

CONECTANDO NO MYSQL VIA NETBEANS

Agora vamos inserir dados no Banco de Dados. Para isso precisamos de 3 coisas:

- Netbeans instalado
- MySQL Community instalado
- MySQL Connector
- Código Exemplo para testar

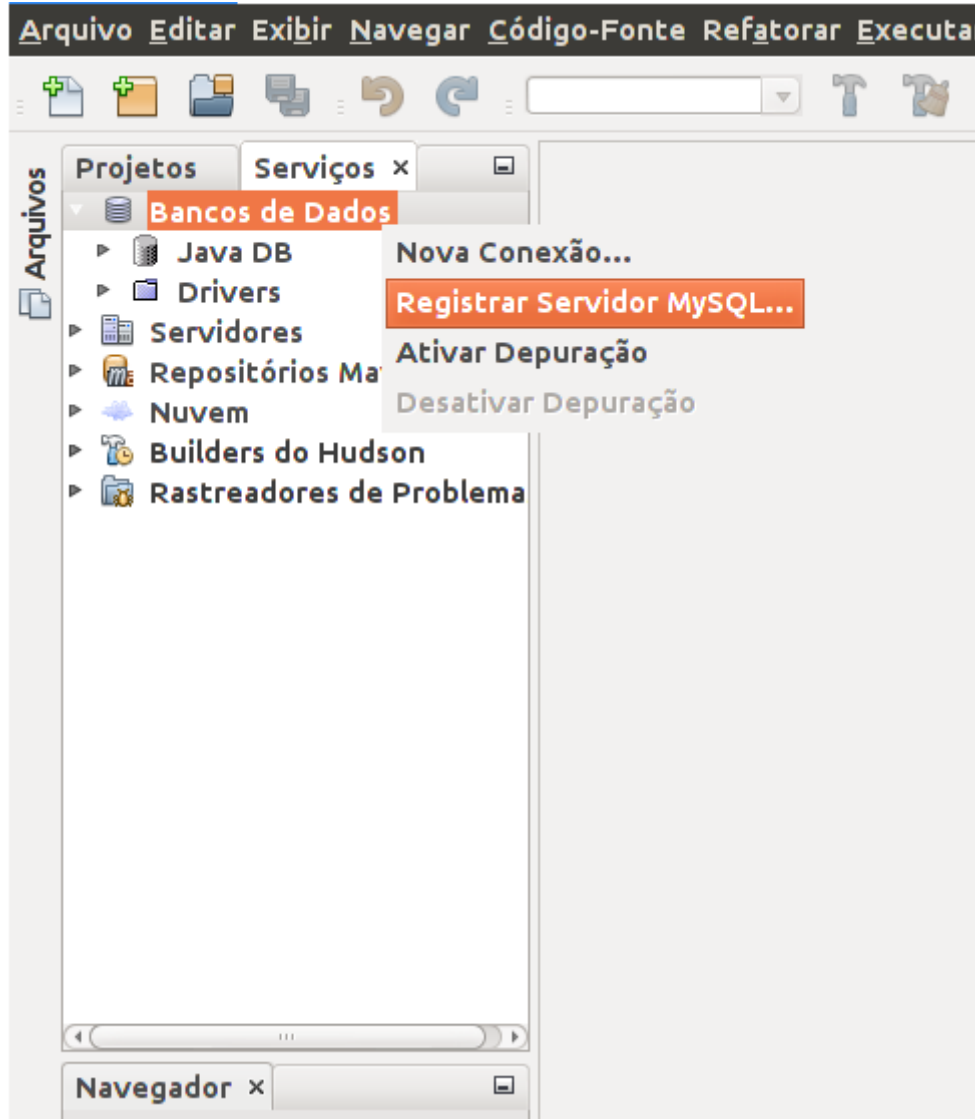
REGISTRANDO O MYSQL NO NETBEANS

Para registrar o MySQL no Netbeans precisamos abrir a guia "Serviços". Para abrir a guia Serviços vá no menu "Janela > Serviços".

Após abrir a guia serviços, clique com o botão direito do mouse na opção Banco de Dados; Em seguida clique na opção "Registrar servidor MySQL", conforme mostra a Figura a seguir.

Atenção: O registro do MySQL só precisa ser feito uma única vez no Netbeans. A partir da segunda vez só precisamos conectar o banco de dados.

GUIA SERVIÇOS



Em seguida coloque a senha do Banco de dados juntamente com o usuário(padão root) conforme Figura. A senha que você deve inserir é a senha de instalação do MySQL, a mesma que você utiliza para acessar o "MySQL line Command".

Obs.: Se o banco de dados não tiver senha, é só deixar o campo em branco.

Após inserir a senha, clique em "OK".

COLOCANDO A SENHA NO BANCO

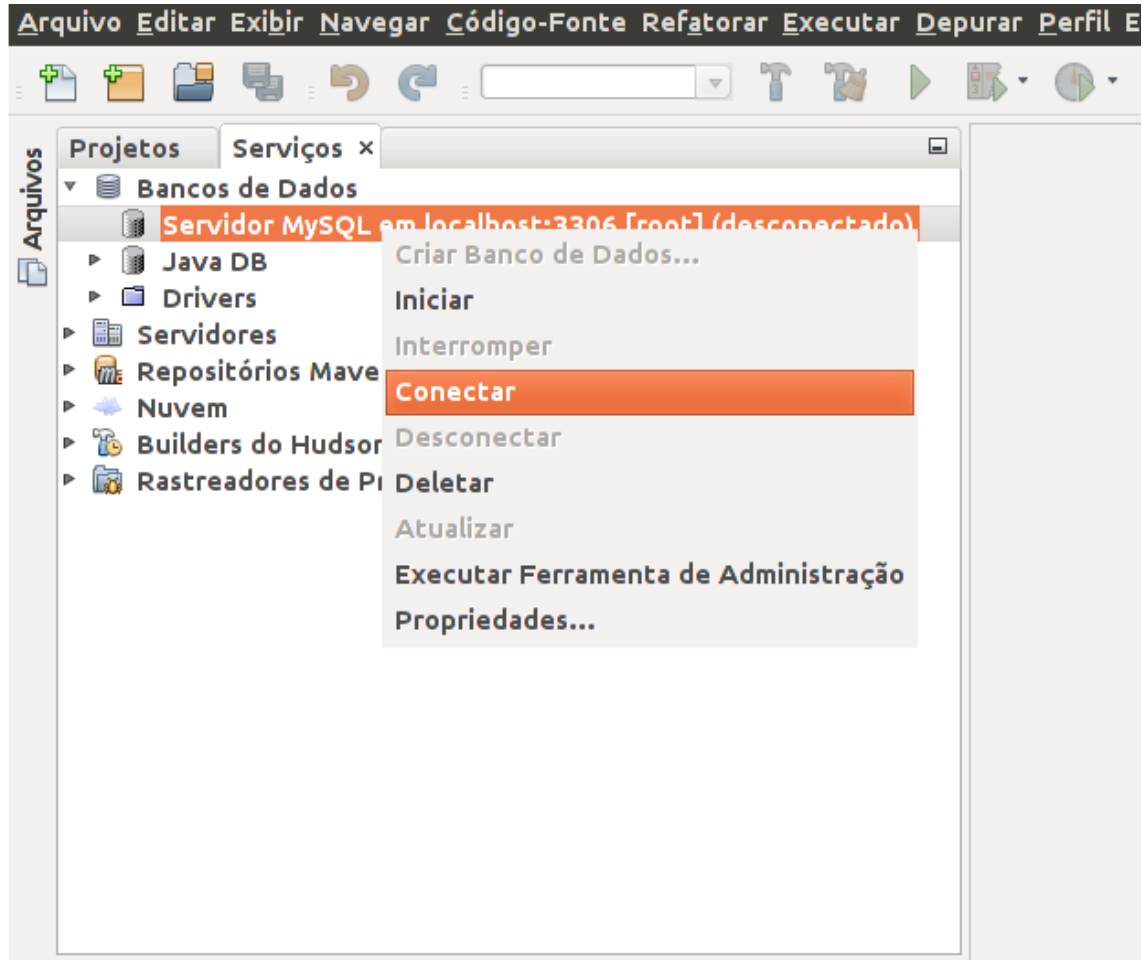
The image shows a Windows-style dialog box titled "Propriedades do Servidor MySQL". It has two tabs: "Propriedade Básicas" and "Propriedades de Admin", with the latter being the active tab. The dialog contains several input fields and a checkbox. The "Nome do Host do Servidor:" field is filled with "localhost". The "Número da Porta do Servidor:" field is filled with "3306". The "Nome de Usuário do Administrador:" field is filled with "root". The "Senha do Administrador:" field is empty and has a red border. Below this field is a checkbox labeled "Lembrar Senha" which is unchecked. To the right of the password field, there is a black rectangular button with the white text "Senha do administrador". At the bottom right of the dialog, there are three buttons: "OK" (orange), "Cancelar" (gray), and "Ajuda" (gray).

