



Banco de Dados

Prof. Ms. Claudiney Sanches Júnior

Java Data Base Connectivity

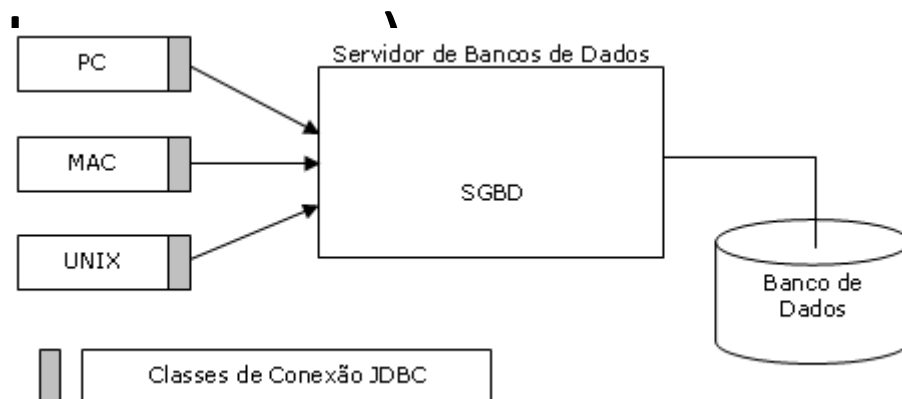
- uma API - *Application Program Interface* - que permite a conexão com bancos de dados
- é possível acessar os mais diversos bancos de dados, tabelas, registros e manipular as informações
- permite que Applets, servlets e outras aplicações acessem os dados dos bancos mais populares
- A linguagem mais utilizada para o acesso aos bancos de dados é a SQL

JDBC

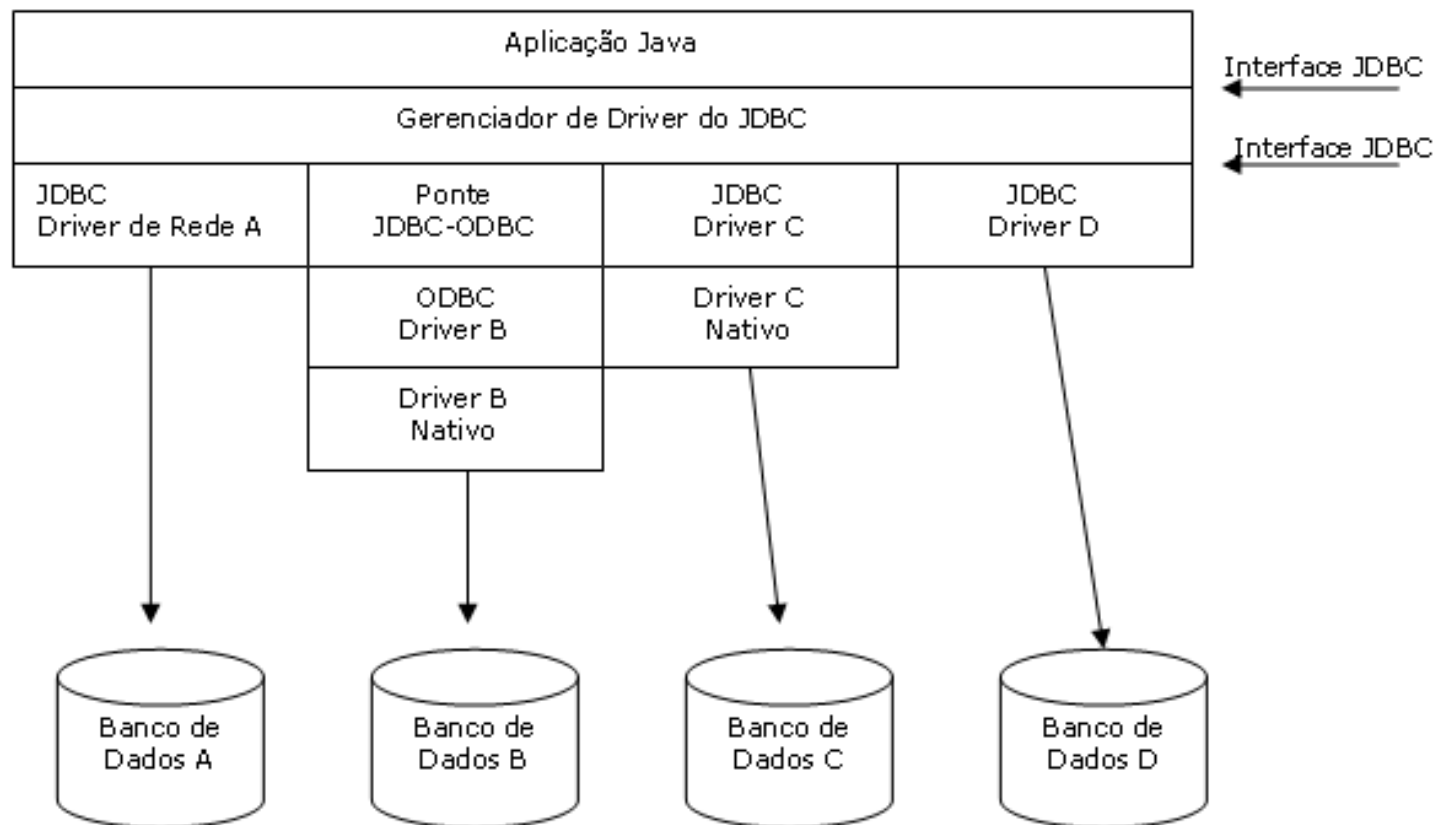
- é um conjunto de classes e interfaces
- foram escritas em Java para realizar a conexão com o Banco de Dados
- cada banco de Dados é necessário o Driver para conexão
- Oracle possui um *driver* próprio para a conexão

