

**FIAP GRADUAÇÃO**

# Aula de hoje

## Conexão com o banco de dados

- Criação de uma classe para conexão
- Criação de instruções de inserção, remoção, atualização e busca de dados
- SQL INJECTION

## Para acompanhar a aula

- TODO O CÓDIGO DESTA APOSTILA ESTÁ DISPONÍVEL NO PROJETO TestandoBanco presente no arquivo JAVA\_BANCO\_1(CRUD COMPLETO E VUNERAVEL).zip

# 1. Criação de uma classe para a conexão

## 1.1 Criação de uma classe

- Seguindo os princípios de OO que estudamos desde o início do ano, a melhor alternativa que temos é criar uma CLASSE para as operações com o banco de dados
- Sem nunca ter programado esse tipo de conexão, quais são os atributos (informações) que a nossa classe deve precisar?

## 1.1 Criação de uma classe

### Atributos

- Endereço do banco de dados
- Porta para acesso ao servidor do banco
- Usuário para acesso ao banco
- Senha para acesso ao banco
- Precisaremos também de um atributo que seja do tipo CONNECTION, para que se estabeleça a conexão em si.

## 1.1 Criação de uma classe

E os métodos?

- Gets e Sets para os atributos anteriores
- abrirConexao
- fecharConexao

## Conexao

-servidor:String

-porta:String

-senha:String

-usuario:String

+conexao: Connection

+Conexao()

+getServidor():String

+getPorta():String

+getUsuario():String

+getSenha():String

+setSevidor(servidor:String)

+setPorta(porta:String)

+setSenha(senha:String)

+setUsuario(usuario:String)

+abrirConexao()

+fecharConexao()

```
12 public class Conexao {  
13     /*Atributos básicos para a conexão com o banco  
14     * */  
15     private String servidor;  
16     private String porta;  
17     private String senha;  
18     private String usuario;  
19  
20     public Connection conexao;
```



```
public String getServidor() {  
    return servidor;  
}  
  
public void setServidor(String servidor) {  
    this.servidor = servidor;  
}  
  
public String getPorta() {  
    return porta;  
}  
  
public void setPorta(String porta) {  
    this.porta = porta;  
}  
  
public String getSenha() {  
    return senha;  
}  
  
public void setSenha(String senha) {  
    this.senha = senha;  
}  
  
public String getUsuario() {  
    return usuario;  
}  
  
public void setUsuario(String usuario) {  
    this.usuario = usuario;  
}
```

■ Até aí, nenhuma novidade!



## 1.2 Abrindo a conexão com o Banco

E os métodos?

- Montar a “string de conexão”, que nada mais é do que um texto com informações sobre a conexão com o banco
- Registrar um Driver de conexão. Um driver de conexão é um arquivo que contém as informações necessárias para lidar com um banco de dados específico. Para o Oracle SQL o driver é um, para o MySQL é outro.
- Usar um gerenciador de Drivers para estabelecer a conexão

```
public void abreConexao(){
    String url="jdbc:oracle:thin:@"+servidor+": "+porta+":orcl";
    try{
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        conexao = DriverManager.getConnection(url, usuario, senha);
    }catch (Exception e) {
        e.printStackTrace();
    }
}
```

Seguindo o modelo disponível no site da Oracle (que é o banco que usaremos), montamos a string de conexão e armazenamos em um objeto do tipo String.