Nome do Host do Servidor:	localhost
Número da Porta do Servidor:	3306
Nome de Usuário do Administrador:	root
Senha do Administrador:	
<input type="checkbox"/> Lembrar Senha	Senha do administrador

OK Cancelar Ajuda

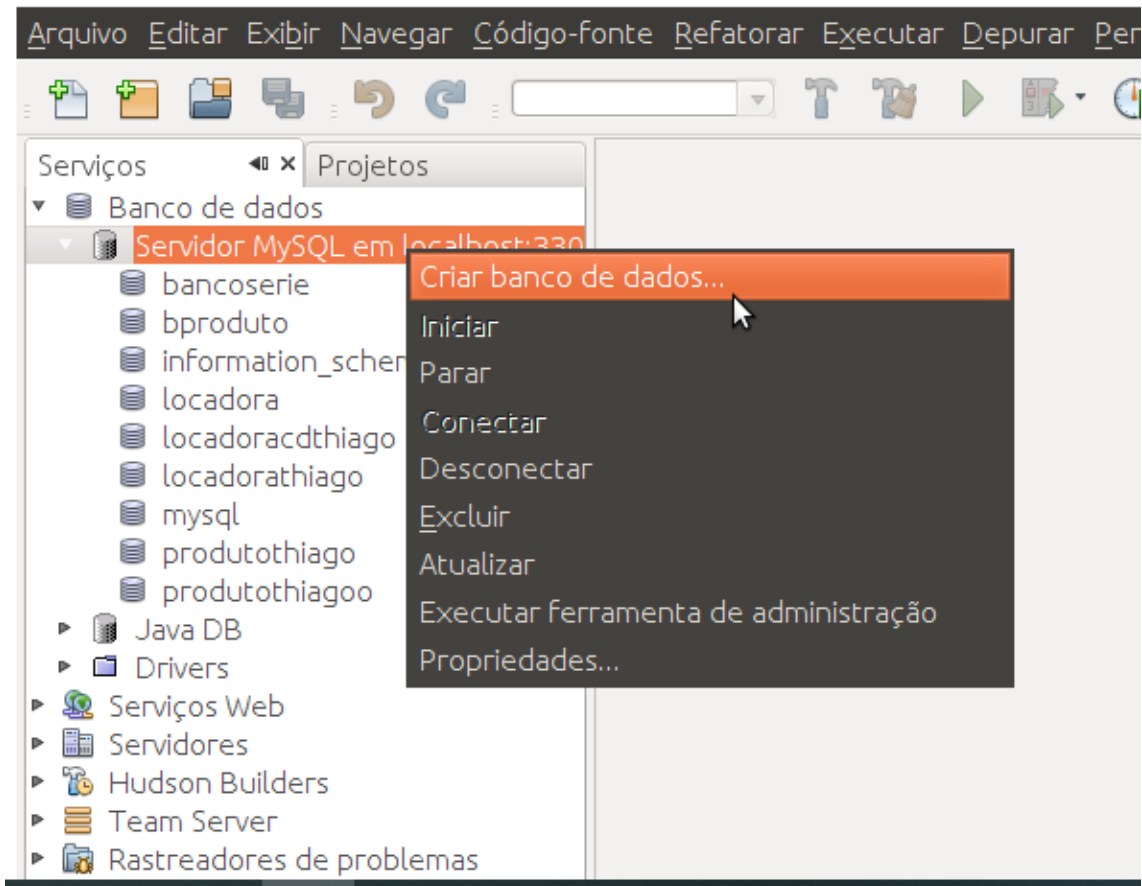
Agora que você já registrou o MySQL e colocou a senha surgirá a opção: "Servidor MySQL em localhost 3306 (desconectado)". Clique nessa opção com o botão direito, em seguida clique na opção "Conectar" conforme Figura.

CONECTANDO O BANCO NO NETBEANS



Agora que já conectamos e iniciamos o MySQL através do Netbeans, vamos criar um banco de dados. Para criar um banco de dados, clique novamente em "Servidor MySQL em localhost 3306...", em seguida escolha a opção "Criar banco de dados...", conforme Figura.

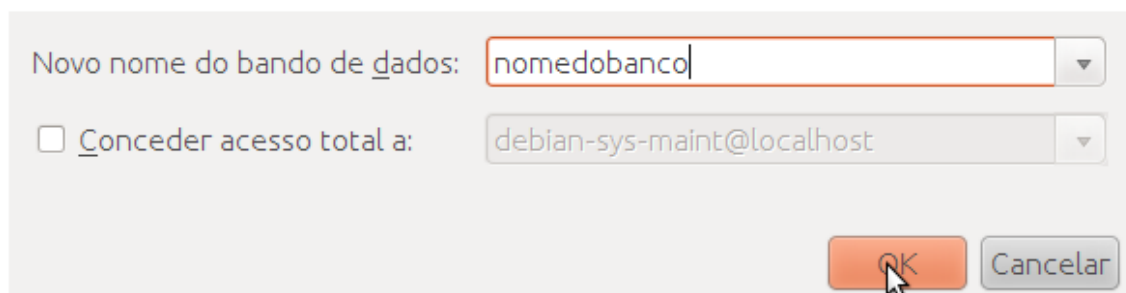
CRIANDO O BANCO DE DADOS



parecerá em seguida uma opção para inserir o nome que deseja para o Banco de dados, conforme Figura. Após digitar o nome do banco, clique em "OK".

Neste exemplo criei um banco de dados com o nome: "nomedobanco".

INSERINDO NOME NO BANCO



Observação: Lembre-se; para não ocorrer problemas futuros, sempre crie o database(banco de dados) com todas letras em caixa baixa(letras minúsculas).

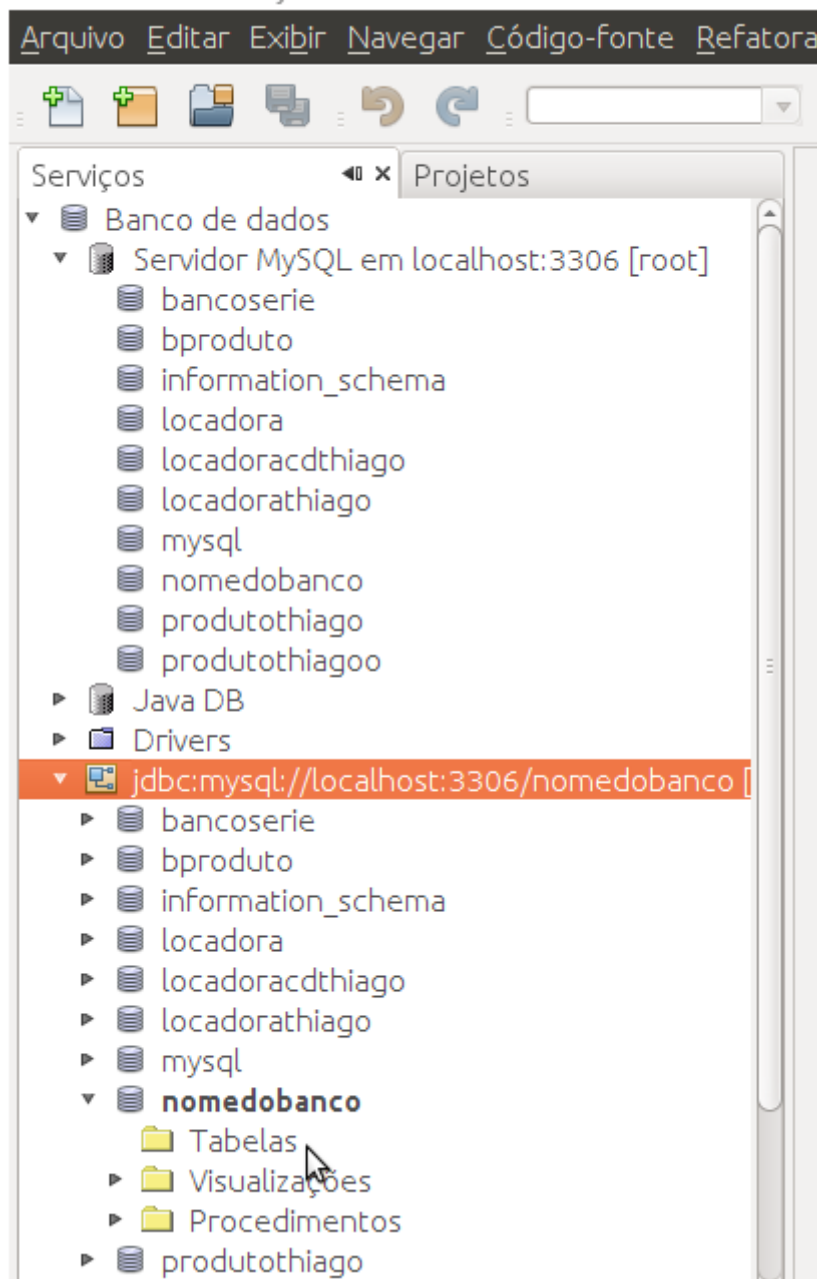
Agora note que aparecerá uma opção com o seguinte nome:

"jdbc:mysql://localhost:3306/nomedobanco"

Ao clicar nessa opção, será aberto o banco recém criado marcado em negrito. Clique no "+" no lado esquerdo dele para expandir e verá a opção "Tabelas". A próxima Figura ilustra a explicação.

