próxima posição de  
    ntro do array
```

Agora, você deve instanciar um objeto da classe **Empresa** para salvar as informações e também adicioná-lo à lista. Após finalizar o *while*, o seu retorno deve ser a lista que foi criada, na qual cada posição é um registro no banco de dados:

```
public List<Empresa> getEmpresa() {
    String sql = "SELECT * FROM empresa";

    try {
        PreparedStatement stmt = this.conn.prepareStatement(sql);
        ResultSet rs = stmt.executeQuery();

        List<Empresa> listaEmpresas = new ArrayList<>();

        while (rs.next()) { //.next retorna verdadeiro caso exista uma próxima po
        sição dentro do array
            Empresa empresa = new Empresa();

            empresa.setId(rs.getInt("id"));
            empresa.setNomeempresa(rs.getString("nomeempresa"));
            empresa.setAreaatuacao(rs.getString("areaatuacao"));

            listaEmpresas.add(empresa);
        }
        return listaEmpresas;

        //Se o método entrar no "Catch" quer dizer que não encontrou nenhuma empr
        esa, então damos um "return null"
    } catch (Exception e) {
        return null;
    }
}
```

Com isso, você conclui seu método **getEmpresa()**.

Criando a tabela para mostrar todos os registros do banco de dados

Agora você criará um novo formulário **JFrame**, no qual colocará uma tabela (**JTable**) para mostrar todos os registros cadastrados no banco de dados.

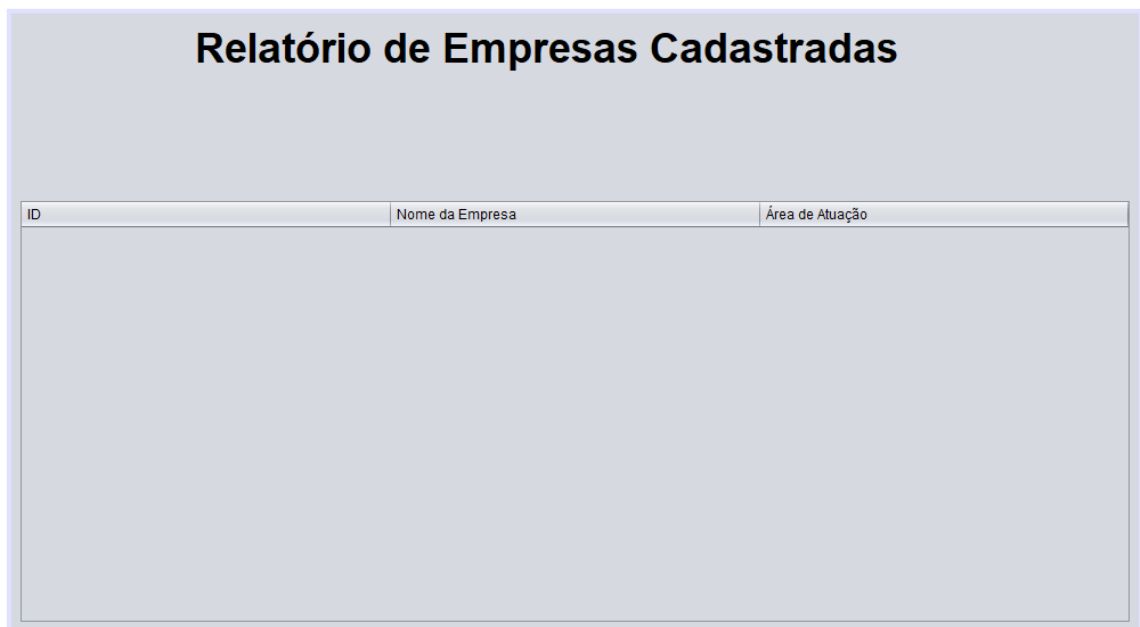
Para criar um novo formulário **JFrame**, clique com o botão direito do *mouse* no pacote **Forms > New > JFrame Form** e dê ao formulário o nome de "FormRelatorioEmpresas".

Antes de colocar os elementos em seu *frame*, você precisa efetuar alguns ajustes, que são:

* Para efetuar os ajustes, clique com o botão direito do *mouse* sobre o *frame* e clique em **Proprieties**.

- ◆ **defaultCloseOperation** = *dispose*
- ◆ **title** = Relatório de empresas
- ◆ Na aba **code**, marque a opção **generate center**.

Você terá uma estrutura semelhante a esta:



ID	Nome da Empresa	Área de Atuação
----	-----------------	-----------------

Figura 7 – Formulário que mostrará as empresas cadastradas no banco de dados
Fonte: Senac EAD (2022)

Após adicionar a **JTable**, que terá o nome da variável de “tblEmpresas” no seu *frame*, clique nela e vá na opção à direita **model**. Nesse local, você especificará as colunas (*column*) da sua tabela, que serão três. Na coluna 1, em **Title**, digite ID, na coluna 2, digite **Nome da Empresa** e, na coluna 3, digite **Área de Atuação**. Depois disso, apague a coluna 4. A opção **Rows** (linhas) deixe com 0 (zero), pois o número de registros que aparecerá será decidido pelo seu banco de dados.

Agora, vá ao código-fonte (*source*) do seu **JFrame**.

Abaixo do nome do seu **JFrame** (**public class FormRelatorioEmpresas extends javax.swing.JFrame {}**), crie um método para preencher a sua tabela, no qual você instanciará um objeto da classe **EmpresaDAO**, obtenha os dados das empresas que estão na lista e coloque-os na tabela, usando o seguinte:

```
private void preencherTabela() {  
    EmpresaDAO empresaDAO = new EmpresaDAO();  
  
    List<Empresa> listaEmpresas = empresaDAO.getEmpresas();  
}
```

Agora, você precisa instanciar um objeto da classe **DefaultTableModel**, pois é com esse tipo de tabela que você conseguirá inserir dinamicamente as linhas dentro da tabela (**JTable**):

```
DefaultTableModel tabelaEmpresas = (DefaultTableModel) tblEmpresas.getModel();
```

Para que você possa ordenar as suas colunas por ordem crescente e decrescente, utilize a classe **TableRowSorter** pertencente à classe **Table**, que fornecerá uma filtragem para a sua tabela. Quando o usuário clicar no cabeçalho de cada coluna, ele conseguirá realizar essa ordenação. A linha será a seguinte:

```
tblFuncionarios.setRowSorter(new TableRowSorter(tabelaFuncionarios));
```

* Será necessária a importação da classe **javax.swing.table.TableRowSorter**;

```
private void preencherTabela() {  
    EmpresaDAO empresaDAO = new EmpresaDAO();  
  
    List<Empresa> listaEmpresas = empresaDAO.getEmpresa();  
  
    DefaultTableModel tabelaEmpresas = (DefaultTableModel) tblEmpresas.getModel();  
  
    tblFuncionarios.setRowSorter(new TableRowSorter(tabelaFuncionarios));  
}
```

Agora você precisa percorrer a lista “listaEmpresas” e inserir na **JTable** “tabelaEmpresas”. Faça isso com o uso do laço *for*, em que, em cada volta do laço, ele adiciona um dado (uma empresa, nesse caso), dentro do objeto que, por fim, é adicionado na tabela. O código final do método fica como o a seguir:

```
private void preencherTabela() {  
    EmpresaDAO empresaDAO = new EmpresaDAO();  
  
    List<Empresa> listaEmpresas = empresaDAO.getEmpresa();  
  