Dentro dos parênteses é preciso indicar qual Drive de conexão está sendo usado!

```
public void abreConexao(){  
    String url="jdbc:oracle:thin:@"+servidor+": "+porta+":orcl";  
    try{  
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  
        conexao = DriverManager.getConnection(url, usuario, senha);  
    }catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Por fim, preenchemos o objeto "conn" com a conexão obtida através do método getConnection do DriverManager

Temos que usar o DriverManager para registrar o driver que cuida de bancos Oracle. Para o DriverManager funcionar, é preciso importar a classe **java.sql.DriverManager**;



**PARA TUDO!**

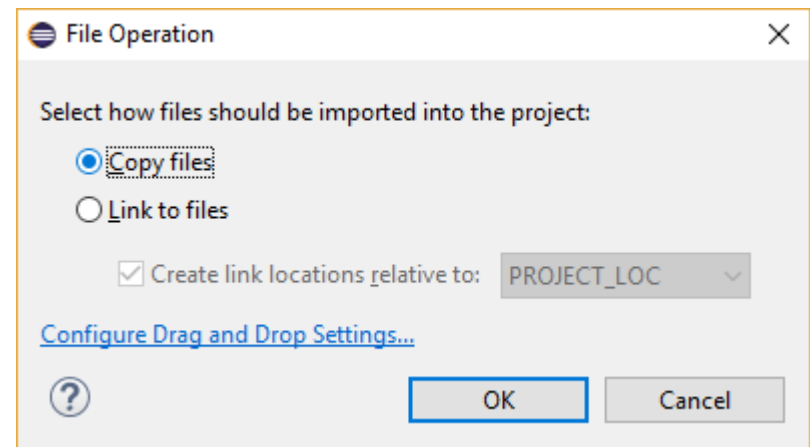
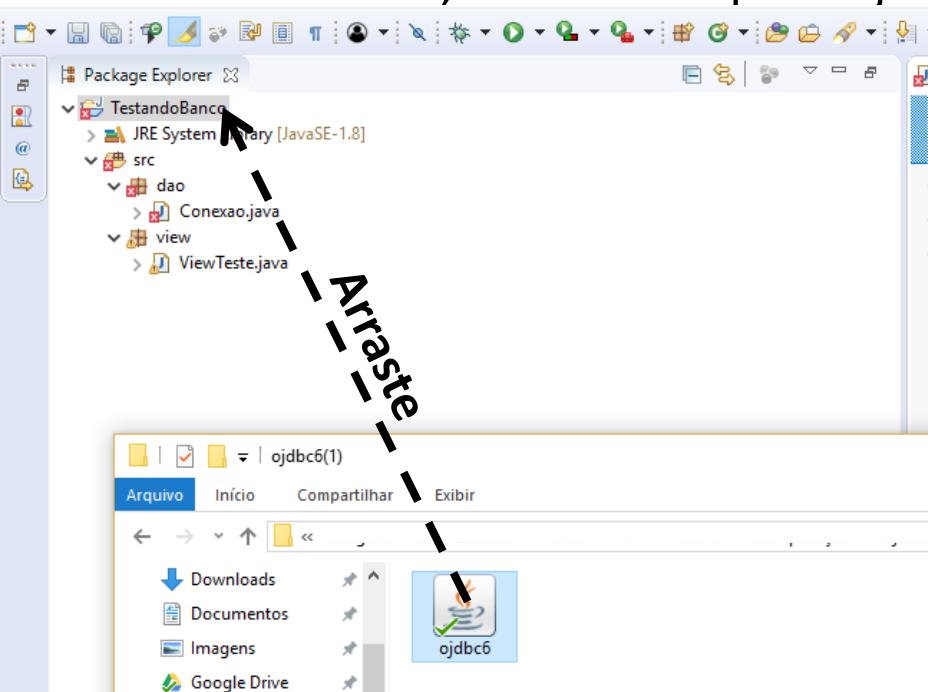
**O que era esse Try?  
Por que está usando esse  
`e.printStackTrace()`?  
Por que meu driver de  
conexão está apontando  
erro?**

## Sobre o Try

- O Try serve para fazer **TRATAMENTO DE ERRO**, ou seja, ao invés do seu aplicativo travar a máquina virtual java, ele exibe uma mensagem ou toma alguma medida definida por você.
- O bloco **CATCH** entra em ação quando ocorre um erro. O `e.printStackTrace()` serve para printar esse erro no console.

# Por que meu Driver de Conexão está com erro?

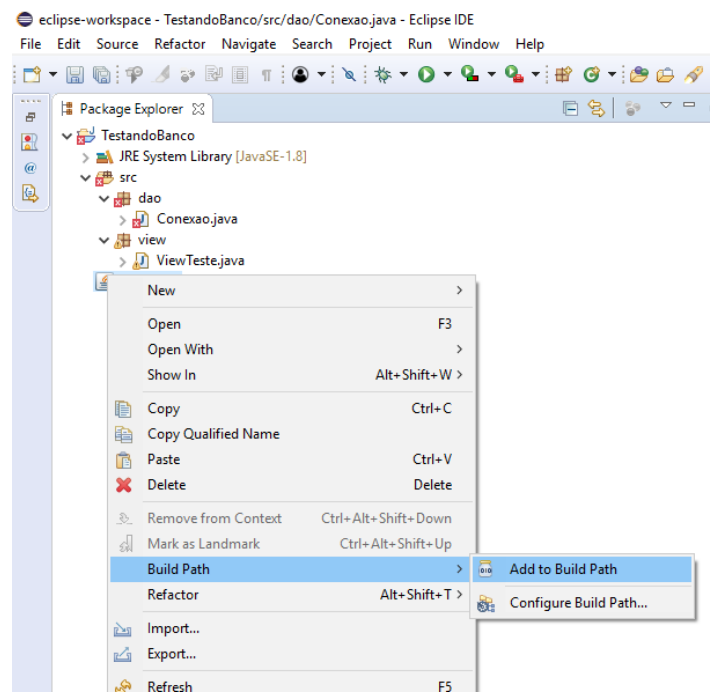
- Porque nós não incluímos a biblioteca de driver de conexão da Oracle no nosso projeto!
- Para isso, arraste o arquivo ojdbc6.jar para dentro do seu projeto





# Por que meu Driver de Conexão está com erro?

- Agora é necessário incluir a biblioteca ao Build Path do projeto
- Para isso, clique sobre o arquivo ojdbc6.jar com o botão direito, selecione a opção “Buid Path” e “Add to Build Path”



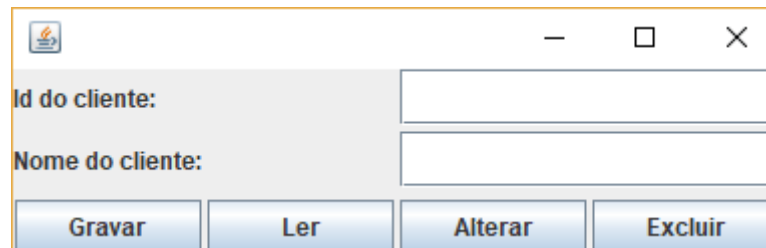
Já que o nosso atributo conn (que é do tipo Connection) armazena a conexão ativa com o banco, para fechar a conexão basta usar o método close().