Observação: No Windows aparecerá o sinal de "+" adição nas opções, já no Linux o sinal será de uma seta.

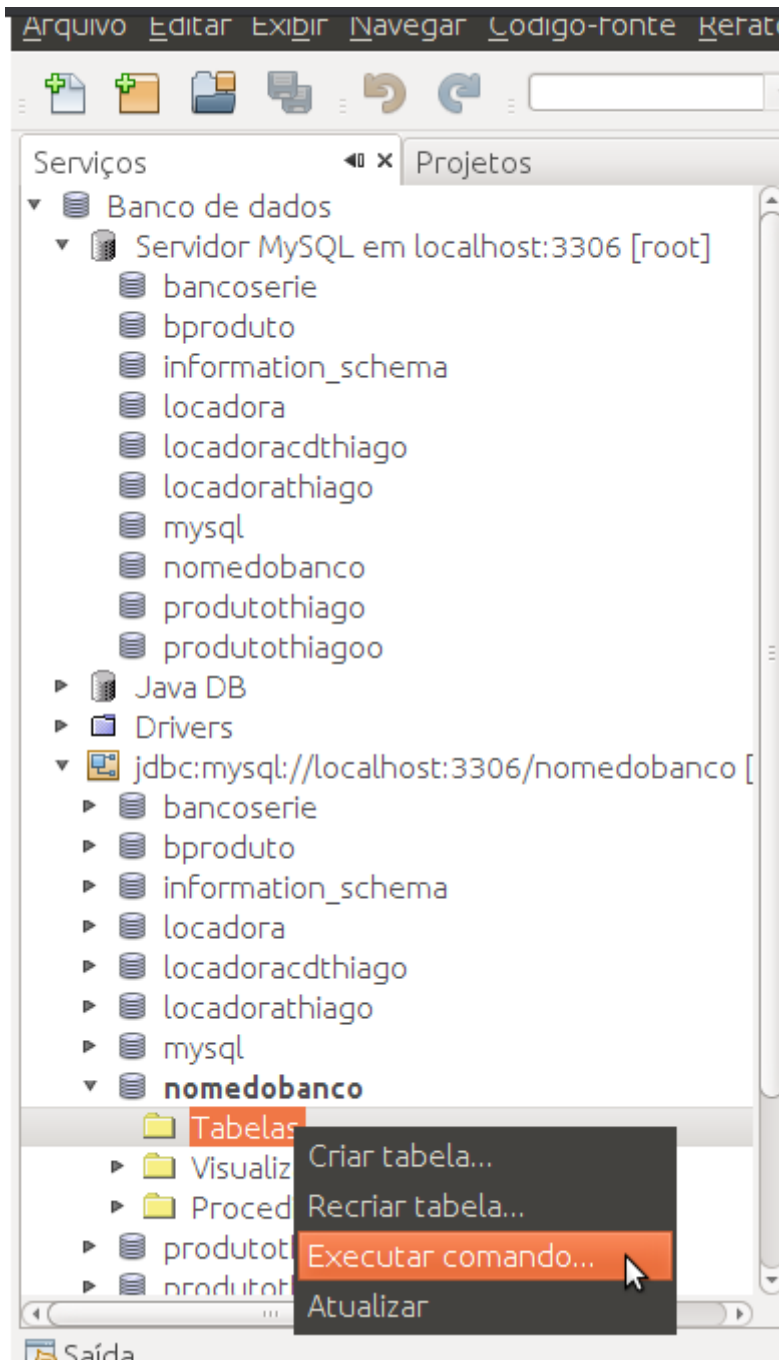
ACESSANDO A OPÇÃO TABELAS



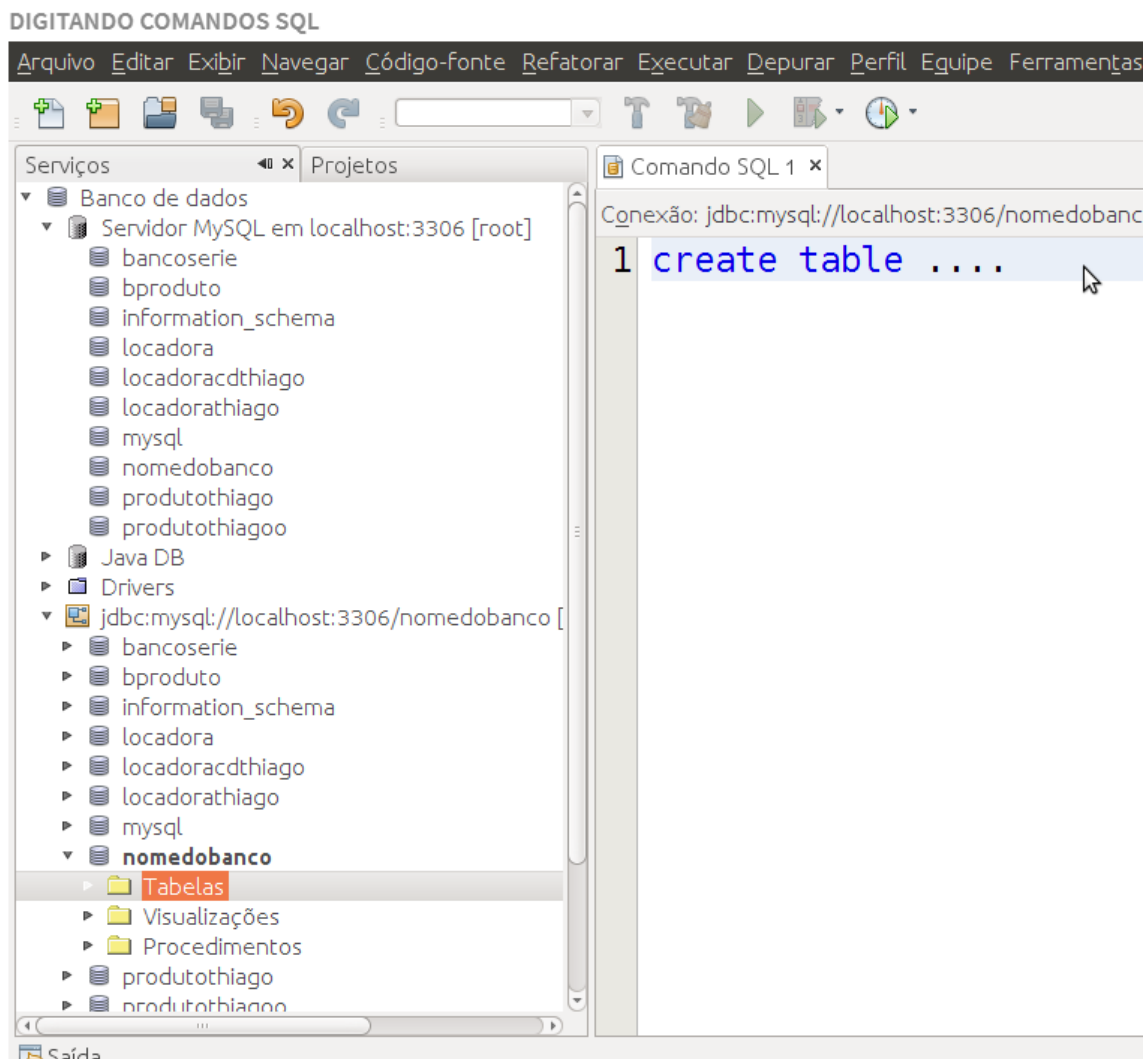
Agora vamos criar a nossa primeira tabela no banco. Para isso clicamos com o botão direito do mouse em cima da opção "Tabelas", então podemos optar em fazer de 2 formas diferentes, são elas:

- "Criar tabela" - criação feita através de um assistente
- "Executar Comando" - criação feita através de comandos SQL

A maneira elegida para a criação da tabela foi a opção "Executar Comando" conforme a Figura.



Aparecerá uma Guia com o nome de "Comando SQL1". Insira os comandos "SQL" que precisar conforme Figura.