    DefaultTableModel tabelaEmpresas = (DefaultTableModel) tblEmpresas.getModel();  
  
    tblFuncionarios.setRowSorter(new TableRowSorter(tabelaFuncionarios));  
  
    for (Empresa c : listaEmpresas) { //em cada volta do laço for, o mesmo adiciona  
        uma dado(empresa) dentro do objeto c  
        Object[] obj = new Object[] {  
            c.getId(),           //id  
            c.getNomeempresa(), //nomeempresa  
            c.getAreaatuacao(),  //areadeatuacao  
        };  
        tabelaEmpresas.addRow(obj);  
    }  
}
```

Importações necessárias:

```
import beans.Empresa;
import dao.EmpresaDAO;
import java.util.List;
import javax.swing.table.DefaultTableModel;
```

Depois de ser criado o método, “chame-o” em algum lugar. Ainda no código-fonte do **JFrame**, desça um pouco e logo em seguida haverá o construtor da classe. Adicione o corpo do seu método ao construtor, ficando assim:

```
public FormRelatorioEmpresas() {
    initComponents();
    preencherTabela(); //Será executado automaticamente sempre que esse form for exec
}
```

Após adicionar o método **preencherTabela** no construtor, execute o seu *frame*, pressionando **Shift + F6**. Note que todos os registros cadastrados do seu banco de dados estarão exibidos na tabela.

Agora, adicione um botão no *frame* **FormEmpresas** para que, quando se clicar nele, abra-se essa tabela com todos os registros cadastrados.

No **FormEmpresas**, adicione um botão ao lado do botão **Excluir**, com o nome “Empresas Cadastradas” e nome de variável “btnEmpCadast”. Depois disso, codifique o evento do botão, clicando com o botão direito do *mouse* sobre ele e então **Events > Action > actionPerformed**. O seu código ficará desta forma:

```
private void btnEmpCadastActionPerformed(java.awt.event.ActionEvent evt) {
    //Vinculando o formRelatorioEmpresas ao botão Empresas Cadastradas
    FormRelatorioEmpresas fre = new FormRelatorioEmpresas();
    //Mostrando o form quando clicar no botão
    fre.setVisible(true);
}
```

Agora execute o *form* **FormEmpresa** e clique no botão **Empresas Cadastradas** para ver se o **JFrame** que foi criado, que contém os registros, será exibido.

Até essa parte do seu projeto, você já conseguiu cadastrar, editar e excluir empresas. Além disso, já pode pesquisar a empresa por ID e também há a opção de visualizar todas as empresas cadastradas.

Utilizando filtro dinâmico

Neste momento, adicione um campo de pesquisa no seu **JFrame FormRelatorioEmpresa** para filtrar o nome da empresa.

Para isso, haverá um ajuste no seu método **public List<Empresa> getEmpresa()**, que está dentro da classe **EmpresaDAO**.

O método modificado ficará assim (as modificações estão em destaque):


```

public List<Empresa> getEmpresa(String nomeempresa ) { //parâmetro para buscar a em
    String sql = "SELECT * FROM empresa WHERE nomeempresa LIKE ?"; //LIKE nos permi

    try {
        PreparedStatement stmt = this.conn.prepareStatement(sql);
        //Como temos um parâmetro, devemos defini-lo
        stmt.setString(1,"%" + nomeempresa + "%"); //Conforme for a palavra ou letr
        //Método para poder executar o SELECT.
        //Os resultados obtivos pela consulta serão armazenados na variavel Results
        ResultSet rs = stmt.executeQuery();

        //Vamos criar um objeto do tipo List
        //Faça a importação do ArrayList
        List<Empresa> listaEmpresas = new ArrayList<>();
        //percorrer o resultSet e salvar as informações dentro de uma variável "Emp
        //Depois salva esse objeto dentro da lista

        //Estrutura de repetição While
        while (rs.next()) { //.next retorna verdadeiro caso exista uma próxima posi
            Empresa empresa = new Empresa();
            //Salvar dentro do objeto empresa as informações
            empresa.setId(rs.getInt("id"));
            empresa.setNomeempresa(rs.getString("nomeempresa"));
            empresa.setAreaatuacao(rs.getString("areaatuacao"));
            //Adicionando os elementos na lista criada
            listaEmpresas.add(empresa);

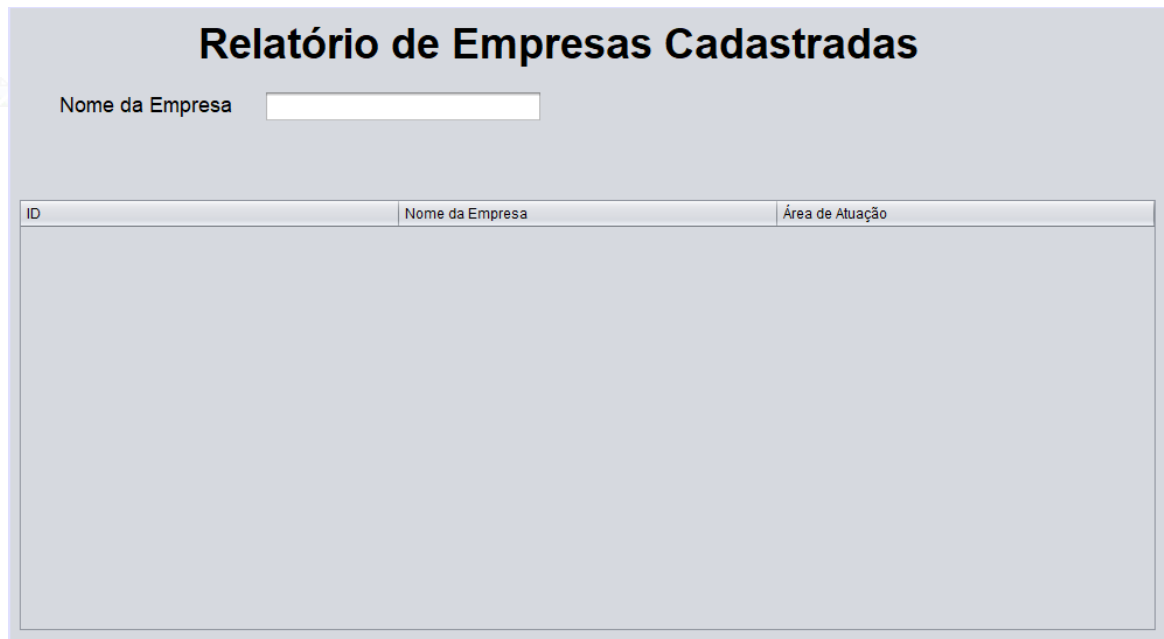
        }
        //Após finalizar o while, o retorno será a listaEmpresas, onde cada posição
        return listaEmpresas;

        //Se o método entrar no "Catch" quer dizer que não encontrou nenhuma empres
    } catch (Exception e) {
        return null;
    }

}

```

Volte ao *form* de relatórios ("FormRelatorioEmpresa"), vá até o código-fonte e veja que ele está indicando um erro na linha **List<Empresa> listaEmpresas = empresaDAO.getEmpresa();** pois o método agora exige um parâmetro. Esse parâmetro será digitado pelo usuário em um campo de texto dentro do *form* "FormRelatorioEmpresa", então você deve criar um campo de texto, com o nome de variável "txtNomeEmpresa", semelhante à imagem:



Relatório de Empresas Cadastradas

Nome da Empresa

ID	Nome da Empresa	Área de Atuação
----	-----------------	-----------------

Figura 8 – Formulário que mostrará as empresas cadastradas, conforme pesquisa do usuário

Fonte: Senac EAD (2022)

Agora, no código-fonte do seu *frame*, acima da linha que está com erro, crie uma *string* para armazenar o que o usuário digitará no campo de pesquisa e passe essa variável como parâmetro na linha que está apontando erro, ficando assim:

```
String nomeempresa = txtNomeEmpresa.getText();  
List<Empresa> listaEmpresas = empresaDAO.getEmpresa(nomeempresa);
```

Voltando ao *form*, na aba **design**, crie um evento para o campo de texto **txtNomeEmpresa**, que filtrará dinamicamente o nome de uma empresa quando você o digitar. Para isso, clique sobre o campo de texto com o botão direito do *mouse* e em seguida em **Events > Caret > caretUpdate**. Você somente chamará o método aqui dentro, ficando o código desta forma:

```
private void txtNomeEmpresaCaretUpdate(javax.swing.event.CaretEvent evt) {  
    preencherTabela();  
}
```

Se executar agora o seu *frame* e testar a pesquisa de alguma empresa, notará que, ao invés de filtrar, o método preencherá novamente a tabela com os mesmos dados. Para ajustar isso, vá ao “FormRelatorioEmpresa” e dentro do método **preencherTabela**, e toda vez que este método for chamado, antes de preencher as novas linhas, faça com que ele exclua as outras linhas que estão aparecendo. Para isso, adicione uma nova linha de código abaixo de **DefaultTableModel tabelaEmpresas = (DefaultTableModel) tblEmpresas.getModel();**, que será a seguinte:

```
//Limpar a tabela para preencher com os novos dados  
tabelaEmpresas.setNumRows(0);
```

Execute seu formulário com **Shift + F6** e tente filtrar alguma empresa por nome. Note que agora está funcionando.

Buscando empresas por faixa de IDs

Crie um método em **EmpresaDAO** para retornar todas as empresas cadastradas entre os IDs informados. Para isso, o seu “FormRelatorioEmpresas” deve estar semelhante a este:

Relatório de Empresas Cadastradas

Nome da Empresa

Pesquisar de um ID até outro ID

ID até ID

ID	Nome da Empresa	Área de Atuação
1	SENAC	Outros
2	Hospital Nossa Senhora Aparecida	Medicina
4	PC Info	Informática
7	CIEMED	Medicina
8	Gang	Outros
9	abc	Administração

Figura 9 – Formulário que mostrará as empresas cadastradas conforme pesquisa do usuário

Fonte: Senac EAD (2022)

Foram adicionados um *label*, com o texto “Pesquisar de um ID até outro ID”; um outro *label*, abaixo, com o texto “ID”; um campo de texto, ao lado, com o nome de variável “txtIdPesquisa”; ao lado deste, um *label* com o texto “até ID”; ao lado, um outro campo de texto com o nome de variável “txtIdPesquisa2”; ao lado, um botão com o texto “Pesquisar” e o nome da variável “btnPesquisaID”.

Volte à classe **EmpresaDAO** e crie o método, o qual receberá dois parâmetros para que seja realizado o filtro na lista.

```
public List<Empresa> getEmpresaPorId(int id1, int id2)
```

A *query* de consulta ao banco de dados fará a pesquisa por ID, do primeiro **id** fornecido até o outro **id** fornecido.

```
public List<Empresa> getEmpresaPorId(int id1, int id2) {  
    String sql = "SELECT * FROM empresa WHERE id >= ? AND id <= ?";  
  
    try {  
        PreparedStatement stmt = this.conn.prepareStatement(sql);  
  
        stmt.setInt(1, id1);  
        stmt.setInt(2, id2 );  
        ResultSet rs = stmt.executeQuery();  
  
        List<Empresa> listaEmpresas = new ArrayList<>();  
  
    } catch (Exception e) {  
        return null;  
    }  
  
}
```

Utiliza-se o *while*. Use o método **.next()** para retornar verdadeiro, caso exista um próxima posição dentro do *array*, salve esses registro dentro de uma variável “empresa”, adicione os elementos na lista que foi criada e por fim retorne essa lista após o *while* realizar toda a verificação.

```

public List<Empresa> getEmpresaPorId(int id1, int id2) {
    String sql = "SELECT * FROM empresa WHERE id >= ? AND id <= ?";

    try {
        PreparedStatement stmt = this.conn.prepareStatement(sql);

        stmt.setInt(1, id1);
        stmt.setInt(2, id2);
        ResultSet rs = stmt.executeQuery();

        List<Empresa> listaEmpresas = new ArrayList<>();
        //percorrer o resultSet e salvar as informações dentro de uma variável "Emp
resa"
        //Depois salva essa variavel dentro da lista

        //Estrutura de repetição While
        while (rs.next()) { //.next retorna verdadeiro caso exista uma próxima posi
ção dentro do array
            Empresa empresa = new Empresa();
            //Salvar dentro da variavel empresa, as informações
            empresa.setId(rs.getInt("id"));
            empresa.setNomeempresa(rs.getString("nomeempresa"));
            empresa.setAreaatuacao(rs.getString("areaatuacao"));
            //Adicionando os elementos na lista criada
            listaEmpresas.add(empresa);
        }
        //Após finalizar o while, o retorno será a listaEmpresas, onde cada posição
é um registro do banco de dados
        return listaEmpresas;

        //Se o método entrar no "Catch" quer dizer que não encontrou nenhuma empres
a, então damos um "return null"
    } catch (Exception e) {
        return null;
    }
}

```

Caso não se encontre nenhuma empresa, por algum erro na sintaxe ou também por não haver registros cadastrados, o método entrará no *catch*.

Depois de criar o método, você deve retornar ao “FormRelatorioEmpresas” para configurar a ação do botão **Pesquisar**.

Clique com o botão direito do *mouse* em **Events > Action > actionPerformed**.

Primeiramente, é preciso obter os IDs que o usuário digitou.

```
private void btnPesquisaIDActionPerformed(java.awt.event.ActionEvent evt) {  
    int idPesquisa = Integer.parseInt(txtIdPesquisa.getText());  
    int idPesquisa2 = Integer.parseInt(txtIdPesquisa2.getText());  
}
```

Crie um objeto da classe **EmpresaDAO** para acessar o método **getEmpresaPorId**. Depois disso, passe os valores digitados pelo usuário, que estão nas devidas variáveis.

```
private void btnPesquisaIDActionPerformed(java.awt.event.ActionEvent evt) {  
    int idPesquisa = Integer.parseInt(txtIdPesquisa.getText());  
    int idPesquisa2 = Integer.parseInt(txtIdPesquisa2.getText());  
    EmpresaDAO empresaDAO = new EmpresaDAO();  
  
    List<Empresa> empresa = empresaDAO.getEmpresaPorId(idPesquisa, idPesquisa2);  
}
```

O trecho restante é bem semelhante aos demais métodos de pesquisa, porém, quando se chama o objeto **empresaDAO**, passa-se o método **getEmpresaPorId** com dois parâmetros.