```
public void fechaConexao () {  
    try{  
        conexao.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

## 2. CRUD

## 2.1 CRUD

- Chamamos de **CRUD** um sistema capaz de **CADASTRAR** (create), **LER** (read), **ATUALIZAR** (update) e **REMOVER** (delete) dados de um banco de dados relacional.



The image shows a web application window with a title bar containing a small icon and standard minimize, maximize, and close buttons. The main content area has two input fields: 'Id do cliente:' and 'Nome do cliente:'. Below these fields are four buttons: 'Gravar', 'Ler', 'Alterar', and 'Excluir'.

## 2.1 CRUD

- Vamos modificar a nossa classe para ser capaz de executar um comando de CADASTRO, o famoso INSERT.
- Para isso, precisamos de um objeto capaz de executar instruções SQL.
- A classe mais *preguiçosa e vulnerável* para fazer isso é a classe **Statement**. Como muitos programadores insistem em usá-la, vamos estudar primeiro essa prática INADEQUADA.

## 2.1 CRUD

- Primeiro será necessário criar um atributo (ou um objeto, se preferir fazer dessa forma) do tipo **Statement** da classe `java.sql.Statement`:

```
21      public Statement comando;
```

- Nesse atributo poderemos armazenar instruções SQL e executá-las no banco.

## 2.1 CRUD

### Inserir

```
67 public void inserir(String nome) {  
68     try {  
69         /*Criamos o Statement com o objeto comando. Caso  
tudo seja executado corretamente,  
70         * retornamos true. Caso ocorra um erro, retornamos  
false*/  
71         comando = conexao.createStatement();  
72         comando.execute("Insert into TESTE (nome_cliente)  
values ('" + nome + "')");  
73     } catch (SQLException erro) {  
74         erro.printStackTrace();  
75     }  
76 }  
77 }  
78 }
```

## 2.1 CRUD

### Inserir

Como nosso banco de exemplo contém uma tabela chamada TESTE com apenas 2 campos (nome\_cliente e id\_cliente – que é auto numerável), criaremos o método Inserir pedindo apenas o nome do cliente