Após digitar o comando para criar a tabela, você pode executar o comando de três formas diferentes, são elas:

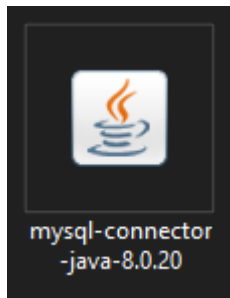
- 1ª Forma: Clicar no ícone banco de dados com uma seta verde.
- 2ª Forma: Clicar em cima da instrução digitada com o botão direito do mouse e escolher a opção "Rodar Instrução".
- 3ª Forma: CTRL + SHIFT + E

INSERINDO O .JAR

Para que o Java conecte no Banco de Dados precisamos adicionar ao projeto o "MySQL Connector". Eu utilizo a conexão via JDBC, logo preciso adicionar nas bibliotecas do projeto esse conector para que o Java realize a comunicação com o banco de dados MySQL.

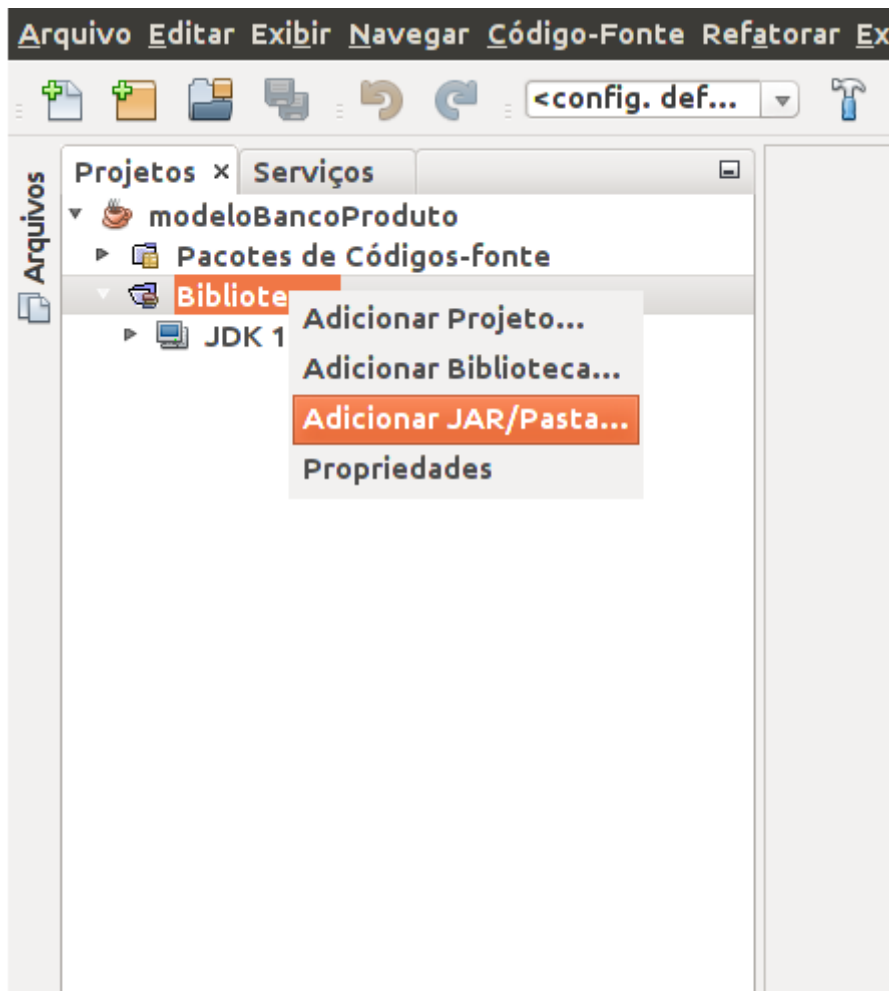
EXEMPLO CONECTOR

Esse é o conector que vamos utilizar para realizar a conexão do Java com o banco de dados.



ADICIONANDO O CONECTOR .JAR

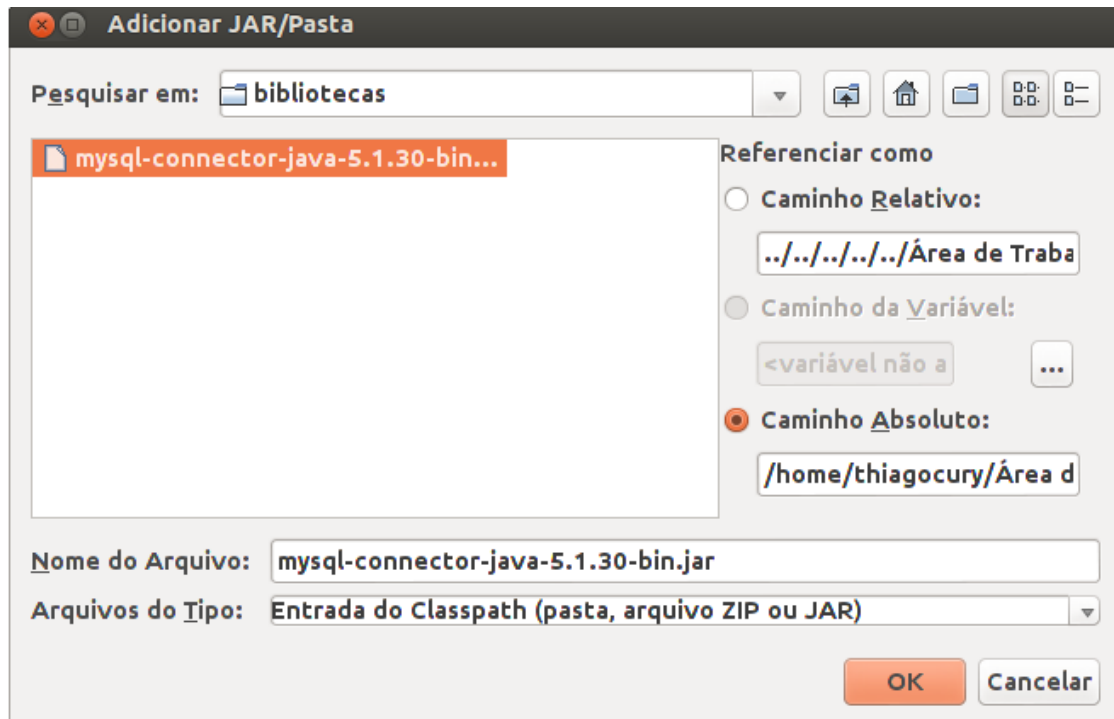
Para adicionar o conector basta clicar com o botão direito do mouse em bibliotecas, em seguida escolher a opção adicionar JAR/PASTA, conforme Figura:



SELECIONANDO O CONECTOR

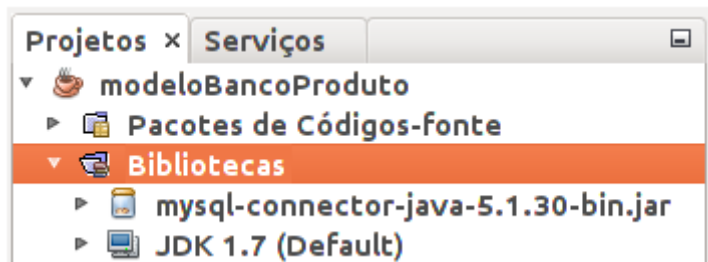
Em seguida procure o conector no seu computador. Observação: Eu sempre insiro o conector em uma pasta chamada "bibliotecas" dentro da

pasta do projeto. Se você está estudando pelos meus códigos de exemplo, note que eu coloco uma pasta chamada bibliotecas na raiz e lá estará o conector. A Figura a seguir ilustra a explicação:



CONECTOR INSERIDO

Após inserir o conector, o mesmo aparecerá em bibliotecas parecendo um "pote". Parece um pote de mel ou um pote com um vagalume dentro. hehe. Qualquer ícone que aparecer que não tiver essa aparência está errado e não funcionará.



CRIANDO CÓDIGO PARA CONECTAR NO BANCO DE DADOS

Classe Conexao – deve ser criada dentro do pacote persistência.