```
if (empresa == null) { //Se empresa for igual a
    JOptionPane.showMessageDialog(this, "Empresa não encontrado!");
}
else {
    List<Empresa> listaEmpresas = empresaDAO.getEmpresaPorId(idPesquisa, idPesquisa
2);

    DefaultTableModel tabelaEmpresas = (DefaultTableModel) tblEmpresas.getModel();

    tabelaEmpresas.setNumRows(0);

    for (Empresa c : listaEmpresas) {
        Object[] obj = new Object[] {
            c.getId(),           //id
            c.getNomeempresa(),  //nomeempresa
            c.getAreaatuacao(),  //areadeatuacao
        };

        tabelaEmpresas.addRow(obj);
    }
}
```

O método completo ficará desta forma:


```
private void btnPesquisaIDActionPerformed(java.awt.event.ActionEvent evt) {  
    int idPesquisa = Integer.parseInt(txtIdPesquisa.getText());  
    int idPesquisa2 = Integer.parseInt(txtIdPesquisa2.getText());  
    //Criar uma variavel do tipo EmpresaDAO. Criamos ela porque o método getEmpresa  
    PorId está dentro e precisamos acessar ele  
    EmpresaDAO empresaDAO = new EmpresaDAO();  
    //Chamar o método getEmpresa, que retorna uma variavel do tipo Empresa e que te  
    m como parâmetro o idPesquisa que o usuário digitou  
  
    List<Empresa> empresa = empresaDAO.getEmpresaPorId(idPesquisa, idPesquisa2);  
  
    //Verificação para saber se o curso existe ou não  
    if (empresa == null) { //Se empresa for igual a  
        JOptionPane.showMessageDialog(this, "Empresa não encontrado!");  
    }  
    else {  
        List<Empresa> listaEmpresas = empresaDAO.getEmpresaPorId(idPesquisa, idPesq  
        uisa2);  
  
        //Criar uma variavel do tipo DefaultTableModel, pois é com esse tipo que co  
        nseguimos inserir dinamicamente linhas dentro da JTable  
        DefaultTableModel tabelaEmpresas = (DefaultTableModel) tblEmpresas.getModel  
        ();  
  
        //Limpar a tabela para preencher com os novos dados  
        tabelaEmpresas.setNumRows(0);  
  
        //Percorrer o listaEmpresas e inserir na tabelaEmpresas  
        for (Empresa c : listaEmpresas) { //em cada volta do laço for, o mesmo adic  
        iona uma dado(empresa) dentro da variavel c  
            Object[] obj = new Object[] {  
                c.getId(), //id  
                c.getNomeempresa(), //nomeempresa  
                c.getAreaatuacao(), //areadeatuacao  
            };  
            //colocar os dados da variavel obj dentro da tabela  
            tabelaEmpresas.addRow(obj);  
        }  
    }  
}
```

Faça um teste em seu *form*: tente filtrar de um ID até outro e veja se tudo ocorrerá certo.

Criando uma nova tabela no banco de dados e relacionando-a com a tabela “empresas”

Agora, crie uma nova tabela em seu banco de dados, a fim de, posteriormente, criar um novo formulário, que agora será o de funcionários.

Explicação sobre a nova tabela:

Você deve criar uma tabela “funcionario”, que terá um campo ID, o qual será autoincremento e chave primária; um campo “nomefunc”, que será um **varchar** (100) e um campo “empresaid”, que será **int**. O campo “empresaid” será a chave estrangeira, que será o código da empresa (ID da empresa) à qual esse funcionário está vinculado.

O *script* SQL da criação da tabela funcionários, somente com os campos, será:

```
CREATE TABLE `funcionario` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `nomefunc` varchar(100) NOT NULL,  
  `empresaid` int(11) NOT NULL,  
  PRIMARY KEY (`id`)  
);
```

Você criará agora, de fato, a chave estrangeira para referenciar o campo “empresaid” com o campo “id” da tabela “empresa”. Para isso, utilize o seguinte código:

```
ALTER TABLE `funcionario` ADD CONSTRAINT `fk_empresa` FOREIGN KEY (`empresaid`) REFERENCES `empresa` (`id`) ON DELETE RESTRICT ON UPDATE RESTRICT;
```

Nome da restrição: `fk_empresa` (“fk” significa *foreign key*, ou “chave estrangeira”).

Feito isso, seu banco de dados está pronto.

Na sequência, volte ao seu projeto para criar a classe referente à tabela que foi criada e também os métodos para conexão com o formulário que será desenvolvido.

Criando as classes “beans.Funcionario” e “FuncionarioDAO”

Crie a classe **Funcionario** e **FuncionarioDAO** dentro do seu projeto.

Seguindo o padrão da orientação a objetos, você deve ter sempre uma classe para representar cada tabela em seu banco de dados.

Primeiramente, crie a classe **Funcionario**. Para isso, clique com o botão direito do *mouse* no pacote **beans** e em seguida em **New > Java Class**; o nome da classe será **Funcionario**.

A classe **Funcionario** terá os mesmos campos da tabela, porém agora serão os atributos na classe. Siga agora o mesmo padrão da classe **Empresa**, no qual você declarará os atributos privados e depois encapsulará os campos com **GET** e **SET**. A classe **Funcionario** ficará assim:

```
package beans;

public class Funcionario {

    private int id;
    private String nomefunc;
    private Empresa empresaid; // Ao invés de esse atributo ser int, aqui criamos
ele do tipo Objeto e fazemos uma associação devido a chave estrangeira
    public int getId() {
        return id;
    }

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

    public String getNomefunc() {
        return nomefunc;
    }

    public void setNomefunc(String nomefunc) {
        this.nomefunc = nomefunc;
    }

    public Empresa getEmpresaid() {
        return empresaid;
    }

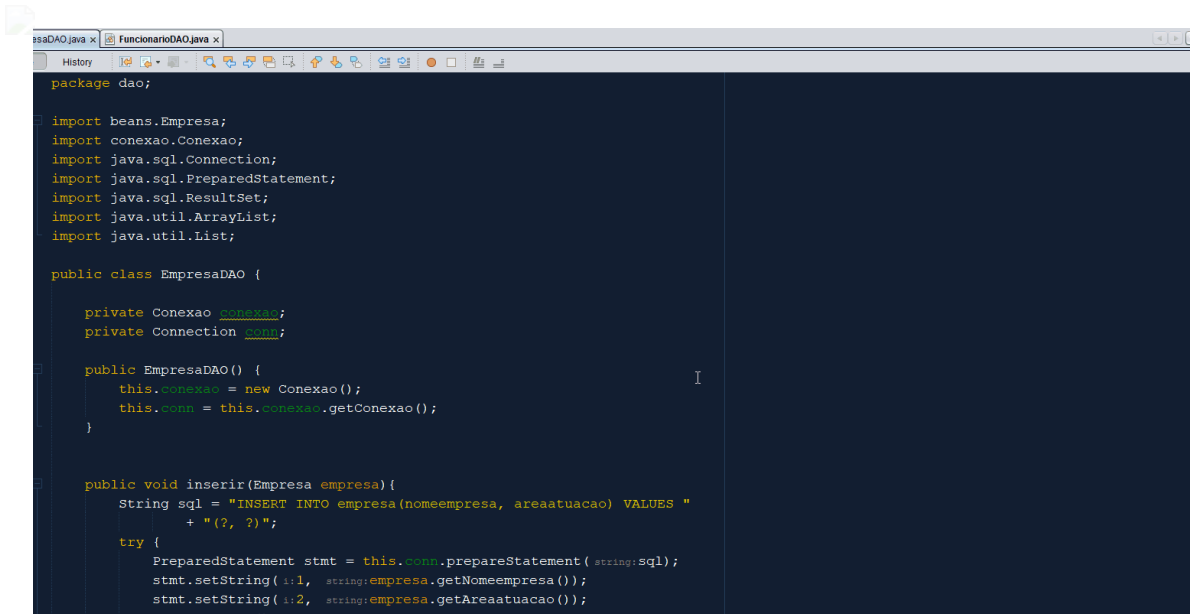
    public void setEmpresaid(Empresa empresaid) {
        this.empresaid = empresaid;
    }
}
```

Sua classe **Funcionario** está pronta!

Agora, crie a classe **FuncionarioDAO** dentro do pacote **dao**. Essa classe será responsável pelos métodos, assim como a classe **EmpresaDAO**. Para isso, clique com o botão direito do *mouse* no pacote **dao** e em seguida em **New > Java Class**; o nome da classe será **FuncionarioDAO**.

Para ficar mais fácil, e também para reaproveitar os códigos, copie quase todo o código da classe **EmpresaDAO**. Selecione o código depois da linha de declaração do nome da classe até a penúltima chave e cole na classe **FuncionarioDAO**.

Confira o GIF a seguir:



Ao colar o código na classe **FuncionarioDAO**, não se preocupe com os erros que ocorrerão, pois isso é comum por você ter copiado o código da classe **EmpresaDAO**.

De todo o código copiado, você apagará o método “getEmpresa”. (public List<Empresa> getEmpresa(String nomeempresa) e (public Empresa getEmpresa (int id)) <- Apagar esses métodos. Tenha cuidado com as chaves.

Deixe somente o “inserir”, o “editar” e “excluir”.