O bloco Try faz o tratamento de erros

Criamos o nosso Statement usando a conexão ativa, para que ele seja, posteriormente, executado nessa conexão

```
67 public void inserir(String nome) {  
68     try {  
69         /*Criamos o Statement  
tudo seja executado corretamente  
70         * retornamos true. Caso ocorra um erro, retornamos  
false*/  
71         comando = conexao.createStatement();  
72         comando.execute("Insert into TESTE (nome_cliente)  
values ('" + nome + "')");  
73     } catch (SQLException erro) {  
74         erro.printStackTrace();  
75     }  
76 }  
77 }  
78 }
```

Executamos a instrução Insert, concatenando a variável *nome* no lugar onde seria inserido o valor do campo nome\_cliente. Essa técnica deixa o sistema vulnerável a ataques SQLInjection



## 2.1 CRUD

### Atualizar

```
87 public void alterar(int id, String nome) {  
88     try {  
89         /*Criamos o Statement com o objeto comando. Caso  
tudo seja executado corretamente,  
90         * retornamos true. Caso ocorra um erro, retornamos  
false*/  
91         comando = conexao.createStatement();  
92         comando.execute("Update TESTE set nome_cliente='" +  
nome + "' where id_cliente=" + id);  
93  
94     } catch (SQLException erro) {  
95         erro.printStackTrace();  
96  
97     }  
98 }  
99
```

## 2.1 CRUD

### Atualizar

Como nosso banco de exemplo contém uma tabela chamada TESTE com apenas 2 campos (nome\_cliente e id\_cliente – que é auto numerável), criaremos o método atualizar pedindo o nome e o id

O bloco Try faz o tratamento de erros

Criamos o nosso Statement usando a conexão ativa, para que ele seja, posteriormente, executado nessa conexão

```
87 public void alterar(int id, String nome) {  
88     try {  
89         /*Criamos o Statement  
90         tudo seja executado corretamente,  
91         * retornamos true. Caso contrário, retornamos  
92         false*/  
93         comando = conexao.createStatement();  
94         comando.execute("Update TESTE set nome_cliente='" +  
95         nome + "' where id_cliente=" + id);  
96     } catch (SQLException erro) {  
97         erro.printStackTrace();  
98     }  
99 }
```

Executamos a instrução Update, concatenando a variável *nome* no lugar onde seria inserido o valor do campo nome\_cliente e a variável *id* onde seria inserido o id. Essa técnica deixa o sistema vulnerável a ataques SQLInjection

## 2.1 CRUD

### Remover

```
106 public void remover(int id) {  
107     try {  
108         /*Criamos o Statement com o objeto comando. Caso  
109         tudo seja executado corretamente,  
110         * retornamos true. Caso ocorra um erro, retornamos  
111         false*/  
112         comando = conexao.createStatement();  
113         comando.execute("Delete from TESTE where  
114         id_cliente=" + id);  
115     } catch (SQLException erro) {  
116         erro.printStackTrace();  
117     }  
118 }
```

## 2.1 CRUD

### Remover

Como nosso banco de exemplo contém uma tabela chamada TESTE com apenas 2 campos (nome\_cliente e id\_cliente – que é auto numerável), pediremos apenas o id para a remoção

O bloco Try faz o tratamento de erros

Criamos o nosso Statement usando a conexão ativa, para que ele seja, posteriormente, executado nessa conexão

```
106 public void remover(int id) {  
107     try {  
108         /*Criamos o Statement  
tudo seja executado corretamente  
109         * retornamos true. Caso ocorra um erro, retornamos  
false*/  
110         comando = conexao.createStatement();  
111         comando.execute("Delete from TESTE where  
id_cliente=" + id);  
112     } catch (SQLException erro) {  
113         erro.printStackTrace();  
114     }  
115 }  
116 }  
117 }
```

Executamos a instrução Delete, concatenando a variável id onde seria inserido o id. Essa técnica deixa o sistema vulnerável a ataques SQLInjection

## 2.1 CRUD

- Para buscar algo em uma tabela do banco, temos que ter em mente que o comando SQL *select* retorna um dado (uma ou mais linhas). Dessa forma, precisaremos de um objeto para armazenar os dados retornados.
- Esse objeto é um ResultSet. Ele é capaz de armazenar as colunas da linha que for retornada do banco.

## 2.1 CRUD

- Suponha que o select retornou 10 linhas do banco, com as colunas ID\_CLIENTE e NOME\_CLIENTE.
- Cada vez que o MÉTODO next() for executado no ResultSet, uma nova linha será selecionada (dentro das 10 que foram retornadas)
- Para obter o dado da primeira coluna (que é do tipo inteiro), usamos o método `getInteger(1)` do ResultSet. *getInteger* porque o campo é inteiro e 1 porque é a primeira coluna.
- O nome, portanto, seria obtido com um `getString(2)`.

