



Tutorial elaborado pelo professor
José Gonçalo dos Santos

Contato: jose.goncalo.santos@gmail.com

Criando um aplicação simples com JAVA e
MySQL usando NetBeans – Parte III

<http://www.pusivus.com.br>

1. Introdução	3
2. Testando o servidor web	3
3. Criando e testando o primeiro site.....	5
4. Verificando os sites	7
5. Criando o banco de dados.....	8
6. Copiando o conector para a pasta lib do tomcat.....	11
7. Criando o projeto para as classes java.....	11
8. Criando o pacote	13
9. Adicionando o conector	14
10. Criando as classes para trabalhar com o banco.....	16
11. Criando o projeto web	35
11.1. Adicionando o tomcat ao netbeans	37
12. Adicionando as classes ao projeto web	39
13. Criando as páginas JSP.....	42
13.1. Criando o formulário para proprietário	42
13.2. Criando a página para as operações CRUD para proprietario	49
13.3. Criando o formulário para carro	55
13.4. Criando a página para as operações CRUD para carro	59

1. Introdução

O objetivo desta terceira parte do tutorial é mostrar como criar uma aplicação web, com acesso a banco de dados, de maneira simples e prática. Portanto, conhecimentos sobre teoria e fundamentação a respeito de desenvolvimento web podem ser adquiridos por meio de cursos específicos, recomendo o curso da pusivus (<http://www.pusivus.com.br>).

Para conseguir acompanhar este tutorial é necessário que se tenha os seguintes recursos instalados na sua máquina:

1 - MySQL Server 5.x ou superior (<http://dev.mysql.com/downloads/>);

2 - Jdk 1.5 ou superior

(<http://www.oracle.com/technetwork/java/javase/downloads/index.html>);

3 - NetBeans 6.8 ou superior (http://netbeans.org/index_pt_BR.html);

4 - TomCat 6.0 ou superior (<http://tomcat.apache.org/download-70.cgi>);

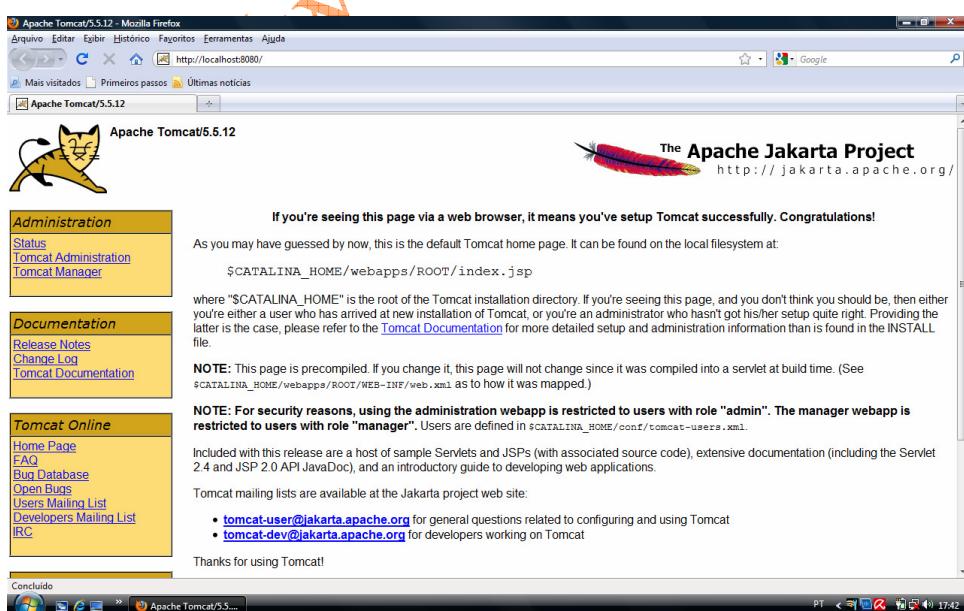
5 - Conector JDBC MySQL (<http://dev.mysql.com/downloads/connector/j/>).