JDBC

- faz a ligação entre os sistemas desenvolvidos em Java com o Banco de Dados através das instruções em SQL (Structured Query

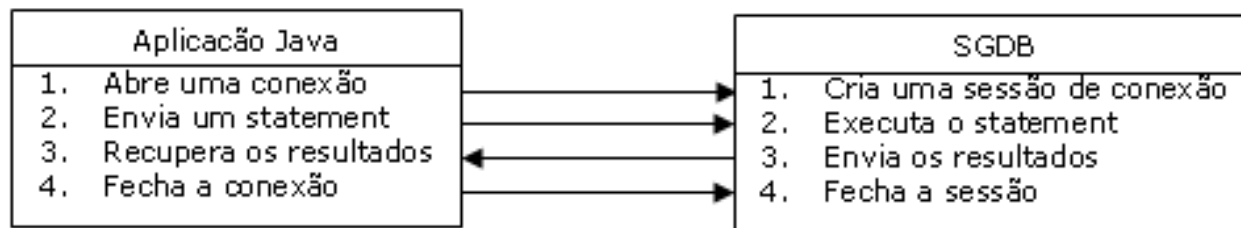


Componentes JDBC



Funcionamento

- a aplicação envia uma mensagem ao gerenciador de drivers do JDBC
- por sua vez instancia o driver correto
- abre uma sessão de conexão junto ao banco de dados que foi solicitado



Esquema básico de conexão

- A conexão é um objeto Java que contém **métodos** para acessar o banco de dados
- A classe mantém informações sobre o **estado** das conexões
- Vários parâmetros são necessários, como a **localização** do banco de dados, os **tipos de drivers** e **protocolos**, **usuário** e **senha** no SGBD
- Isto é conhecido como **esquema de nomes JDBC**

Driver

- atua como um tradutor entre a aplicação JAVA e o banco de dados relacional
- Estabelece as comunicações através de protocolos que garantem a correta troca de informações entre as duas partes

sql.java

- essas classes e interfaces são agrupadas e organizadas dentro do pacote

Classes	Interface
DriverManager	Driver
	Connection
	Statement
	ResultSet
	PreparedStatement

DriverManager

- é a classe que registrar o ***driver***
- no caso do JAVA o *driver* vai ser um arquivo “.jar”
- pode ser obtido nos sites dos fabricantes do SGBDs

Connection

- representa a própria conexão com o banco de dados
- os principais métodos:
- **close** – fecha uma conexão
- **isClosed** – verifica se a conexão está fechada
- **createStatement** – cria um objeto que é capaz de executar instruções SQL no SGBD
- **preparedStatement** – é usado para “montar” uma instrução SQL que será executada no SGBD

Statement

- reúne os métodos de execução de ***query***
 - que é o processo de execução das instruções da SQL diretamente no banco
- os dois mais importantes:
- **executeQuery** – executa instruções SQL de consulta (leitura) no SGBD, esse método retorna ao fim de sua execução um objeto de ResultSet
- **executeUpdate** – executa instruções de criação, alteração e exclusão de registros no SGBD

ResultSet

- representa um conjunto de resultados retornados de uma tabela do banco de dados
- os dados são organizados dentro do objeto através de um cursor que sempre começa no índice zero
- o índice zero é sempre vazio então se faz necessário mover o curso para o índice um 1 através do método **next()**

ResultSet

- sempre que for necessário passar para o próximo índice o método **next()** deve ser chamado
- o acesso as colunas da tabela podem ser feitas pelo nome das colunas ou pelo valor numérico referente a coluna, nesse caso sempre iniciando por um (1)