## 2.1 CRUD

### Buscar

```
126 public void buscar(int id) {
127     try {
128         /*Criamos o Statement com o objeto comando. Caso
129         tudo seja executado corretamente,
130         exibimos um JOptionPane com o nome do cliente*/
131         ResultSet rs;
132         comando = conexao.createStatement();
133         /*O ResultSet armazena cada uma das colunas da
134         tupla resultante do select*/
135         rs = comando.executeQuery("Select nome_cliente
136         from TESTE where id_cliente=" + id);
137         rs.next();
138         JOptionPane.showMessageDialog(null,
139         rs.getString(1));
140
141     } catch (SQLException erro) {
142         erro.printStackTrace();
143     }
144 }
```

## 2.1 CRUD

### Buscar

```
126 public void buscar(int id)
127     try {
128         /*Criamos o Statement e fazemos com que
129         tudo seja executado corretamente,
130         exibimos um JOptionPane.showMessageDialog
131         ResultSet rs;
132         comando = conexao.createStatement();
133         /*O ResultSet armazena o resultado da
134         tupla resultante do select*/
135         rs = comando.executeQuery("Select nome cliente
136         from TESTE where id_cliente=" + id);
137         rs.next();
138         JOptionPane.showMessageDialog(null,
139         rs.getString(1));
140     } catch (SQLException erro) {
141         erro.printStackTrace();
142     }
143 }
```

O método buscar pedirá apenas o id.  
Posteriormente pode se criar um outro método  
que busque pelo nome.

O bloco Try faz o tratamento de erros

Criamos o ResultSet para armazenar o  
resultado

Criamos o Statement na conexão ativa

Executamos a busca e armazenamos o  
resultado no ResultSet

Usamos o método next para irmos para a  
primeira linha obtida

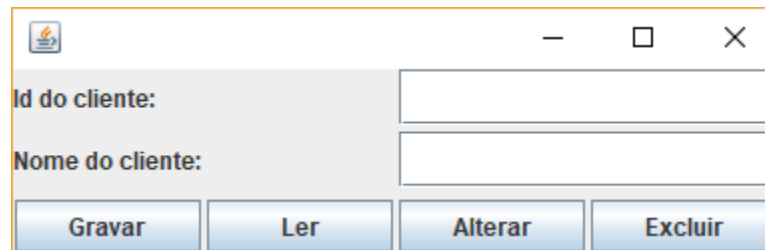
Exibimos o nome do cliente usando o método  
getString(1)



### 3. Testando a classe Conexão

## 3.1 Testando a classe

- Vamos usar a classe “ViewTeste” para podermos testar as capacidades da classe Conexao.
- Essa classe contém um formulário como o abaixo e foi feita usando todas as técnicas estudadas no primeiro semestre.



The image shows a screenshot of a Java Swing window titled "ViewTeste". The window contains a form with two text input fields. The first field is labeled "Id do cliente:" and the second field is labeled "Nome do cliente:". Below the input fields, there are four buttons: "Gravar", "Ler", "Alterar", and "Excluir". The window has a standard title bar with a minimize button, a maximize button, and a close button.

```
public class ViewTeste extends JFrame implements ActionListener{
```

```
    private JPanel painelFormulario;  
    private JLabel lblIdCliente;  
    private JTextField txtIdCliente;  
    private JLabel lblNomeCliente;  
    private JTextField txtNomeCliente;
```