Substituir os “XXX” por nome do banco, usuário e senha do banco de dados instalado.

```
public class Conexao {
    //cria uma constante com o endereçamento da base de dados no conector com mysql
    private static String URL = "jdbc:mysql://localhost:3306/XXX";
    //cria uma constante para determinar usuário predefinido do banco de dados (na instalação)
    private static String USUARIO= "XXX";
    //cria uma constante para determinar a senha do usuário
    private static String SENHA = "XXX";

    //método que estabelece a conexão com o banco de dados
    public static Connection getConnection() throws SQLException{
        //diz que ainda não foi estabelecida conexão com o BD
        Connection c = null;
        //tenta estabelecer conexão
        try{
            c = DriverManager.getConnection(URL, USUARIO, SENHA);
            //caso a tentativa tenha sido falha, gera uma exceção
        }catch(SQLException e){
            throw new SQLException("Erro ao conectar!" + e.getMessage());
        }
        return c;
    }
}
```

SIGLAS

- VO = Value Object - Objeto de valor
- DAO = Data Access Object - Acesso aos dados do objeto
- Factory = Fábrica

CLASSES NOVAS

Para conectar com o Banco de Dados, vamos criar algumas classes novas dentro do nosso projeto. As classes são:

CLASSE PRODUTODAO

Dentro dessa classe colocaremos todos os códigos de SQL DML(Data manipulation language), ou seja, vamos colocar os seguintes códigos:

- insert
- select
- update
- delete

Esses códigos SQL serão inseridos nos seguintes métodos:

- cadastrar - insert

- buscar - select
- atualizar - update
- excluir - delete

O primeiro método que vamos inserir nessa classe é o método `cadastrarProduto`. Esse método receberá um objeto produto vindo da interface.

```
public void cadastrarProduto(ProdutoVO pVO) throws SQLException {

    //Buscando uma conexão com o Banco de Dados
    Connection con = ConexaoBanco.getConexao();
    /*Criando obj. capaz de executar instruções
    SQL no banco de dados*/
    Statement stat = con.createStatement();

    try {
        //String que receberá instrução SQL
        String sql;
        /* Montando a instrução INSERT para inserir
        um objeto pVO no Banco MySQL */
        sql = "insert into produto(idproduto,nome,valorcusto,quantidade)"
            +"values(null,'" +pVO.getNome()+"','" +pVO.getValorCusto()+"','" +pVO.getQuantidade()+"'";

        //Executando o sql
        stat.execute(sql);
    } catch (SQLException e) {
        throw new SQLException("Erro ao inserir produto!");
    } finally {
        con.close();
        stat.close();
    } //fecha finally
} //fecha método
```

CLASSE DAOFACTORY

A classe `DAOFactory` é uma fábrica de DAO, ou seja, ela fabrica objetos de `ProdutoDAO` e retorna para qualquer classe que solicitar. Quando ela retorna um objeto da classe `ProdutoDAO` ela está enviando o objeto com os métodos que tem dentro dele, nesse caso o método `cadastrarProduto`.

Em suma, a classe `Factory` apenas cria um objeto da classe `ProdutoDAO` e retorna o mesmo para a classe que solicitar.

```
- public class DAOFactory {

    private static ProdutoDAO produtoDAO = new ProdutoDAO();

    public static ProdutoDAO getProdutoDAO(){
        return produtoDAO;
    } //fecha método

- } //fecha classe
```

CLASSE PRODUTOSERVICOS

A classe `ProdutoServicos` nós vamos criar dentro do pacote `servicos`.

Essa classe vai conter "uma cópia" dos mesmos métodos que estão em ProdutoDAO.

Essa classe é responsável pela exibição dos métodos para as interfaces(janelas, GUI's). O primeiro método que vamos criar é o "cadastrarProduto(ProdutoVO pVO)"

Esse método recebe um objeto "pVO" vindo da interface, após ele envia para o "cadastrarProduto()" da classe ProdutoDAO.

EXEMPLO CLASSE PRODUTOSERVICOS

```
public class ProdutoServicos {  
    /**  
     * @param pVO Objeto proveniente da interface  
     * @throws SQLException Lançando exceção de SQL  
     */  
    public void cadastrarProduto(ProdutoVO pVO) throws SQLException {  
        ProdutoDAO pDAO = DAOFactory.getProdutoDAO();  
        pDAO.cadastrarProduto(pVO);  
    } //fecha método  
  
    /* próximo método: buscarProdutos() */  
} //fecha classe
```

CLASSE SERVICOSFACTORY

A classe ServicosFactory apenas cria um objeto da classe ProdutoServicos e retorna o mesmo para a classe que solicitar, geralmente uma classe de interface.

Ela é uma Fabrica de serviços, ou seja, ela terá um método que retornará um objeto proveniente da classe ProdutoServicos.

EXEMPLO CLASSE SERVICOSFACTORY

```
public class ServicosFactory {  
    private static ProdutoServicos produtoServicos = new ProdutoServicos();  
    public static ProdutoServicos getProdutoServicos(){  
        return produtoServicos;  
    } //fecha método  
} //fecha classe
```

ALTERANDO A GUICADPRODUTO

Para que a GUICadProduto envie as informações que o usuário digitou, precisamos inserir 2 linhas após receber os dados no objeto.

EXEMPLO CLASSE GUICADPRODUTO

```
private void cadastrar() {
    try {
        ProdutoVO pVO = new ProdutoVO();
        pVO.setNome(jtNome.getText());
        pVO.setValorCusto(Double.parseDouble(jtValorCusto.getText()));
        pVO.setQuantidade(Integer.parseInt(jtQuantidade.getText()));

        //Enviando o objeto pVO para o banco de dados
        ProdutoServicos ps = servicos.ServicosFactory.getProdutoServicos();
        //Chamando o método cadastrarProduto e enviando o objeto pVO.
        ps.cadastrarProduto(pVO);

        JOptionPane.showMessageDialog(
            rootPane,
            "Produto cadastrado com sucesso!");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(
            rootPane,
            "Erro! "+e.getMessage(),
            "erro",
            JOptionPane.ERROR_MESSAGE);
    } //fecha catch
} //fecha método cadastrar
```

ALTERANDO A CLASSE CONEXAOBANCO

Nunca esqueça de alterar a classe ConexaoBanco no início ou após terminar todo o código. Ela é importante pois é nela que inserimos a String de conexão, usuário e a senha para o Java se conectar no banco de dados.

Os atributos que devem ser modificados encontram-se na lista logo abaixo:

- private static String URL = "jdbc:mysql://localhost:3306/XXXX";
- private static String USUARIO = "XXXX";
- private static String SENHA = "XXXX";

Na URL você deve modificar inserindo o nome do banco ao invés de "XXXX". Já no usuário, devemos colocar o nome do usuário que vai acessar o banco de dados, geralmente o nome é "root". Isso claro, quando estamos trabalhando com um banco local na nossa máquina, apenas para teste.

A senha você deve alterar pela senha que você cadastrou ao instalar o banco de dados MySQL.

MUDANÇAS NA CLASSE PRODUTODAO

MÉTODO NOVO

Para buscar os dados do Banco de Dados, vamos criar um novo método dentro da classe ProdutoDAO. Esse método vai conter o seguinte SQL dentro:

- select
Esse código SQL será inserido no seguinte método:
- buscarProdutos()

EXEMPLO MÉTODO BUSCARPRODUTOS()

```

/**
 * @author Thiago Cury
 * @since 09/04/2014 - 09:05
 * @version 1.0 beta
 */
public ArrayList<ProdutoVO> buscarProdutos() throws SQLException {
    Connection con = ConexaoBanco.getConexao();
    Statement stat = con.createStatement();
    try {
        String sql;
        sql = "select * from produto";

        ResultSet rs = stat.executeQuery(sql);
        ArrayList<ProdutoVO> prod = new ArrayList<>();

        while (rs.next()) {
            ProdutoVO p = new ProdutoVO();
            p.setIdProduto(rs.getLong("idproduto"));
            p.setNome(rs.getString("nome"));
            p.setValorCusto(rs.getDouble("valorcusto"));
            p.setQuantidade(rs.getInt("quantidade"));

            prod.add(p);
        } // fecha while
        return prod;
    } catch (SQLException e) {
        throw new SQLException("Erro ao buscar produtos! " + e.getMessage());
    } finally {
        con.close();
        stat.close();
    } // fecha finally
} // fecha método

```

ABRINDO AS CONEXÕES COM O BANCO

Assim como o método cadastrarProduto, a primeira coisa que esse método fará é criar uma conexão com o banco de dados e um Statement para poder executar códigos SQL.

MONTANDO UMA STRING COM O SQL

Após criar a conexão com banco será montada uma string contendo o comando "SQL" que fará o "select" na tabela do banco de dados.

Esse comando select será executado por um método chamado "executeQuery". Esse método retorna um "ResultSet". O ResultSet retorna todo o conjunto de dados do banco de dados, ou seja, os dados. Se você reparar na Figura a seguir, verá que todos os

dados que estão sendo retornados do banco de dados estão entrando dentro de um objeto chamado "rs" proveniente da classe "ResultSet".

EXEMPLO MONTAGEM E EXECUÇÃO DO SQL

```
String sql;  
sql = "select * from produto";  
  
ResultSet rs = stat.executeQuery(sql);
```

Como utilizarei apenas 1 ArrayList, eu crio o mesmo antes do laço(while) iniciar.

O laço que vou utilizar é o while, isso porque como os dados acabaram de retornar do banco de dados não sabemos ao certo quantas linhas retornaram do banco. Se você reparar, verá que o while vai ser executado enquanto houver uma próxima linha, ficando: `while(rs.next()){ }`

Cada volta que o while executar será criado um novo objeto produto, preenchido e no final do while inserido dentro do ArrayList.