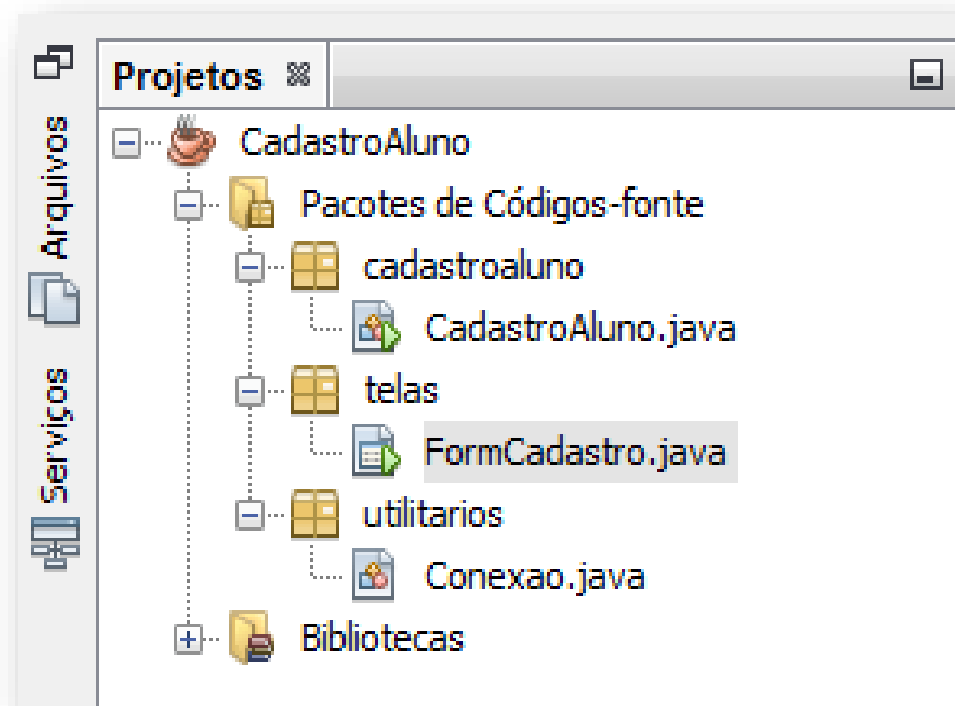
ResultSet

- possui métodos de **getter** capazes de recuperar valores dos campos da tabela baseados em seus tipos
- permite a conversão automática e correta dos tipos do banco de dados do SGBD para os tipos do JAVA
- São eles:
 - getInt, getString, getDouble, getFloat, getLong entre outros

Criando o projeto

- crie um projeto chamado de **CadastroAluno**
- adicione mais dois pacotes além do pacote padrão um chamado **utilitarios** e outro chamado **telas**
- crie uma classe chamada **Conexao** no pacote **utilitarios**
- crie um **JFrame** chamado **FormCadastro** no pacote **telas**

Estrutura inicial do projeto



Adquirindo e configurando o driver do Oracle

- Agora vamos adquirir e configurar o driver de conexão do Oracle no nosso projeto
- Para a adquirir o driver de conexão do Oracle basta a acessar o link:

<http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html>

ou acessar: ojdbc7.jar

[\\Sanches\Oracle](#)

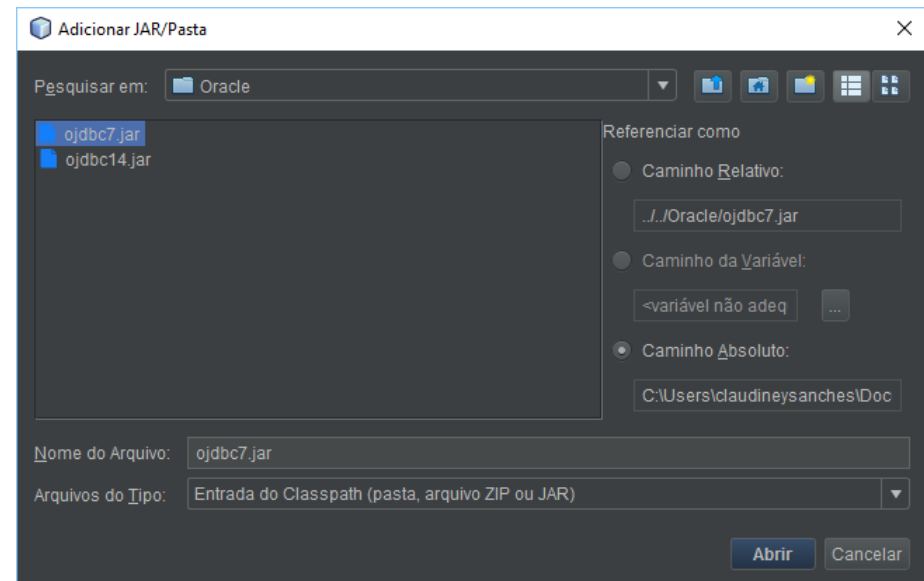
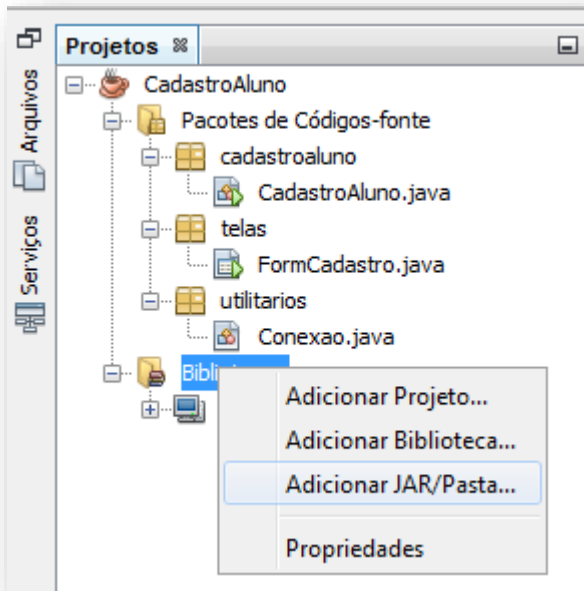
Usuário: Aluno