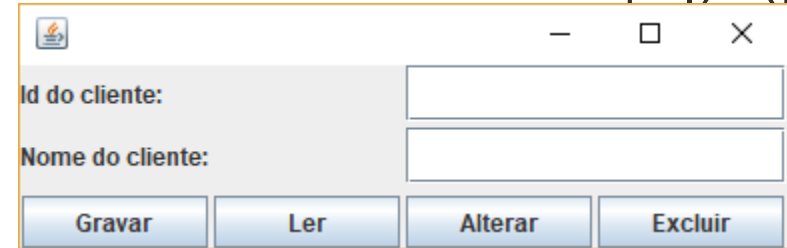
```
    private JPanel painelBotoes;  
    private JButton btnGravar;  
    private JButton btnLer;  
    private JButton btnAlterar;  
    private JButton btnExcluir;
```

```
    public ViewTeste() {  
        painelFormulario = new JPanel(new GridLayout(2,4,3,3));  
        lblIdCliente = new JLabel("Id do cliente:");  
        txtIdCliente = new JTextField(10);  
        lblNomeCliente = new JLabel("Nome do cliente:");  
        txtNomeCliente = new JTextField(10);  
  
        painelFormulario.add(lblIdCliente);  
        painelFormulario.add(txtIdCliente);  
        painelFormulario.add(lblNomeCliente);  
        painelFormulario.add(txtNomeCliente);  
  
        painelBotoes = new JPanel(new GridLayout(1,4,3,3));  
        btnGravar = new JButton("Gravar");  
        btnLer = new JButton("Ler");  
        btnAlterar = new JButton("Alterar");  
        btnExcluir = new JButton("Excluir");  
  
        painelBotoes.add(btnGravar);  
        painelBotoes.add(btnLer);  
        painelBotoes.add(btnAlterar);  
        painelBotoes.add(btnExcluir);
```

```
btnGravar.addActionListener(this);  
btnLer.addActionListener(this);  
btnAlterar.addActionListener(this);  
btnExcluir.addActionListener(this);  
  
setLayout(new BorderLayout(5,5));  
add(painelFormulario, BorderLayout.CENTER);  
add(painelBotoes, BorderLayout.SOUTH);  
}
```

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    ViewTeste janela = new ViewTeste();  
    janela.setSize(400,130);  
    janela.setDefaultCloseOperation(EXIT_ON_CLOSE);  
    janela.show();  
}
```

## 3.1 Testando a classe



The screenshot shows a Java Swing window with a title bar containing a small icon and standard window controls (minimize, maximize, close). The window contains a form with two text input fields. The first field is labeled 'Id do cliente:' and the second is labeled 'Nome do cliente:'. Below the input fields, there are four buttons arranged horizontally: 'Gravar', 'Ler', 'Alterar', and 'Excluir'.

- A chamada dos métodos será feita dentro do método `actionPerformed` da classe `ViewTeste` (também conforme estudado semestre passado) e obedecerá à sequência:
- Criar um objeto do tipo conexão
- Abrir a conexão
- Executar o método correspondente (Inserir, Ler, Alterar ou Excluir)
- Fechar a conexão

## Inserir

```
86     public void actionPerformed(ActionEvent event) {
87         if (event.getSource()==btnGravar) {
88             /*Aqui vamos inserir o que vai ocorrer
89              * quando o botão GRAVAR for clicado*/
90             Conexao banquinho = new Conexao();
91             banquinho.abreConexao();
92             banquinho.inserir(txtNomeCliente.getText());
93             JOptionPane.showMessageDialog(null, "O Cliente foi inserido
com sucesso!");
94             banquinho.fechaConexao();
95
96         }else {
```

```
96         }else {
97             if (event.getSource()==btnLer) {
98                 /*Aqui vamos inserir o que vai ocorrer
99                 * quando o botão LER for clicado*/
100
101                 if(txtIdCliente.getText().isEmpty()) {
102                     if(txtNomeCliente.getText().isEmpty()) {
103                         JOptionPane.showMessageDialog(null, "Insira um
id ou nome do cliente para buscar!");
104                     }
105                     else {
106                         Conexao banquinho = new Conexao();
107                         banquinho.abreConexao();
108                         banquinho.buscar(txtNomeCliente.getText());
109                         banquinho.fechaConexao();
110                     }
111
112                 }else {
113                     Conexao banquinho = new Conexao();
114                     banquinho.abreConexao();
115                     banquinho.buscar
(Integer.parseInt(txtIdCliente.getText()));
116                     banquinho.fechaConexao();
117                 }
118
119
120             }else {
```



## Alterar

```
120         }else {
121             if (event.getSource()==btnAlterar) {
122                 /*Aqui vamos inserir o que vai ocorrer
123                 * quando o botão ALTERAR for clicado*/
124                 if(txtIdCliente.getText().isEmpty()) {
125                     JOptionPane.showMessageDialog(null, "Insira um
id para alterar o dado!");
126
127                     }else {
128                         Conexao banquinho = new Conexao();
129                         banquinho.abreConexao();
130                         banquinho.alterar
(Integer.parseInt(txtIdCliente.getText()), txtNomeCliente.getText());
131                         JOptionPane.showMessageDialog(null, "O Cliente
foi alterado com sucesso!");
132                         banquinho.fechaConexao();
133                     }
134                 }else {
```

## Excluir