Note que após o término do while, ou seja depois do comentário "fecha while" o ArrayList chamado "prod" com todos os produtos é retornado através da instrução "return".

A Figura a seguir ilustra a explicação:

EXEMPLO MAPEAMENTO OBJETO RELACIONAL NO WHILE

```
ArrayList<ProdutoVO> prod = new ArrayList<>();  
  
while (rs.next()) {  
    ProdutoVO p = new ProdutoVO();  
    p.setIdProduto(rs.getLong("idproduto"));  
    p.setNome(rs.getString("nome"));  
    p.setValorCusto(rs.getDouble("valorcusto"));  
    p.setQuantidade(rs.getInt("quantidade"));  
  
    prod.add(p);  
} // fecha while  
return prod;
```


CLASSE PRODUTOSERVICOS

Essa classe vai conter "uma cópia" do método "buscarProdutos" que está em ProdutoDAO. Essa cópia será apresentada para as GUI's(interfaces), para que seja escolhido qual o "serviço" será utilizado, como: cadastrar, buscar, etc.

EXEMPLO MÉTODO BUSCAR CLASSE PRODUTOSERVICOS

```
/**
 * @return Retornando um Array contendo todos os produtos do banco de dados.
 * @throws SQLException Lançando exceção de SQL
 */
public ArrayList<ProdutoVO> buscarProdutos() throws SQLException {
    ProdutoDAO pDAO = DAOFactory.getProdutoDAO();
    return pDAO.buscarProdutos();
} // fecha método
```

CLASSE GUIMANUTENCAOPRODUTO

Para essa interface utilizaremos um componente chamado "JTable" e 2 "JButton's".

O primeiro botão preencherá a tabela quando o usuário solicitar, já o segundo botão limpará as linhas da tabela se o usuário assim desejar.

Além desses componentes, criaremos 2 métodos, são eles: preencherTabela e limparTabela.

DEFAULTTABLEMODEL

O DefaultTableModel será responsável pela criação de um modelo de tabela com o cabeçalho que quisermos. Para isso precisamos declarar ele como variável global da classe e inserir o nome das colunas que desejamos que tenha na tabela, conforme Figura a seguir:

EXEMPLO DECLARAÇÃO DEFAULTTABLEMODEL

```
/**
 * @author Thiago Cury
 * @since 09/04/2014 - 09:55
 * @version 1.0 beta
 */
public class GUIManutencaoProduto extends javax.swing.JInternalFrame {

    /* Criando um modelo de tabela padrão
    com o nome das colunas */
    DefaultTableModel dtm = new DefaultTableModel(
        new Object[][] {},
        new Object[] { "Código", "Nome", "Valor Custo", "Quantidade" });
}
```

PREENCHERTABELA

Esse método será responsável por buscar o ArrayList com os dados e mapear os mesmos para entrarem no componente "DefaultTableModel".

Para preencher o modelo de tabela "dtm" foi utilizado um "laço" (for). Após a conclusão do laço o "dtm" vai ser "setado" no componente "JTable".

Obsevação: No componente JTable só podemos inserir String, ou seja, para fazer a conversão vou utilizar um método da própria classe "String" chamado "String.valueOf()". Esse método recebe por parâmetro qualquer tipo de dado e retorna convertendo sempre uma "String".

EXEMPLO MÉTODO PREENCHERTABELA

```
private void preencherTabela() {
    try {
        //Buscando objeto ProdutoServicos
        ProdutoServicos ps = ServicosFactory.getProdutoServicos();

        /* Criando um ArrayList<ProdutoVO> vazio
        para receber o ArrayList com os dados */
        ArrayList<ProdutoVO> prod = new ArrayList<>();

        //Recebendo o ArrayList cheio em produtos
        prod = ps.buscarProdutos();

        for (int i = 0; i < prod.size(); i++) {
            dtm.addRow(new String[]{
                String.valueOf(prod.get(i).getIdProduto()),
                String.valueOf(prod.get(i).getNome()),
                String.valueOf(prod.get(i).getValorCusto()),
                String.valueOf(prod.get(i).getQuantidade())
            });
        }
        //fecha for

        /* Adicionando o modelo de tabela
        com os dados na tabela produto */
        jTableProduto.setModel(dtm);

    } catch (Exception e) {
        JOptionPane.showMessageDialog(
            this,
            "Erro! " + e.getMessage());
    }
    //fecha catch
}
//fecha preencherTabela
```

EXEMPLO MÉTODO LIMPARTABELA

O único código que coisa que precisamos colocar nesse método é o "dtm.setNumRows(0)". Ele setará para "0"(zero) o número de linhas do modelo da tabela.

```

private void LimparTabela() {
    /* Para limpar a tabela temos que setar o número de
       linhas para zero no default table model */
    dtm.setNumRows(0);
} //fecha LimparTabela

```

EXEMPLO MÉTODO PREENCHERTABELA NO CONSTRUTOR

Após criar o método preencherTabela precisamos chamar o mesmo no construtor da classe. Sendo chamado no construtor ele vai ser executado no momento que a janela for aberta, fazendo com que os dados sejam preenchidos sem o usuário ter que clicar no botão.

```

public GUIManutencaoProduto() {
    initComponents();
    /* Chamando o método preencherTabela
       no construtor */
    preencherTabela();
}

```

CRIANDO CÓDIGO PARA DELETAR DADOS DO BANCO

MUDANÇAS NA CLASSE PRODUTODAO

MÉTODO NOVO

Para deletar os dados do Banco de Dados, vamos criar um novo método dentro da classe ProdutoDAO. Esse método vai conter o seguinte SQL dentro:

- "delete from produto where idproduto="+idProduto;
Esse código SQL será inserido no seguinte método:
- deletarProduto(long idProduto)

EXEMPLO MÉTODO DELETARPRODUTO()

```
public void deletarProduto(long idProduto) throws SQLException {  
    //Buscando uma conexão com o Banco de Dados  
    Connection con = ConexaoBanco.getConexao();  
    /*Criando obj. capaz de executar instruções  
    SQL no banco de dados*/  
    Statement stat = con.createStatement();  
  
    try {  
        String sql;  
        /* Montando uma String que delete um registro através de um  
        código(idProduto) enviado pelo usuário. */  
        sql = "delete from produto where idproduto="+idProduto;  
  
        stat.execute(sql);  
    } catch (SQLException se) {  
        throw new SQLException("Erro ao deletar produto! "+se.getMessage());  
    } finally {  
        stat.close();  
        con.close();  
    } //fecha finally  
} //fecha método deletarProduto  
} //fecha classe
```

ABRINDO AS CONEXÕES COM O BANCO

Assim como os métodos cadastrar e buscar, a primeira coisa que esse método fará é criar uma conexão com o banco de dados e um Statement para poder executar códigos SQL.

MONTANDO UMA STRING COM O SQL

Após criar a conexão com banco será montada uma string contendo o comando "SQL" que fará o "delete" na tabela do banco de dados.

Esse comando delete será executado por um método chamado "execute". Esse método retorna um "boolean", porém não utilizarei o mesmo para não complicar muito o entendimento do código.

CLASSE PRODUTOSERVICOS

Essa classe vai conter "uma cópia" do método "deletarProduto" que está em ProdutoDAO. Essa cópia será apresentada para as GUI's(interfaces), para que seja escolhido qual o "serviço" será utilizado, como: cadastrar, buscar, deletar, etc.

EXEMPLO MÉTODO DELETAR CLASSE PRODUTOSERVICOS

```
//Método deletarProduto
/**
 * @author Thiago Cury
 * @since 17/04/2014
 * @version 1.0beta
 * @param idProduto
 * @throws SQLException
 */
public void deletarProduto(long idProduto) throws SQLException{
    ProdutoDAO pDAO = DAOFactory.getProdutoDAO();
    pDAO.deletarProduto(idProduto);
} // fecha método deletarProduto
```

CLASSE GUIMANUTENCAOPRODUTO

Nessa interface utilizaremos apenas um botão chamado deletar. Ele chamará um método chamado "deletar()". Esse método utilizará 2 método novos, são eles:

- `getSelectedRow()`
- `getValueAt(row,column)`

GETSELECTEDROW()

Esse método retorna a linha(int) que foi selecionada pelo usuário. Caso o usuário não selecione nenhuma linha o método retorna "-1". Lembrando que o Java trata a tabela como uma matriz, ou seja, a primeira linha sempre será o número zero e assim por diante.

GETVALUEAT(ROW,COLUMN)

Esse método retorna um Object, tipo mais primitivo e pai de todos os tipos da linguagem Java. Com esse tipo de dados conseguimos converter o retorno da tabela para qualquer outro tipo, no caso String.