Senha: 123456

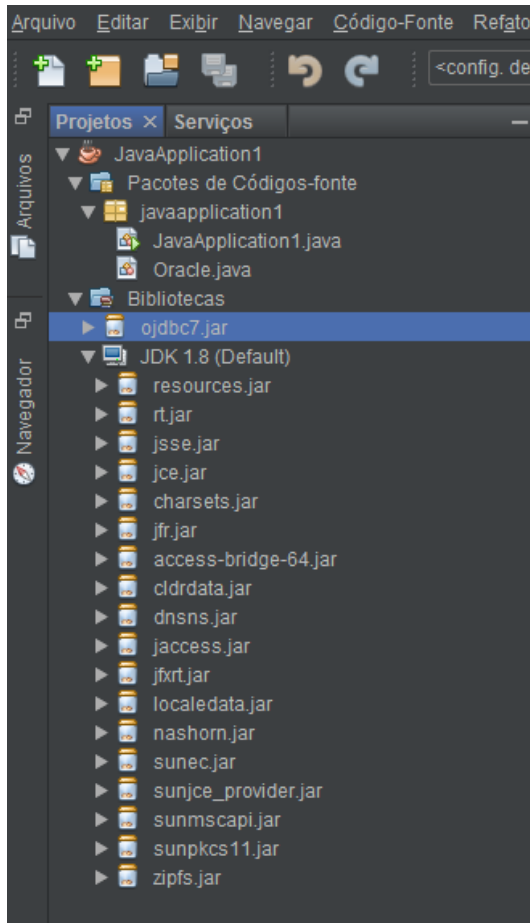
Configurar o driver

- no NetBeans é bastante simples
- selecione com o botão direito na pasta **Bibliotecas**
- clicar em seguida na opção **Adicionar JAR/Pasta**
- navegar até o local onde se localiza o arquivo do driver

Configurar o driver

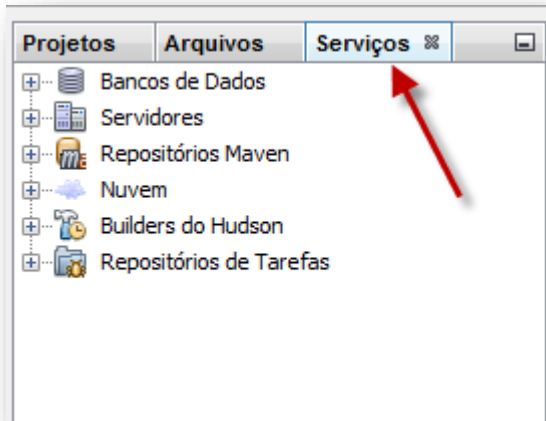


Criar uma conexão

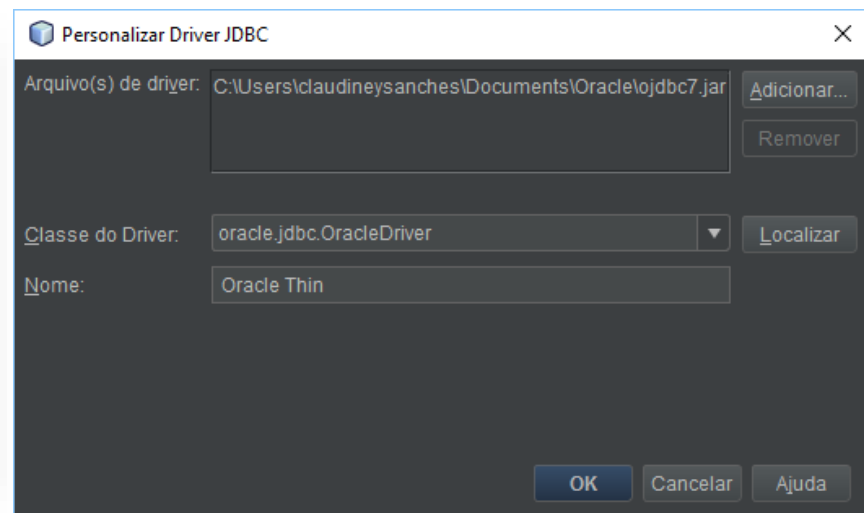
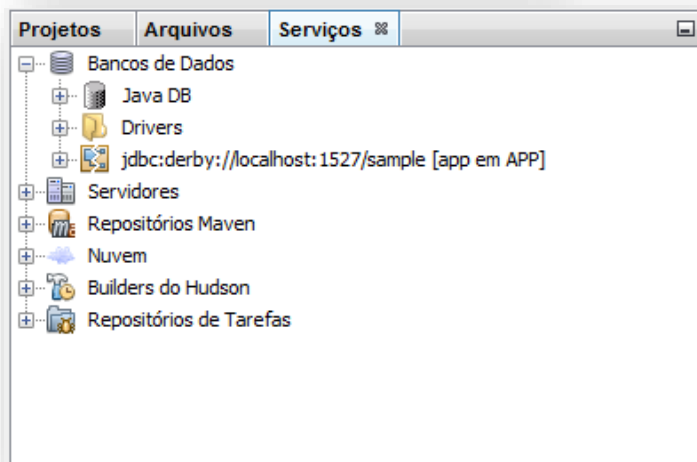


- agora que já temos o driver devidamente inserido
- criar uma conexão (que não vai ser usada no sistema)
- se faz necessário para podermos montar a base de dados sem ter a necessidade de migrar para uma IDE como o SQL Developer

Acesse a aba serviços

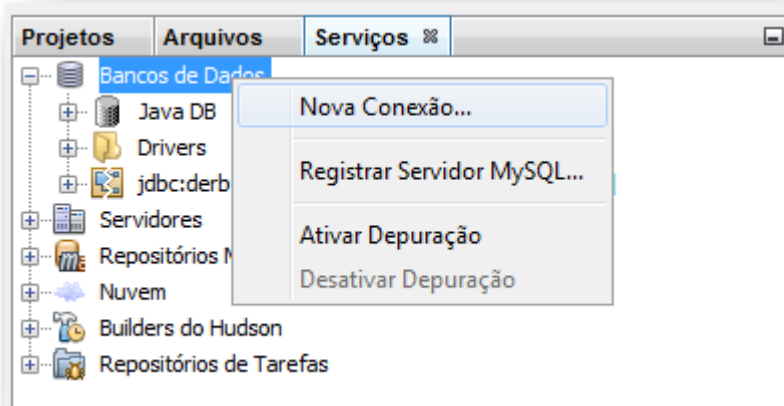


- na aba serviços temos um grupo chamado **Banco**
- clique no sinal de + a esquerda desse para expandi-lo
- Abra Drivers
- Novo drivers