```
134         }else {
135             if (event.getSource()==btnExcluir) {
136                 /*Aqui vamos inserir o que vai ocorrer
137                 * quando o botão EXCLUIR for clicado*/
138
139                 if(txtIdCliente.getText().isEmpty()) {
140                     JOptionPane.showMessageDialog(null, "Insira
141 um id para remover o dado!");
142
143                 }else {
144                     Conexao banquinho = new Conexao();
145                     banquinho.abreConexao();
146                     banquinho.remover
147 (Integer.parseInt(txtIdCliente.getText()));
148                     JOptionPane.showMessageDialog(null, "O
149 Cliente foi removido com sucesso!");
150                     banquinho.fechaConexao();
151                 }
152             }
```

## 3.1 Testando a classe

- Na classe `Conexao.java` que criamos inicialmente, vamos definir os parâmetros para a conexão logo no construtor

```
public Conexao(){  
    String servidor="oracle.fiap.com.br";  
    String porta="1521";  
    String senha="SUA SENHA";  
    String usuario="SEU USUARIO";  
}
```

## 2.1 Testando a conexão

- Lembre-se: por termos usado a classe Statement, nosso projeto está vulnerável ao SQLInjection

## 4. SQLInjection

## 4.1 SQLInjection

- O SQLInjection é um dos mais infames e danosos ataques que um sistema pode sofrer.
- Ele consiste em inserir uma INSTRUÇÃO SQL em um campo destinado aos dados do usuário final.

- 
- The image displays a collection of 15 login screen design mockups, arranged in a grid. The designs are diverse, featuring various color schemes and layouts. Key elements visible include:
- Color Schemes:** Blue, purple, white, red, green, and dark grey.
  - Layouts:** Some are centered, some are side-by-side, and some feature large background images or illustrations.
  - Elements:** Email/password input fields, "Login" or "Sign In" buttons, "Forgot Password" links, social media links, and branding elements like logos and company names.
  - Branding:** Logos for "MOTION", "Mayo Clinic", "H5 Anggal", and "WordPress" are visible.
  - Interactions:** Some designs show hover states or active states for buttons and links.



## 4.1 SQLInjection

- E imagine que o programador tenha criado uma instrução da seguinte forma:
- `"Select * from usuarios where username=" + nome + " and password=" + senha + """`

## 4.1 SQLInjection

- Agora imagine que ao preencher as caixas de usuário e senha, o atacante insira:
- Nome de usuário: nem\_ligo
- Senha: oi' OR '1' = '1

## 4.1 SQLInjection

- O comando SQL ficará contaminado da seguinte maneira:
- “Select \* from usuarios where username= ‘nemligo’ and password= ‘oi’ or ‘1’=‘1’”

## 4.1 SQLInjection

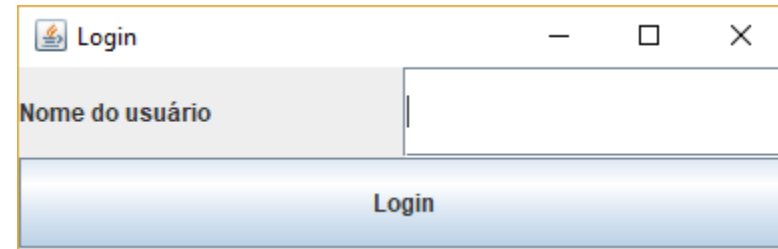
- Como a condição final é OR 1=1, o atacante obterá acesso ao sistema.
- Para exemplificar isso, vamos programar um método chamado loginVulnerável na nossa classe Conexão.