MÉTODO DELETAR

A primeira coisa que o método faz é buscar a linha que o usuário selecionou. Caso o usuário não tenha selecionado nenhuma linha, o mesmo envia uma mensagem para o usuário solicitando que ele selecione uma linha. Após esse teste o método busca o código na coluna zero(0) através do método "getValueAt()". Após buscar o código, o mesmo é enviado para o método "deletarProduto"

EXEMPLO MÉTODO DELETAR()

```
private void deletar() {  
    try {  
        /* Buscando a linha que o usuário clicou */  
        int linha = jTableProduto.getSelectedRow();  
        /* Testando se o usuário selecionou alguma linha. */  
        if (linha == -1) {  
            JOptionPane.showMessageDialog(  
                this,  
                "Você não selecionou nenhuma linha!");  
        } else {  
            ProdutoServicos ps = ServicosFactory.getProdutoServicos();  
  
            /* Buscando o idProduto da linha selecionada. 0 zero(0) indica  
            que vamos buscar o valor da primeira coluna. */  
            String idProduto = (String) jTableProduto.getValueAt(linha, 0);  
  
            /* Enviando o idProduto para ser excluído. */  
            ps.deletarProduto(Long.parseLong(idProduto));  
  
            //Mensagem de sucesso para o usuário!  
            JOptionPane.showMessageDialog(this,  
                "Produto excluído com sucesso!");  
        }  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(this,  
            "Erro ao deletar! " + e.getMessage());  
    }  
}
```

CRIANDO CÓDIGO PARA FILTRAR DADOS DO BANCO

MUDANÇAS NA CLASSE PRODUTODAO

MÉTODO NOVO

Para filtrar os dados do Banco de Dados, vamos criar um novo método dentro da classe ProdutoDAO. Esse método vai conter o seguinte SQL dentro:

- "select * from produto "+query;
Esse código SQL será inserido no seguinte método:
- filtrar()

EXEMPLO MÉTODO FILTRAR(STRING QUERY)

```
public ArrayList<ProdutoV0> filtrar(String query) throws SQLException{
    Connection con = ConexaoBanco.getConexao();
    Statement stat = con.createStatement();
    try {
        String sql;
        sql = "select * from produto "+query;

        ResultSet rs = stat.executeQuery(sql);
        ArrayList<ProdutoV0> prod = new ArrayList<>();

        while(rs.next()){
            ProdutoV0 pV0 = new ProdutoV0();
            pV0.setIdProduto(rs.getInt("idproduto"));
            pV0.setNome(rs.getString("nome"));
            pV0.setValorCusto(rs.getDouble("valorcusto"));
            pV0.setQuantidade(rs.getInt("quantidade"));
            prod.add(pV0);
        }//Fechou while
        return prod;
    } catch (SQLException se) {
        throw new SQLException("Erro ao buscar dados do Banco! "+se.getMessage());
    } finally {
        stat.close();
        con.close();
    }//fechou finally
} //fechou método filtrar
```

ABRINDO AS CONEXÕES COM O BANCO

Assim como os métodos cadastrar e buscar, a primeira coisa que esse método fará é criar uma conexão com o banco de dados e um Statement para poder executar códigos SQL.

MONTANDO UMA STRING COM O SQL

Após criar a conexão com banco será montada uma string contendo o comando "SQL" que fará o "select" especializado na tabela do banco de dados. Note que junto com o select a variável "query" é concatenada, sendo que essa é formada na GUIManutenção conforme o que o usuário digitará.

Esse comando "select" será executado por um método chamado "executeQuery". Esse método retorna um "ResultSet" com tudo que foi pesquisado no banco de dados.

CLASSE PRODUTOSERVICOS

Essa classe vai conter "uma cópia" do método "filtrar" que está em ProdutoDAO. Essa cópia será apresentada para as GUI's(interfaces), para que seja escolhido qual o "serviço" será utilizado, como: cadastrar, buscar, deletar, filtrar etc.

EXEMPLO MÉTODO FILTRAR NA CLASSE PRODUTOSERVICOS

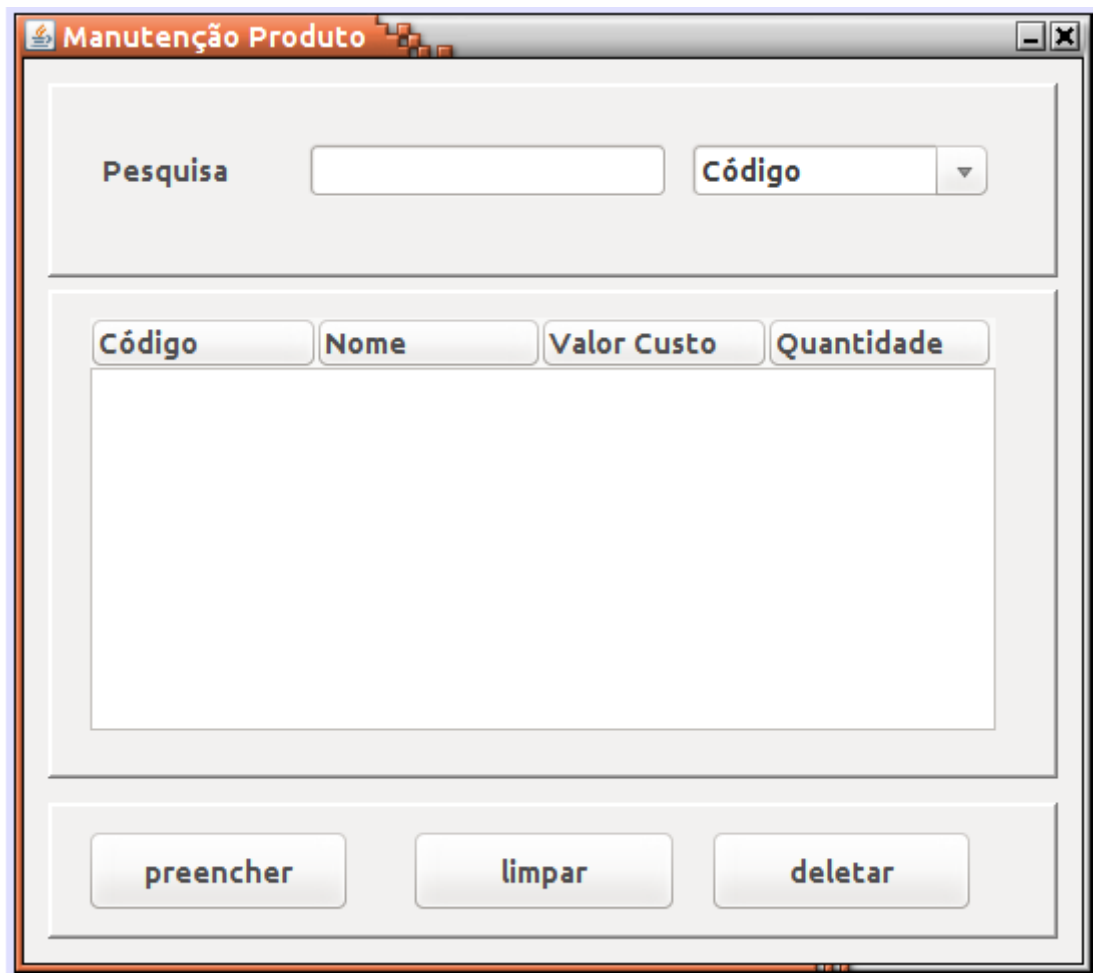
```
public ArrayList<ProdutoV0> filtrar(String query) throws SQLException{  
    ProdutoDAO pDAO = DAOFactory.getProdutoDAO();  
    return pDAO.filtrar(query);  
} //fecha método filtrar
```

CLASSE GUIMANUTENCAOPRODUTO

Nessa interface utilizaremos 3 componentes além de todos que já se encontram na janela, são eles:

- JLabel
- JTextField
- JComboBox

EXEMPLO DE COMO DEVE FICAR A INTERFACE



JCOMBOBOX

Através da "JComboBox" o usuário poderá escolher como prefere consultar um produto. Para esse exemplo foram disponibilizadas apenas 4 opções, são elas: Código, Nome, Valor Custo e Quantidade.

MÉTODO FILTRAR

A primeira coisa que o método faz é testar se o campo de pesquisa está vazio. Se ele estiver, a tabela é preenchida com todos os dados. Se o campo não estiver preenchido significa que o usuário digitou pelo menos uma letra, então o código busca a letra que o usuário digitou, monta uma query e realiza um filtro no banco de dados.

Observação: Para continuar com o mesmo tamanho de fonte esse método ficou cortado, faltando assim a última parte. Após o "fecha else" ficou faltando apenas o encerramento do "try-catch".