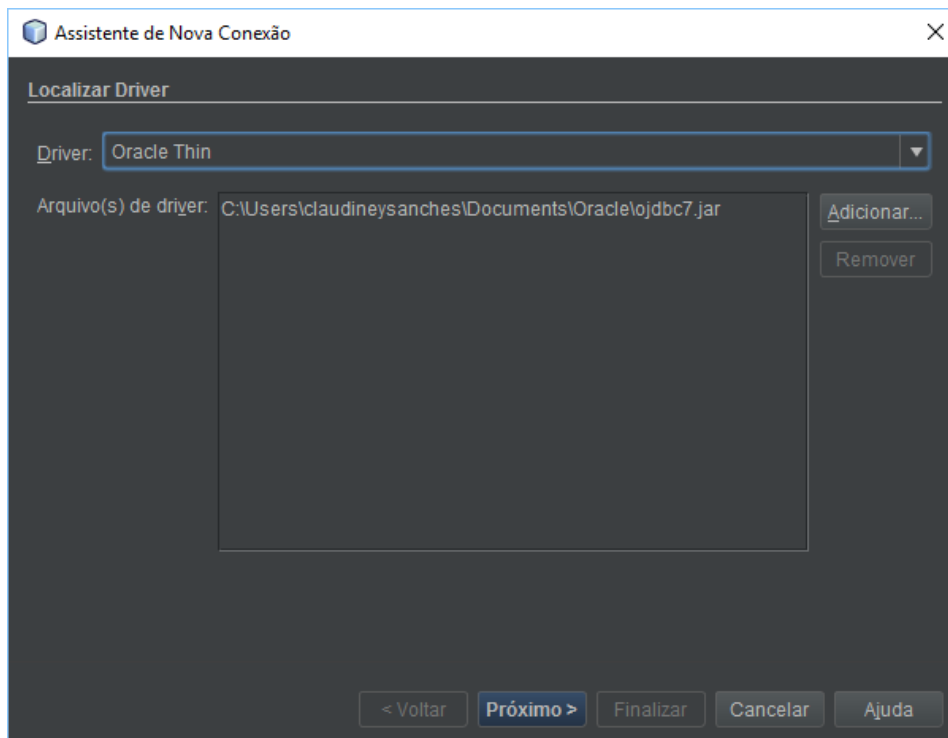
Para criarmos uma conexão

- clicar com o botão direito sobre o ícone com o rotulo **Banco de Dados**
- em seguida na opção **Nova Conexão**



Para criarmos uma conexão

- escolha o driver do banco de dados
- selecione a opção **Oracle Thin**
- clicar em próximo



Configuração

Assistente de Nova Conexão

Personalizar Conexão

Nome do Driver: Oracle Thin (ID do Serviço (SID)) ▼

Host: localhost Porta: 1521

ID do Serviço (SID): XE

Nome do Usuário: hr

Senha: ••

☐ Lembrar senha

Propriedades da Conexão Testar Conexão

JDBC URL: jdbc:oracle:thin:@localhost:1521:XE

< Voltar Próximo > Finalizar Cancelar Ajuda

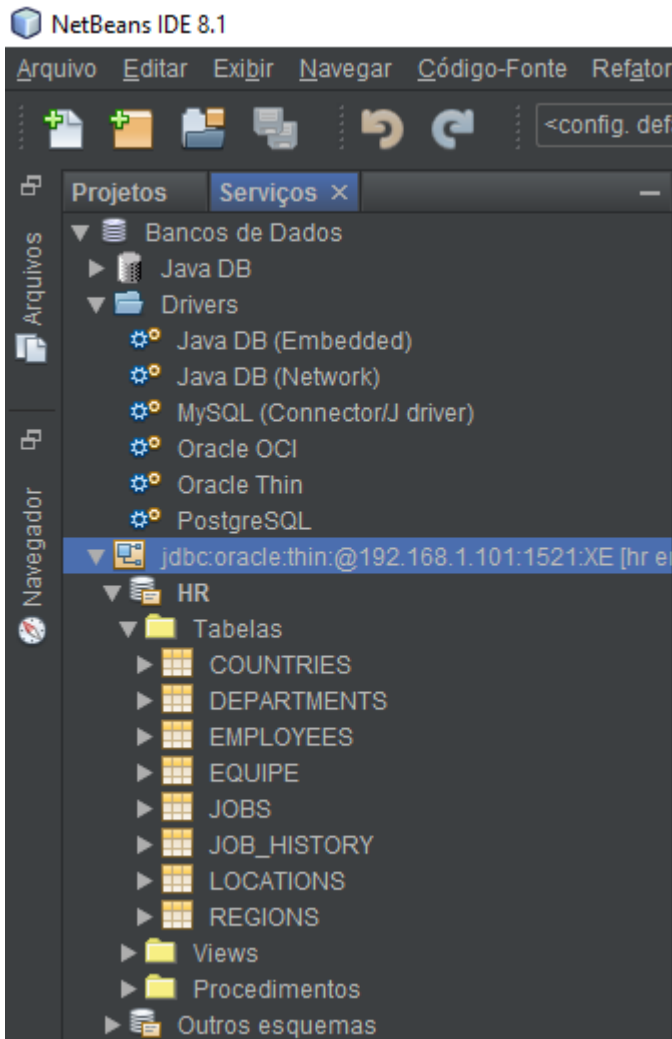
Configuração

- **Host** – local onde fica o servidor de dados, nesse caso **localhost** ou **127.0.0.1** também conhecido como hospedeiro local, o ambiente de desenvolvimento deverá ser o ip da VM
- **Porta** – é a porta logica que o Oracle usa para comunicação e transmissão de dados o padrão é **1521**, mas pode ser mudada