## 4.1 SQLInjection

- O método loginVulnerável pedirá apenas um nome de cliente cadastrado no banco. Caso o nome exista, retornará verdadeiro (sinalizando que o acesso foi permitido)

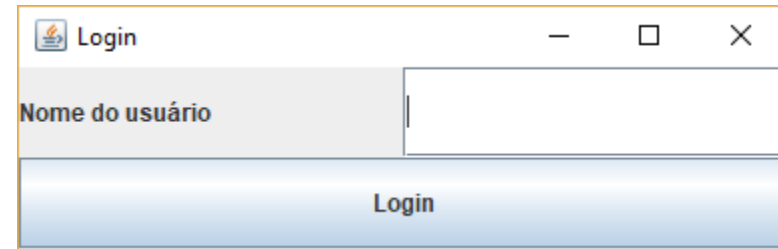
```
36 public boolean loginVulneravel(String nome) {
37     /*Muitos programas usam um select pelo nome de usuário e senha
para fazer
38     * um login, retornando verdadeiro caso o registro
correspondente
39     * seja encontrado. O nosso login vulnerável faz o mesmo, mas
usando o
40     * nome de cliente*/
41     try {
42         ResultSet rs;
43         comando = conexao.createStatement();
44         /*O ResultSet armazena cada uma das colunas da tupla
resultante do select*/
45         rs = comando.executeQuery("Select * from TESTE where
nome_cliente='" + nome + "'");
46         if(rs.next()) {
47             return true;
48         }else {
49             return false;
50         }
51
52
53     }catch(SQLException erro) {
54         erro.printStackTrace();
55         return false;
56
57     }
58 }
```

## 4.1 SQLInjection



- Para testar, usaremos a classe “ViewLoginVulnerável” que contém uma caixa para a digitação de um nome de usuário e permite acesso à tela ViewTeste apenas se o método de loginVulnerável retornar verdadeiro

## 4.1 SQLInjection

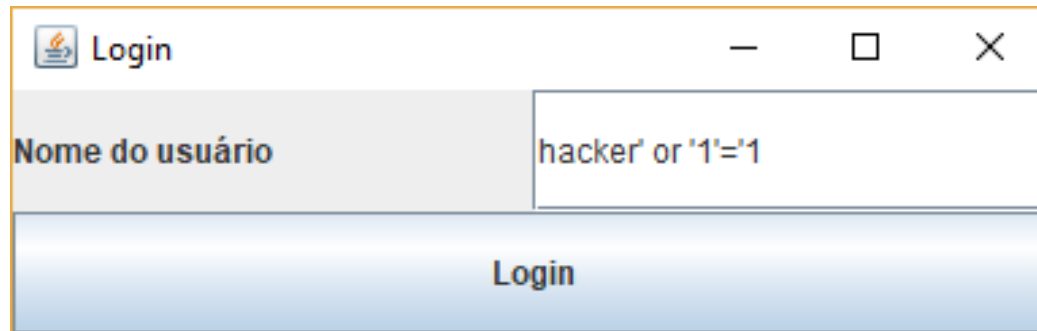


- Para testar, usaremos a classe “ViewLoginVulnerável” que contém uma caixa para a digitação de um nome de usuário e permite acesso à tela ViewTeste apenas se o método de loginVulnerável retornar verdadeiro



## 4.1 SQLInjection

- Ao digitar um usuário correto, o login deve ser liberado.
- Ao digitar um usuário incorreto, o login deve ser impedido.
- Ao digitar o sql injection abaixo...



The screenshot shows a web browser window with a title bar that says 'Login'. Inside the window, there is a form with a label 'Nome do usuário' (User Name) and a text input field. The input field contains the text 'hacker' or '1'='1'. Below the input field is a blue button labeled 'Login'.

## **5. Sobre o conteúdo exposto nessa apostila**

## ■ 5.1 String de Conexão

- Que tipo de informações a String de Conexão armazena?

## 5.2 Driver

- Para conectar o programa em Java ao banco Oracle, foi necessário registrar um Driver. Que arquivo contém esse driver e qual a sua função?

## 5.3 Try

- No momento de abrir ou fechar a conexão foi utilizado o bloco Try. Esse bloco também foi usado na hora de executar comandos no banco.
- Qual é a função do bloco Try...Catch?

## 5.4 Try

- No momento de abrir ou fechar a conexão foi utilizado o bloco Try. Esse bloco também foi usado na hora de executar comandos no banco.
- Qual é a função do bloco Try...Catch?

## ■ 5.5 CRUD

- O que é um sistema CRUD?

## 5.6 Statement

- Para que serve um objeto da classe Statement?
- Por que ele não é indicado?



## 5.6 Buscar

- Para buscar dados no banco, usamos um objeto do tipo Statement.
- Por que?

## ■ 5.7 SQLInjection

- O que é um ataque SQLInjection? Por que devemos tomar cuidado com ele?