EXEMPLO MÉTODO FILTRAR()

```
private void filtrar(){
    try {
        if (jtPesquisa.getText().isEmpty()) {
            preencherTabela();
        }else{
            ProdutoServicos ps = ServicosFactory.getProdutoServicos();
            String pesquisa = (String) jcbPesquisa.getSelectedItem();
            String query;
            if(pesquisa.equals("Código")){
                query = "where idproduto = "+jtPesquisa.getText();
            }else if(pesquisa.equals("Nome")){
                query = "where nome like '%" + jtPesquisa.getText() + "%'";
            }else if(pesquisa.equals("Valor Custo")){
                query = "where valorcusto like '%" + jtPesquisa.getText() + "%'";
            }else{
                query = "where quantidade = "+ jtPesquisa.getText();
            }//fecha else

            ArrayList<ProdutoVO> prod = new ArrayList<>();
            prod = ps.filtrar(query);

            for (int i = 0; i < prod.size(); i++) {
                dtm.addRow(new String[]{
                    String.valueOf(prod.get(i).getIdProduto()),
                    String.valueOf(prod.get(i).getNome()),
                    String.valueOf(prod.get(i).getValorCusto()),
                    String.valueOf(prod.get(i).getQuantidade())
                });
            }//fecha for
            jTableProduto.setModel(dtm);
        }//fecha else
    }
}
```

Para que o filtro aconteça a cada letra que o usuário digitar o método filtrar() é chamado no evento KeyReleased. O evento KeyReleased acontece toda vez que o usuário "liberar"(soltar) uma tecla. O método limparTabela é chamado antes somente para que o dados não se repitam a cada letra digitada pelo usuário.

EXEMPLO DO EVENTO KEYRELEASED UTILIZADO

```
private void jtPesquisaKeyReleased(java.awt.event.KeyEvent evt) {
    limparTabela();
    filtrar();
}
```

CRIANDO CÓDIGO PARA ALTERAR DADOS DO BANCO

MUDANÇAS NA CLASSE PRODUTODAO

MÉTODO NOVO

Para alterar os dados do Banco de Dados, vamos criar um novo método dentro da classe ProdutoDAO. Esse método vai conter o seguinte SQL dentro:

- "update produto
Esse código SQL será inserido no seguinte método:
- alterarProduto(ProdutoVO pVO)

EXEMPLO MÉTODO ALTERARPRODUTO(PRODUTOVO PVO)

```
public void alterarProduto(ProdutoVO pVO) throws SQLException {  
    Connection con = ConexaoBanco.getConexao();  
    Statement stat = con.createStatement();  
  
    try {  
        String sql;  
  
        sql = "update produto set "  
            + "nome='" + pVO.getNome() + "',"  
            + "valorcusto=" + pVO.getValorCusto() + ","  
            + "quantidade=" + pVO.getQuantidade() + "  
            + "where idproduto=" + pVO.getIdProduto() + "";  
  
        stat.executeUpdate(sql);  
    } catch (SQLException se) {  
        throw new SQLException("Erro ao alterar produto! " + se.getMessage());  
    } finally {  
        con.close();  
        stat.close();  
    } //fecha finally  
} //fecha alterarProduto
```

ABRINDO AS CONEXÕES COM O BANCO

Assim como os métodos cadastrar, buscar, filtrar e deletar, a primeira coisa que esse método fará é criar uma conexão com o banco de dados e um Statement para poder executar códigos SQL.

MONTANDO UMA STRING COM O SQL

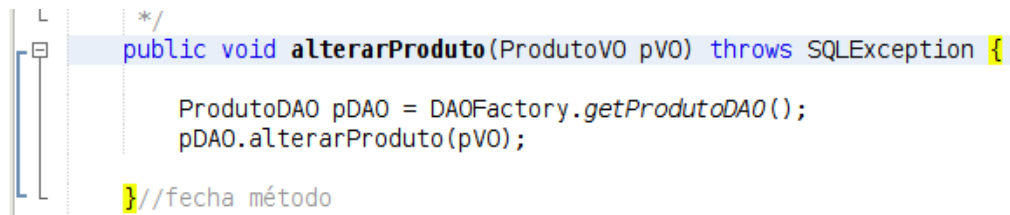
Após criar a conexão com banco será montada uma string contendo o comando "SQL" que fará o "update" na tabela produto no banco de dados. Note que independente dos atributos que foram alterados pelo usuário, colocamos todos para realizar a alteração junto ao banco de dados. Caso o usuário não tenha alterado algum campo, o mesmo permanecerá com o mesmo valor.

Esse comando "update" será executado por um método chamado "executeUpdate".

CLASSE PRODUTOSERVICOS

Essa classe vai conter "uma cópia" do método "alterarProduto" que está em ProdutoDAO. Essa cópia será apresentada para as GUI's(interfaces), para que seja escolhido qual o "serviço" será utilizado, como: cadastrar, buscar, deletar, filtrar, alterar, etc.

EXEMPLO MÉTODO FILTRAR NA CLASSE PRODUTOSERVICOS



```
public void alterarProduto(ProdutoVO pVO) throws SQLException {  
    ProdutoDAO pDAO = DAOFactory.getProdutoDAO();  
    pDAO.alterarProduto(pVO);  
}
```

//fecha método

CLASSE GUIMANUTENCAOPRODUTO

A GUIManutencao além de mostrar os produtos vai permitir que os mesmos sejam alterados pelo usuário. Para isso precisaremos de alguns componentes extras além dos componentes básicos para realizar a manutenção.

COMPONENTES EXTRAS

- 2 botões extras(alterar+confirmar alteração)
- 4 JLabels*
- 4 JTextFields*

*Para esse caso em específico.

EXEMPLO DE COMO DEVE FICAR A INTERFACE

A interface do alterar deve seguir o mesmo padrão para cadastrar, ou seja, os mesmas colunas que aparecem na tabela devem aparecer logo abaixo da tabela para que o usuário possa alterar as informações.

The screenshot shows a Java Swing window titled "produtos cadastrados". It contains a table with the following data:

Código	Nome	Valor Custo	Quantidade
2	trakinas	1.9	30
3	farinha	1.5	20
4	feijão	6.5	400
5	arroz	2.5	500
6	suco	3.98	200
7	café Bom Jesus	5.5	100
8	café Melita	4.5	23
9	Erva Mate Ta...	5.5	70

Below the table are three buttons: "buscar", "limpar", and "alterar". The "alterar" button is highlighted with a blue border. Below these buttons are four text input fields with labels: "Código" (containing "7"), "Nome" (containing "café Bom Jesus"), "Valor Custo" (containing "5.5"), and "Quantidade" (containing "100"). At the bottom is a button labeled "confirmar alteração".

BOTÃO ALTERAR

O botão alterar é responsável apenas por buscar a linha selecionada na tabela e enviar os dados para os JTextField's. Isso deve acontecer para que o usuário possa alterar os dados antes de confirmar a alteração enviando os mesmos para o banco de dados.

EXEMPLO DE COMO DEVE FICAR O CÓDIGO DO BOTÃO ALTERAR

```
/* Esse método busca os valores da linha que o usuário selecionou na
   tabela e coloca os mesmos nas JTextFields */
private void alterar(){
    int linha = jTableProduto.getSelectedRow();
    if(linha != -1){
        /* Buscando valores da linha selecionada na tabela como Objeto e
           transformando os mesmos para String */
        jtCodigo.setText((String)jTableProduto.getValueAt(linha, 0));
        jtNome.setText((String)jTableProduto.getValueAt(linha, 1));
        jtValorCusto.setText((String)jTableProduto.getValueAt(linha, 2));
        jtQuantidade.setText((String)jTableProduto.getValueAt(linha, 3));
    }else{
        JOptionPane.showMessageDialog(this,
            "Você não selecionou nenhuma linha!");
    }//fecha else
} //fecha método
```

BOTÃO CONFIRMAR ALTERAÇÃO

Esse método é bem parecido com o método cadastrar, porém ele busca as informações que o usuário alterou nos JTextFields e envia as mesmas para o método alterarProduto.

EXEMPLO DE COMO DEVE FICAR O CÓDIGO DO BOTÃO CONFIRMAR ALTERAÇÃO

```
/* Método que monta um objeto produto pV0 e envia para o método
   alterarProduto */
private void confirmarAlteracao(){
    try {
        ProdutoV0 pV0 = new ProdutoV0();
        //Buscando dados alterados pelo usuário nos JTextFields
        pV0.setIdProduto(Long.parseLong(jtCodigo.getText()));
        pV0.setNome(jtNome.getText());
        pV0.setValorCusto(Double.parseDouble(jtValorCusto.getText()));
        pV0.setQuantidade(Long.parseLong(jtQuantidade.getText()));

        ProdutoServicos ps = ServicosFactory.getProdutoServicos();
        //Enviando o objeto para ser alterado
        ps.alterarProduto(pV0);

        JOptionPane.showMessageDialog(
            this,
            "Produto Alterado com sucesso!");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(
            this,
            "Erro! "+e.getMessage());
    } //fecha catch
} //fecha método confirmarAlteração
```

CRIANDO CÓDIGO PARA BUSCAR DADOS DO BANCO

Ver projeto.