Configuração

- **Nome do usuário** – é o usuário que vai usado para **logar** no banco de dados
- **Senha** – é a senha do usuário que vai ser usada para **logar** no banco de dados
- **JDBC URL** – o endereço do driver JDBC que será usado para realizar a conexão com o banco de dados

Conexão criada



- clicar com o botão direito do mouse sobre o banco desejado
- optar por **Defini como catalogo default**

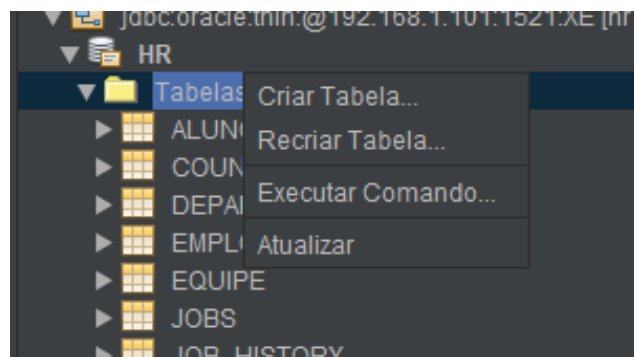
Criar um banco de dados novo com uma tabela

```
1 create table aluno(  
2     cod_aluno number(4) primary key,  
3     nome varchar2(200) not null,  
4     idade number(3) ,  
5     email varchar2(150) ,  
6     telefone varchar2(25)  
7 );  
8
```

Executar o código



- para digitar e executar o código basta



- dar um clique com o botão direito do mouse sobre o ícone da conexão com o **Oracle**
- escolher a opção **Executar comando...**

Desenhe o JFrame

The image shows a Java Swing JFrame window with a dark gray background and a light gray border. The window is divided into two main sections. The top section is titled 'Aluno' and contains five input fields: 'Código:' (a small rectangular field), 'Nome:' (a long rectangular field), 'Idade:' (a small rectangular field), 'e-mail:' (a small rectangular field), and 'Telefone:' (a long rectangular field). Below these fields are five buttons: 'Limpar', 'Inserir', 'Alterar', 'Excluir', and 'Selecionar'. The bottom section is titled 'Conexão' and contains a single input field labeled 'Conexão: ?' and a button labeled 'Sair'.

Aluno

Código:

Nome:

Idade: e-mail:

Telefone:

Limpar Inserir Alterar Excluir Selecionar

Conexão

Conexão: ? Sair

Nomes de variáveis dos componentes

Componente	Nome de variável do componente
Label de código	lbl_codigo
Caixa de texto de nome	txt_nome
Caixa de texto de idade	txt_idade
Caixa de texto de e-mail	txt_email
Caixa de texto de telefone	txt_telefone
Botão de Inserir	btn_Inserir
Botão de selecionar	btn_selecionar
Botão de alterar	btn_alterar
Botão de excluir	btn_excluir
Botão de sair	btn_sair

Classe de conexão

- o código da classe que realiza a conexão com o banco de dados e o sistema em JAVA

```
1 package javaapplication1;  
2 import java.sql.Connection;  
3 import java.sql.DriverManager;  
4 import java.sql.SQLException;  
5
```

Classe de conexão

- a declaração do pacote ao qual a classe pertence linha 2
- da linha 2 até a linha 4 temos declaração das importações necessárias para a classe **Conexao** poder abrir a conexão como **Oracle** todas as classes pertencem ao pacote **java.sql**
- a **SQLException** vai servir para exibir as mensagens de erros referentes ao banco de dados

```
6 public class Conexao {  
7     public static Connection abrirConexao() {  
8         Connection con = null;  
9         String url = "jdbc:oracle:thin:@192.168.1.101:1521:xe";  
10        String usr = "hr";  
11        String pwd = "hr";  
12        try {  
13            con = (Connection) DriverManager.getConnection(url,usr,pwd);  
14            return con;  
15        } catch (SQLException e) {  
16            System.out.println("Erro: "+e.getMessage());  
17        }  
18        return con;  
19    }  
20 }
```



- na linha 6 a declaração da classe
- na linha 12 temos a declaração de um método público chamado **abrirConexao()**
- observe que o método em questão possui um tipo de retorno que nesse caso é um objeto da classe **Connection** do JAVA
- o método **abrirConexao** retorna uma conexão valida e ativa no final de sua tarefa

- a linha 7 cria uma variável local do tipo **Connection** que vai armazenar o objeto de conexão que será retornado pelo método no final de sua execução
- a linha 13 inicia uma estrutura “**try...catch**” onde o “**try**” tenta fazer alguma coisa nesse caso abrir uma conexão com o banco de dados, caso ocorra algum problema uma exceção (erro) do JAVA será lançada e a estrutura **catch** vai tratar essa exceção

- a linha 9 cria uma **string** de conexão com o banco de dados
- **jdbc:oracle:thin:@** - indica que será usado o driver JDBC do Oracle
- **192.168.1.101** - endereço da hospedagem do servidor de dados
- **usr=hr** – passagem de parâmetro referente ao nome do usuário através do rotulo **user**, nesse caso o nome do usuário é **hr**
- **pwd=** - Passagem de parâmetro referente a senha do banco de dados do usuário informado através rotulo **password**