```

package dao;

import beans.Empresa;
import conexao.Conexao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;

public class FuncionarioDAO {

    private Conexao conexao;
    private Connection conn;

    public EmpresaDAO() {
        this.conexao = new Conexao();
        this.conn = this.conexao.getConexao();
    }

    public void inserir(Empresa empresa){
        String sql = "INSERT INTO empresa(nomeempresa, areaatuacao) VALUES "
            + "(?, ?)";
        try {
            PreparedStatement stmt = this.conn.prepareStatement(sql);
            stmt.setString(1, empresa.getNomeempresa());
            stmt.setString(2, empresa.getAreaatuacao());
            stmt.execute();

        } catch (Exception e) {
            System.out.println("Erro ao inserir empresa: " + e.getMessage());
        }
    }

    public void editar (Empresa empresa){
        //string sql com o código de update para o banco de dados
        String sql = "UPDATE empresa SET nomeempresa=?, areaatuacao=? WHERE id
=?";
        try {
            //esse trecho é igual ao método inserir
            PreparedStatement stmt = conn.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_UPDATABLE);
            //Setando os parâmetros
            stmt.setString(1, empresa.getNomeempresa());
            stmt.setString(2, empresa.getAreaatuacao());
            stmt.setInt(3, empresa.getId());
            //Executando a query
            stmt.execute();
            //tratando o erro, caso ele ocorra

```

```
        } catch (Exception e) {  
            System.out.println("Erro ao editar empresa: " + e.getMessage());  
        }  
    }  
  
    public void excluir (int id){  
        //string sql com o código de exclusão para o banco de dados  
        String sql = "DELETE FROM empresa WHERE id = ?";  
        try {  
            //esse trecho é igual ao método editar e inserir  
            PreparedStatement stmt = this.conn.prepareStatement(sql);  
            stmt.setInt(1, id);  
  
            //Executando a query  
            stmt.execute();  
            //tratando o erro, caso ele ocorra  
        } catch (Exception e) {  
            System.out.println("Erro ao excluir curso: " + e.getMessage());  
        }  
    }  
}
```

* Classe com os métodos, porém ainda não ajustada.

Agora, você fará algumas adaptações em sua classe, por exemplo, ajustará o construtor, as consultas SQL, o nome dos parâmetros, entre outros. A classe ficará assim:

```
package dao;

import beans.Funcionario;
import conexao.Conexao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class FuncionarioDAO {

    private Conexao conexao;
    private Connection conn;

    public FuncionarioDAO() {
        this.conexao = new Conexao();
        this.conn = this.conexao.getConexao();
    }

    public void inserir(Funcionario funcionario){
        String sql = "INSERT INTO funcionario(nomefunc, empresaid) VALUES "
            + "(?, ?)";
        try {
            PreparedStatement stmt = this.conn.prepareStatement(sql);
            stmt.setString(1, funcionario.getNomefunc());
            stmt.setInt(2, funcionario.getEmpresaid().getId());
            stmt.execute();

        } catch (Exception e) {
            System.out.println("Erro ao inserir funcionario: " + e.getMessage());
        }
    }

    public void editar (Funcionario funcionario){
        //string sql com o código de update para o banco de dados
        String sql = "UPDATE funcionario SET nomefunc=?, empresaid=? WHERE id=?";
        try {
            //esse trecho é igual ao método inserir
            PreparedStatement stmt = conn.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_UPDATABLE);
            //Setando os parâmetros
            stmt.setString(1, funcionario.getNomefunc());
            stmt.setInt(2, funcionario.getEmpresaid().getId());
            stmt.setInt(3, funcionario.getId());
            //Executando a query
            stmt.execute();
            //tratando o erro, caso ele ocorra
        } catch (Exception e) {
            System.out.println("Erro ao editar funcionario: " + e.getMessage());
        }
    }
}
```



```
}

public void excluir (int id){
    //string sql com o código de exclusão para o banco de dados
    String sql = "DELETE FROM funcionario WHERE id = ?";
    try {
        //esse trecho é igual ao método editar e inserir
        PreparedStatement stmt = this.conn.prepareStatement(sql);
        stmt.setInt(1, id);

        //Executando a query
        stmt.execute();
        //tratando o erro, caso ele ocorra
    } catch (Exception e) {
        System.out.println("Erro ao excluir funcionario: " + e.getMessage());
    }

}

}
```

Criando o formulário (JFrame) para inserir funcionários

Agora, você criará seu formulário para usar os métodos da classe **FuncionarioDAO**. Para isso, clique com o botão direito do *mouse* no pacote **forms** e em seguida em **New > JFrame Form** e dê o nome ao formulário de “FormFuncionario”.

Depois disso, faça alguns ajustes, clicando com o botão direito sobre ele e selecionando a opção **Properties**:

- ◆ **title** será “Formulário de Funcionários”.
- ◆ Na aba **code**, marque a opção **Generate Center** para que o formulário sempre seja gerado ao centro da tela.

Agora, disponha os elementos conforme a imagem:

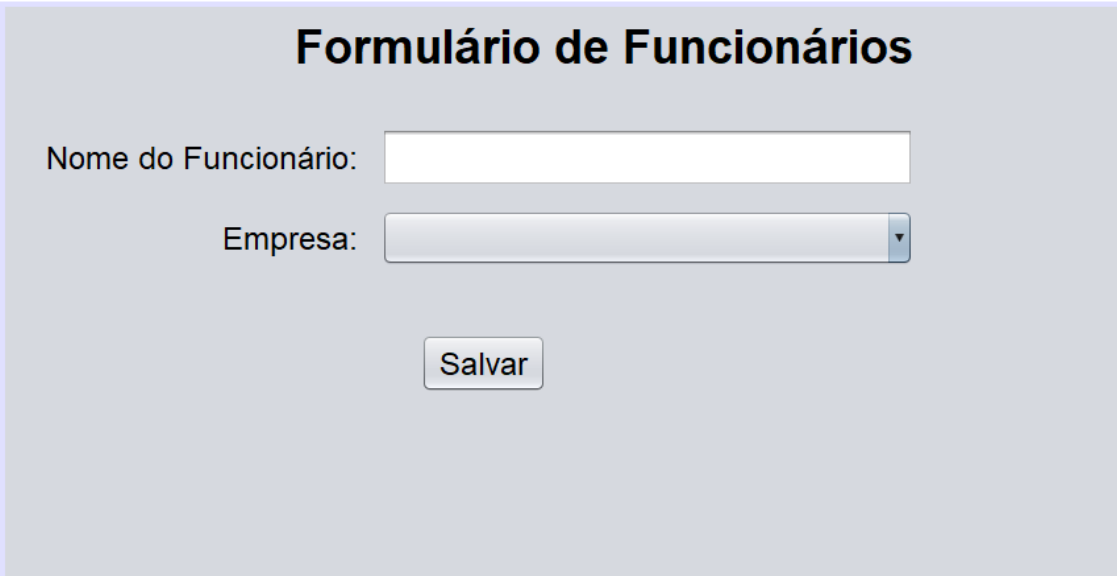
A imagem mostra uma interface de usuário para um formulário. No topo, há um título "Formulário de Funcionários" em negrito. Abaixo dele, há dois campos de entrada: "Nome do Funcionário:" seguido de um campo de texto branco, e "Empresa:" seguido de uma caixa de combinação (combobox) com uma seta para baixo no canto direito. Abaixo desses campos, há um botão "Salvar" com um contorno cinza.

Figura 10 – Formulário para salvar o nome de um funcionário e vincular a uma empresa existente no banco de dados

Fonte: Senac EAD (2022)

Explicando o formulário

Você colocará três labels, somente com título e rótulos dos campos. Também adicionará um campo de texto (“`TextField`”) ao lado do *label* que contém “Nome do Funcionário”, com a variável “`txtNomeFuncionario`”, e uma caixa de combinação (*combobox*) ao lado do label que contém “Empresa”, com o nome da variável “`cmbEmpresa`”.

Você deve também apagar o conteúdo que está preenchido por padrão dentro da caixa de combinação. Para fazer isso, clique sobre ela com o botão direito do *mouse* e vá em **Proprieties**. Haverá uma opção em negrito com o nome de **model**. Clique nos pontos mais à direita e uma caixa abrirá com todos os itens adicionados por padrão; então apague todos, pois os dados dessa caixa de combinação serão mostrados automaticamente (são as empresas cadastradas no campo “nomeempresa” da tabela “empresa”). Outro ajuste que deve ser feito nas propriedades da caixa de combinação será na aba **código** (*code*) e, na opção “Parâmetros de Tipo” (*Type Parameters*), apague o **<String>**, clicando nos pontinhos mais à direita. Esse ajuste é necessário devido ao fato de que você mostrará na caixa de combinação um objeto do tipo **Empresa**. Por padrão, a caixa de combinação só aceita textos (*string*). Adicione um botão também com o texto **Salvar** e nome da variável “`btnSalvar`”.

Antes de configurar o botão **Salvar**, faça com que a caixa de combinação mostre automaticamente as empresas cadastradas. Para fazer isso, vá até o código-fonte (*source*) do seu formulário “FormFuncionario” e, após a declaração do nome da classe (**public class FormFuncionario extends javax.swing.JFrame**), crie o método para preencher a caixa de combinação, que será o seguinte:

```
private void preencherComboEmpresa() {
    EmpresaDAO empresaDAO = new EmpresaDAO();
    List<Empresa> lista = empresaDAO.getEmpresa("");
    //Percorrer essa lista e cada empresa que ele achar, colocar dentro da caixa de
    combinação
    for (Empresa c : lista){
        cmbEmpresa.addItem(c);//devido ao ajuste que fizemos na caixa de combinaçã
        o, podemos adicionar a ela objetos, e não mais somente String
    }
}
```

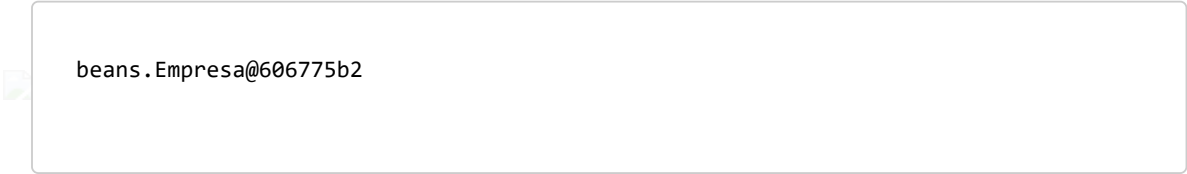
Importações necessárias

```
import beans.Empresa;
import beans.Funcionario;
import dao.EmpresaDAO;
import java.util.List;
```

Agora, faça o método dentro do formulário. Para isso, role a tela um pouco para baixo até o construtor e adicione o método que foi criado. O construtor ficará assim:

```
public FormFuncionario() {
    initComponents();
    preencherComboEmpresa();
}
```

Se você tentar executar o formulário agora (**Shift + F6**), terá, dentro da caixa de combinação, nomes semelhantes a estes:



```
beans.Empresa@606775b2
```

Essa informação significa a referência de cada registro no banco de dados. Para ajustar isso, crie um método **toString** na classe **Empresa**.

O método **toString** é a representação do objeto em *string* e significa que, quando um objeto do tipo **Empresa** for chamado, ele mostrará o que estiver sendo retornado por esse método.

Vá então até a classe **Empresa** e, abaixo do último **set**, declare-o da seguinte maneira:

```
@Override
public String toString()
{
    return this.nomeempresa;
}
```

Volte ao “FormFuncionario” e tente executá-lo novamente, com **Shift + F6**. Repare que agora estão sendo mostrados os nomes das empresas cadastradas em seu banco de dados.

Realizando a codificação para que o botão “Salvar” funcione corretamente

No formulário **FormFuncionario**, clique no botão **Salvar** com o botão direito do *mouse* e, em seguida, em **Events > Action > actionPerformed**. Seu código ficará desta forma:

```
private void btnSalvarActionPerformed(java.awt.event.ActionEvent evt) {  
    //Pegando os valores do formulário digitado e selecionado pelo usuário  
    String nomefunc = txtNomeFunc.getText();  
    Empresa empresaid = (Empresa) cmbEmpresa.getSelectedItem();  
  
    //Passando os valores fornecidos pelo usuário  
    Funcionario funcionario = new Funcionario();  
    funcionario.setNomefunc(nomefunc);  
    funcionario.setEmpresaid(empresaid);  
  
    //Chamando o método de inserção  
    FuncionarioDAO funcionarioDAO = new FuncionarioDAO();  
    funcionarioDAO.inserir(funcionario);  
  
    //Limpar os campos  
    txtNomeFunc.setText("");  
}
```

Para verificar se o método foi configurado corretamente, execute o formulário e tente realizar um cadastro. Verifique no banco de dados, em sua tabela “Funcionario”, que o dado foi armazenado.

Manipulando data no formulário e no banco de dados

Agora você criará mais um campo em sua tabela, o qual receberá a data de admissão do funcionário.

Note que, para esse único ajuste, você precisará efetuar os seguintes ajustes:

- ◆ No banco de dados: alterar a tabela “funcionario” para que ela tenha o campo “admissao”.
- ◆ No “formFuncionario”: criar um *label* com o texto “Data de admissão” e um campo de texto com o nome da variável “txtDataAdmissao”; ajustar o botão **Salvar** para que ele passe a data ao banco de dados, além do nome do funcionário e a empresa selecionada.
- ◆ Na classe **Funcionario**: criar o atributo “private Date dataAdmissao” e criar os métodos **GET** e **SET** desse atributo.
- ◆ Na classe **FuncionarioDAO**: ajustar o método **inserir**.

Comece seus ajustes!



O formulário de cadastro de funcionário ficará semelhante a este:

The image shows a web form titled "Formulário de Funcionários" on a light gray background. It contains three input fields: "Nome do Funcionário:" followed by a text box, "Empresa:" followed by a dropdown menu, and "Data de Admissão:" followed by a date picker. Below these fields is a button labeled "Salvar".

Figura 11 – Formulário para salvar admissão de funcionário

Fonte: Senac EAD (2022)

O código para você criar o campo “admissao” na tabela “funcionario” será:

```
ALTER TABLE funcionario ADD admissao date;
```

Você está realizando uma alteração na tabela “funcionario”, em que estará criando o campo “admissao” que será do tipo **Date**.

Agora, na classe **Funcionario**, dentro do pacote **beans**, crie o atributo “dataAdmissao”, que será do tipo **Date**:

```
private Date dataAdmissao;
```

* Não esqueça de criar os métodos **GET** e **SET** para esse atributo.

Na classe **FuncionarioDAO**, que está dentro do pacote **dao**, altere o método **inserir**, que ficará da seguinte forma:

Primeiro, você precisará criar um método para realizar a conversão da data para o padrão brasileiro. Utilize a classe **SimpleDateFormat**, usando o padrão de data “yyyy/MM/dd”. Tenha cuidado ao informar o mês, ele deve ser “MM”, pois, caso você use “mm” (com letras minúsculas), a classe entenderá que você está fornecendo os minutos. O bloco de código será este:

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd");
```

Na *query* SQL, adicione o campo “admissao” e também mais um parâmetro “?”:

```
String sql = "INSERT INTO funcionario(nomefunc, empresaid, admissao) VALUES (?, ?, ?)";
```

Dentro do *try*, a única alteração que será feita é usando o objeto instanciado da classe **SimpleDateFormat**. Você realizará a conversão da data:

```
stmt.setString(3, sdf.format(funcionario.getDataAdmissao()));
```

Ao final, você terá o seguinte método:

```
public void inserir(Funcionario funcionario){

    SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd");
    String sql = "INSERT INTO funcionario(nomefunc, empresaid, admissao) VALUES "
        + "(?, ?, ?)";
    try {
        PreparedStatement stmt = this.conn.prepareStatement(sql);
        stmt.setString(1, funcionario.getNomefunc());
        stmt.setInt(2, funcionario.getEmpresaid().getId());
        stmt.setString(3, sdf.format(funcionario.getDataAdmissao()));
        stmt.execute();

    } catch (Exception e) {
        System.out.println("Erro ao inserir funcionario: " + e.getMessage());
    }

}
```

* Será necessária a importação da classe **java.util.Date**;

Por fim, ajuste os métodos do botão **Salvar**, no **FormFuncionario**, que ficará assim:


```
private void btnSalvarActionPerformed(java.awt.event.ActionEvent evt) {  
    //Pegando os valores do formulário digitado e selecionado pelo usuário  
    String nomefunc = txtNomeFunc.getText();  
    Empresa empresaid = (Empresa) cmbEmpresa.getSelectedItem();  
  
    //Passando os valores fornecidos pelo usuário  
    Funcionario funcionario = new Funcionario();  
    funcionario.setNomefunc(nomefunc);  
    funcionario.setEmpresaid(empresaid);  
    //Essa classe fará a conversão da data para o padrão brasileiro  
    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");  
    try {  
        //Método parse converte de String para Date  
        funcionario.setDataAdmissao(sdf.parse(txtDataAdmissao.getText()));  
    } catch (ParseException ex) {  
        System.out.println("Erro ao converter o texto para date");  
    }  
  
    //Chamando o método de inserção  
    FuncionarioDAO funcionarioDAO = new FuncionarioDAO();  
    funcionarioDAO.inserir(funcionario);  
  
    //Limpar os campos  
    txtNomeFunc.setText("");  
}
```

* Será necessária a importação da classe **java.util.Date**;

Feitos esses ajustes, você pode executar o formulário usando **Shift + F6** e realizar o cadastro de um funcionário, informando a data de admissão dele. Note que, dentro do objeto da classe **SimpleDateFormat**, você está passando a data com o separador usando barras (dd/MM/yyyy). Logo, você deve informar a data nesse mesmo padrão. Caso você queira informar a data com o separador usando hífen, o padrão deveria ser dd-MM-yyyy.

Depois disso, verifique se, em seu banco de dados, o registro foi salvo.

Criando o campo de pesquisa para buscar o aluno pelo ID

Volte à classe **FuncionarioDAO** e crie um método para buscar o funcionário pelo ID. O código ficará desta forma:

```

public Funcionario getFuncionario (int id) {

    //código responsável por buscar o funcionário dentro do banco de dados
    String sql = "SELECT * FROM funcionario WHERE id = ?";
    try {
        PreparedStatement stmt = conn.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
        //PreparedStatement stmt = this.conn.prepareStatement(sql);
        //Passar o parâmetro da consulta
        stmt.setInt(1, id);

        //Método para poder executar o SELECT.
        //Os resultados obtidos pela consulta serão armazenados na variável ResultSet

        //Faça a importação da classe import java.sql.ResultSet;
        ResultSet rs = stmt.executeQuery();
        rs.next();

        //Criar um objeto do tipo empresa, que irá pegar os dados do rs(ResultSet) e armazenar na variável
        Funcionario funcionario = new Funcionario();

        rs.first(); //irá posicionar o ResultSet na primeira posição

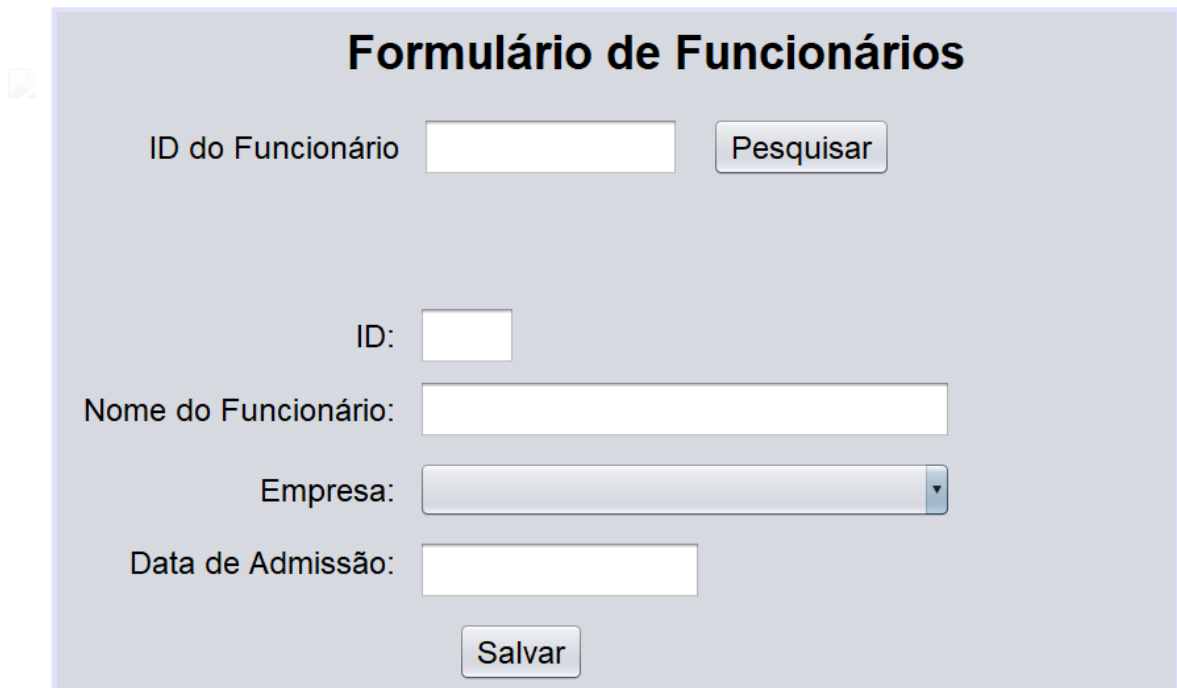
        //Atribuir os dados do "rs" para dentro do objeto funcionario
        //Iremos definir o que cada atributo irá mostrar, vinculando com a sua respectiva coluna no banco de dados
        funcionario.setId(id);
        funcionario.setNomefunc(rs.getString("nomealuno"));
        Empresa empresaid = new Empresa();
        empresaid.setId(rs.getInt("empresaid"));
        funcionario.setEmpresaid(empresaid);
        funcionario.setDataAdmissao(rs.getDate("admissao"));

        //retornar o objeto funcionario, pois o método pede esse retorno
        return funcionario;

        //tratando o erro, caso ele ocorra
    } catch (Exception e) {
        System.out.println("erro: " + e.getMessage());
        return null;
    }
}

```

Depois disso, volte ao formulário “FormFuncionario” e coloque os campos necessários para que seja possível pesquisar por ID e também para que seja mostrado o ID do funcionário. Seu formulário ficará assim:



The image shows a web form titled "Formulário de Funcionários" (Employee Form). It contains the following elements:

- A label "ID do Funcionário" followed by a text input field and a "Pesquisar" (Search) button.
- A label "ID:" followed by a small text input field.
- A label "Nome do Funcionário:" followed by a long text input field.
- A label "Empresa:" followed by a dropdown menu.
- A label "Data de Admissão:" followed by a date input field.
- A "Salvar" (Save) button at the bottom.