- a linha 14 abre a conexão através da classe **Drivermanager** e de seu método estático **getConnection** que deve receber como parâmetro a **string** de conexão
- o resultado é passado para a variável **con** que vai representar o objeto de conexão aberto (classe **Connection**)
- a linha 16 inicia a estrutura **catch** tratando qualquer erro de **SQL** como indica a classe **SQLException** que será representada dentro da estrutura **catch** pela variável de objeto **e**

Método fecharConexão()

```
20  
21    
22     public static void fechaConexao(Connection con) {  
23         try {  
24             con.close();  
25         } catch (SQLException e) {  
26             System.out.println("Erro: " + e.getMessage());  
27         }  
28     }
```


- o método responsável por fechar a conexão com o banco de dados **fecharConexao** também **static**
 - não temos que criar um objeto da classe **Conexao** para usar o método
- note que temos uma estrutura **try...catch** onde dentro do **try** linha 22 chamado o método **close** através do objeto da classe **Connection** representado pela variável **com**
- muito parecida a do método de abrir a conexão tratamos a exceção no **catch**, linha 24

```
29 public static boolean isConexao(Connection con){
30     String url = "jdbc:oracle:thin:@192.168.1.101:1521:xe";
31     String usr = "hr";
32     String pwd = "hr";
33     boolean isConnected = false;
34     try {
35         Class.forName("oracle.jdbc.driver.OracleDriver")
36             .newInstance();
37         con = DriverManager.getConnection(url,usr,pwd);
38         isConnected = true;
39     } catch (SQLException e) {
40         System.out.println(e.getMessage());
41         isConnected = false;
42     } catch ( ClassNotFoundException e ) {
43         System.out.println(e.getMessage());
44         isConnected = false;
45     } catch ( InstantiationException e ) {
46         System.out.println(e.getMessage());
47         isConnected = false;
48     } catch ( IllegalAccessException e ) {
49         System.out.println(e.getMessage());
50         isConnected = false;
51     }
52     System.out.println("Conexao: "+isConnected);
53     return isConnected;
54 }
55 }
56
```

Testar a Conexão

```

1 package cadastroAluno;
2 import java.sql.DriverManager;
3 import java.sql.Connection;
4 import java.sql.SQLException;
5 import java.sql.ResultSet;
6 import java.sql.PreparedStatement;
7 import javax.swing.JOptionPane;
8 /**
9  *
10  * @author claudineysanches
11  */
12 public class FormCadastro extends javax.swing.JFrame {
13     Connection con = Conexao.abrirConexao();
14     /** Creates new form FormCadastro ...3 linhas */
15
16
17
18     public FormCadastro() {
19
20         initComponents();
21         if (Conexao.isConexao(con))
22             jLabel1.setText("Conectado com sucesso!");
23         else
24             jLabel1.setText("Conexão falhou! :(");
25     }
26

```

Programação dos botões

```
10 public class FormCadastro extends javax.swing.JFrame {  
11  
12     Connection con = Conexao.abrirConexao();  
13  
14     public FormCadastro() {  
15         initComponents();  
16         btn_alterar.setEnabled(false);  
17         btn_excluir.setEnabled(false);  
18     }
```

```
209 private void btnInserirActionPerformed(java.awt.event.ActionEvent evt) {  
210     int codigo = Integer.parseInt(txtCodigo.getText());  
211     String nome = txtNome.getText();  
212     int idade = Integer.parseInt(txtIdade.getText());  
213     String email = txtEmail.getText();  
214     String telefone = txtTelefone.getText();  
215     String sql = "Insert Into Aluno (cod_aluno, nome, idade, email, telefone) values (?,?,?,?,?)";  
216  
217     try {  
218         PreparedStatement ps = con.prepareStatement(sql);  
219         ps.setInt(1, codigo);  
220         ps.setString(2, nome);  
221         ps.setInt(3, idade);  
222         ps.setString(4, email);  
223         ps.setString(5, telefone);  
224         ps.executeUpdate();  
225         JOptionPane.showMessageDialog(null, "Aluno Cadastrado com sucesso!", "Cadastro", JOptionPane.INFORMATION_MESSAGE);  
226         txtCodigo.setText("");  
227         txtNome.setText("");  
228         txtIdade.setText("");  
229         txtEmail.setText("");  
230         txtTelefone.setText("");  
231  
232     } catch (SQLException e){  
233         JOptionPane.showMessageDialog(null, "Erro: " + e.getMessage(), "Erro", JOptionPane.ERROR_MESSAGE);  
234     }  
235 }
```

Botão Alterar

```
246
247 private void btnAlterarActionPerformed(java.awt.event.ActionEvent evt) {
248     int codigo = Integer.parseInt(txtCodigo.getText());
249     String nome = txtNome.getText();
250     int idade = Integer.parseInt(txtIdade.getText());
251     String email = txtEmail.getText();
252     String telefone = txtTelefone.getText();
253     String sql = "update aluno set nome = ?, idade = ?, email = ?, telefone = ? where cod_aluno = ?";
254
255     try {
256         PreparedStatement ps = con.prepareStatement(sql);
257         ps.setString(1, nome);
258         ps.setInt(2, idade);
259         ps.setString(3, email);
260         ps.setString(4, telefone);
261         ps.setInt(5, codigo);
262         ps.executeUpdate();
263         JOptionPane.showMessageDialog(null, "Cadastrado Aletrado com sucesso!", "Alteração", JOptionPane.INFORMATION_MESSAGE);
264         txtCodigo.setText("");
265         txtNome.setText("");
266         txtIdade.setText("");
267         txtEmail.setText("");
268         txtTelefone.setText("");
269
270     } catch (SQLException e) {
271         JOptionPane.showMessageDialog(null, "Erro: " + e.getMessage(), "Erro", JOptionPane.ERROR_MESSAGE);
272     }
273 }
274
```