Figura 12 – Formulário para inserir e pesquisar por ID um funcionário

Fonte: Senac EAD (2022)

Foram adicionados um *label* com o texto “ID do funcionário” e um campo de texto com o nome da variável “txtPesquisar”; um *label* com o texto “ID” e a propriedade “editable” desmarcada para que o usuário não preencha esse campo. Ao lado, há um campo de texto com o nome da variável “txtID” e, em seguida, um botão com o texto **Pesquisar** e o nome da variável “btnPesquisar”.

Agora, crie o evento do botão pesquisar, para que ele, de fato, pesquise o ID informado. Clique com o botão direito do *mouse* no botão **Pesquisar** e vá até **Events > Action > actionPerformed**.

O código ficará do seguinte modo:

```
private void btnPesquisarActionPerformed(java.awt.event.ActionEvent evt) {  
    //Pegar o ID que será pesquisado  
    int id = Integer.parseInt(txtPesquisar.getText());  
    FuncionarioDAO funcionarioDAO = new FuncionarioDAO();  
  
    Funcionario f = funcionarioDAO.getFuncionario(id);  
  
    //Preencher os campos  
    txtId.setText(String.valueOf(f.getId()));  
    txtNomeFunc.setText(f.getNomefunc());  
    txtDataAdmissao.setText(String.valueOf(f.getDataAdmissao()));  
    cmbEmpresa.setSelectedItem(f.getEmpresaid());  
}
```

Ajustando a classe “Empresa”

Depois do método **toString**, adicione um método **Equals**, que definirá qual será a regra para comparar dois objetos do tipo **Empresa**. Os dois objetos serão iguais, se o ID de ambos for igual. O código será este:

```
@Override  
public boolean equals (Object objeto)  
{  
    Empresa e = (Empresa) objeto;  
    if (this.id == e.getId()){  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

* Sem o código anterior, a pesquisa de funcionário por ID não funciona corretamente. O registro é encontrado, mas o nome da empresa não consegue ser alterado corretamente caso o usuário realize outra pesquisa.

Criando o relatório de funcionários

Crie um relatório de funcionários, no qual será feita uma consulta usando o **JOIN**.

Explicando a *query*:



```
String sql = "SELECT funcionario.id as id, nomefunc, empresaid, nomeempresa, admissao FROM funcionario INNER JOIN empresa ON funcionario.empresaid = empresa.id";
```

Busque o campo **id** da tabela “funcionário”, o qual foi renomeado para somente **id**, os campos **nomefunc**, **empresaid**, **nomeempresa** e **admissao**, nos quais será feita a comparação das linhas de cada tabela para que o atributo **empresaid**, que é a chave estrangeira da tabela “funcionário”, seja igual a ele mesmo.

Vá até a classe **FuncionarioDAO** para criar o método, que será o seguinte:

```
public List<Funcionario> getFuncionarios() {
    String sql = "SELECT funcionario.id as id, nomefunc, empresaid, nomeempresa,
admissao FROM funcionario "
        + "INNER JOIN empresa ON funcionario.empresaid = empresa.id";
    try {
        PreparedStatement stmt = this.conn.prepareStatement(sql);
        ResultSet rs = stmt.executeQuery();
        List<Funcionario> lista = new ArrayList<>();
        while(rs.next()){
            Funcionario funcionario = new Funcionario();
            Empresa empresa = new Empresa();

            funcionario.setId(rs.getInt("id"));
            funcionario.setNomefunc(rs.getString("nomefunc"));
            funcionario.setDataAdmissao(rs.getDate("admissao"));
            empresa.setId(rs.getInt("empresaid"));
            empresa.setNomeempresa(rs.getString("nomeempresa"));
            funcionario.setEmpresaid(empresa);

            lista.add(funcionario);
        }
        return lista;
    } catch (Exception e) {
        return null;
    }
}
```

Agora, crie um novo **JFrame**, que terá o nome de “FormRelatorioFuncionarios”.



ID	Nome	Empresa	Data de Admissão
----	------	---------	------------------

Figura 13 – Formulário que mostrará os funcionários cadastrados no banco de dados

Fonte: Senac EAD (2022)

Este é o método para preencher a tabela (deve ser incluído na classe **FormRelatorioFuncionarios**):

```

private void preencherTabela()
{
    //Criar um objeto "FuncionarioDAO"
    FuncionarioDAO funcionarioDAO = new FuncionarioDAO();
    //Pegar os dados dos funcionarios da lista e colocar dentro da tabela
    List<Funcionario> listaFuncionarios = funcionarioDAO.getFuncionarios();

    //Criar uma variavel do tipo DefaultTableModel, pois é com esse tipo que
    conseguimos inserir dinamicamente linhas dentro da JTable
    DefaultTableModel tabelaFuncionarios = (DefaultTableModel) tblFuncionario
s.getModel();
    //permite clicar nas colunas para ordenar por ordem crescente ou decrese
nte

    tblEmpresas.setRowSorter(new TableRowSorter(tabelaEmpresas));
    //Limpar a tabela para preencher com os novos dados
    tabelaFuncionarios.setNumRows(0);

    //Percorrer o listaFunc inserir na tabelaFuncionarios

    for (Funcionario f : listaFuncionarios) { //em cada volta do laço for, o
mesmo adiciona uma dado(funcionario) dentro da variavel f
        Object[] obj = new Object[] {
            f.getId(),                //id
            f.getNomefunc(),          //nomefuncionario
            f.getEmpresaid().getNomeempresa(), //Empresa
            f.getDataAdmissao()       //Data de Admissão

        };
        //colocar os dados da variavel obj dentro da tabela
        tabelaFuncionarios.addRow(obj);
    }
}

```

Chame o método dentro do construtor:

```

public FormRelatorioFuncionarios() {
    initComponents();
    preencherTabela();
}

```

Execute usando **Shift + F6** e assim serão exibidos os registros.

Realize o cadastramento de alguns funcionários, para que a sua tabela fique com registro, como o exemplo a seguir:

Relatório de Funcionários			
ID	Nome	Empresa	Data de Admissão
1	Gustavo	SENAC	
2	Maria	PC Info	
3	Brunna	Gang	
4	Pedro	SENAC	2022-11-01
5	Bruno	SENAC	2010-10-03
6	Daniela	Hospital Nossa Senhora Aparecida	2022-10-01
7	Jorge	CIEMED	2022-02-10

Figura 14 – Relatório de funcionários

Fonte: Senac EAD (2022)

Encerramento

O uso de banco de dados dentro de projetos em Java é muito importante para que se possa armazenar, editar, alterar e excluir os dados, conforme a necessidade.

Quando se tem esse conceito e consegue-se criar uma interface gráfica para fazer essas funções, o programa fica muito mais acessível e de fácil entendimento, por isso, é necessário saber quais passos e procedimentos devem ser feitos para que tudo funcione corretamente.