- das linhas 183 a 187 recuperamos os valores das caixas de texto através do método **getText** da classe **TextField**
- associamos esses valores a variáveis locais do método
- atenção extra para a linha 185 que atribui o valor da caixa de texto a uma variável do tipo **int**
- todo dado vindo de um **TextField** é do tipo **String**, logo se faz necessário a conversão do tipo **String** para **int** que é feito através do método **parseInt()** da classe **Integer**

- a linha 189 monta **query** SQL de inserção de dados, observe que no lugar dos valores são usadas interrogações (?)
- é necessário pois os dados precisam passar por uma **preparação** antes de serem enviados para o Oracle pois seus tipos de dados ainda estão definidos como do JAVA e não do Oracle
- ai que o JDBC entra com a abertura da conexão e a conversão para o tipo certo no banco de dados

- a linha 193 temos a criação do objeto da classe **PreparedStatement** que vai receber a declaração SQL permitindo assim que sejam informados os valores que serão atribuídos aos sinais de ? é nesse ponto que é feita a tradução dos tipos de valores do JAVA para o Oracle
- o código das linhas 195 até 199 informam o tipo, a ordem e o valor a ser passado para o objeto de **PreparedStatement**
- na linha 201 é executada a declaração SQL agora já devidamente **montada** para isso chamamos o método **executeUpdate** da classe **PreparedStatement**

Botão SELECIONAR

```
322 private void btnSelecionarActionPerformed(java.awt.event.ActionEvent evt) {  
323     int codigo = Integer.parseInt(JOptionPane.showInputDialog(null, "Informe um código", "Consultar", JOptionPane.QUESTION_MESSAGE));  
324     String sql = "select * from aluno where cod_aluno = ?";  
325     try {  
326         PreparedStatement ps = con.prepareStatement(sql);  
327         ps.setInt(1, codigo);  
328         ResultSet rs = ps.executeQuery();  
329         rs.next();  
330         txtCodigo.setText(rs.getString("cod_aluno"));  
331         txtNome.setText(rs.getString("nome"));  
332         txtIdade.setText(rs.getString("idade"));  
333         txtEmail.setText(rs.getString("email"));  
334         txtTelefone.setText(rs.getString("telefone"));  
335     } catch (SQLException e) {  
336         JOptionPane.showMessageDialog(null, "Erro: " + e.getMessage(), "Erro", JOptionPane.ERROR_MESSAGE);  
337     }  
338 }
```

- a linha 233 traz um objeto da classe **RecordSet**
 - é um objeto capaz de armazenar e fornecer acesso aos possíveis múltiplos registros
- uma instrução SQL de consulta pode retorna nenhum ou muito registros
 - será sempre nenhum (0) ou um (1) dado o fato que o critério da pesquisa é o código do aluno que na tabela é uma chave primaria
- o objeto de **RecordSet** recebe a execução da instrução SQL que nesse caso por ser tratar de uma instrução **SELECT**
- deve ser executada com a chamada do método **executeQuery** do objeto de **PreapredStatement** e não com o método **executeUpdate** como foi feito anteriormente

- trabalha com o conceito de cursores
 - é o que permite o acesso as possíveis múltiplas linhas de registro do retorno da consulta
 - o cursor começa sempre no índice 0 onde não existe nenhum registro
- para acessar um registro se faz necessário mover o cursor para o índice seguinte para tanto usamos o método **next** do objeto de **RecordSet** que é feito na linha 235

Análise do código

- o código das linhas 237 até a linha 242 configuram os valores da consulta nos campos do formulário através do método **setText** da classe **TextField** os valores são passados através da chamada do método **getString** do objeto da classe **RecordSet**

Botão Limpar

```
309 private void btnLimparActionPerformed(java.awt.event.ActionEvent evt) {  
310     txtCodigo.setText("");  
311     txtNome.setText("");  
312     txtIdade.setText("");  
313     txtEmail.setText("");  
314     txtTelefone.setText("");  
315 }
```

```
274
275 private void btnExcluirActionPerformed(java.awt.event.ActionEvent evt) {
276     int codigo = Integer.parseInt(txtCodigo.getText());
277     String sql = "delete from aluno where cod_aluno = ?";
278     try {
279         PreparedStatement ps = con.prepareStatement(sql);
280         ps.setInt(1, codigo);
281         ps.executeUpdate();
282         JOptionPane.showMessageDialog(null, "Cadastrado Excluido com sucesso!", "Exclusão", JOptionPane.INFORMATION_MESSAGE);
283         txtCodigo.setText("");
284         txtNome.setText("");
285         txtIdade.setText("");
286         txtEmail.setText("");
287         txtTelefone.setText("");
288
289     } catch (SQLException e) {
290         JOptionPane.showMessageDialog(null, "Erro: " + e.getMessage(), "Erro", JOptionPane.ERROR_MESSAGE);
291     }
292 }
293
```

Botão SAIR

```
298  
299 private void btnSairActionPerformed(java.awt.event.ActionEvent evt) {  
300     Conexao.fechaConexao(con);  
301     System.exit(0);  
302 }  
303
```