2. Testando o servidor web

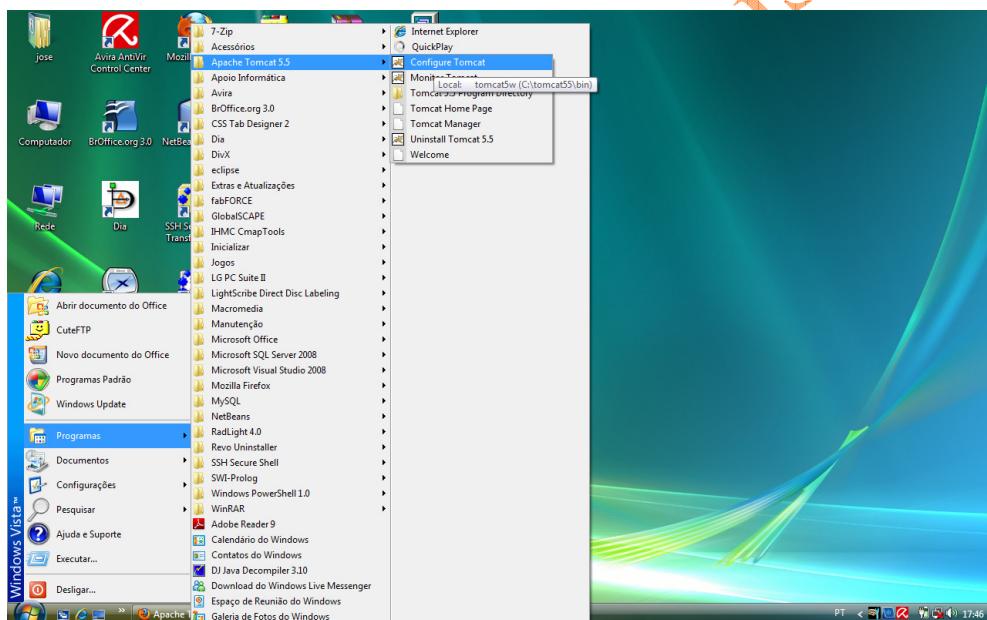
De posse de todas essas ferramentas, vamos testar o nosso servidor web para verificar se está tudo funcionando corretamente . Para tanto, siga os passos abaixo:

Passo 1 – Digite na barra de endereços, do browser, o endereço:

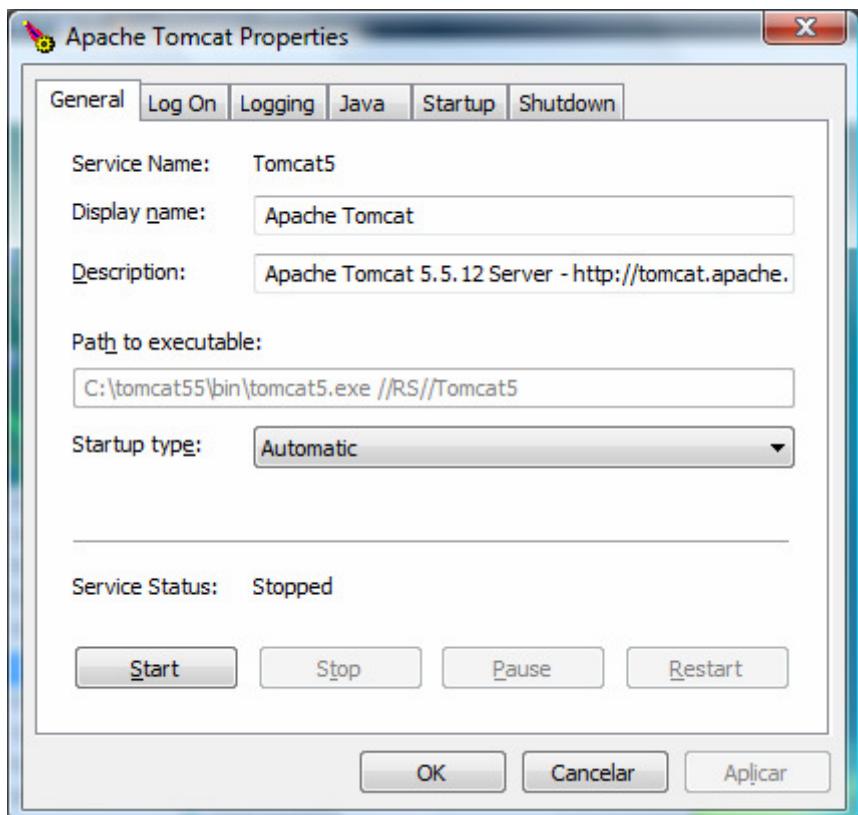
<http://localhost:8080> e dê enter. Se aparecer uma das figuras a seguir, está tudo ok, siga para o tópico 3. Caso contrário, siga para o passo 2.



Passo 2 – Acesse o “Configure Tomcat” (iniciar -> Apache TomCat x.x), conforme figura a seguir.



Passo 3 – Click em “Start” na janela da figura a seguir. Se o botão “Start” desabilitar e “Stop” estiver ativo, faça o teste do passo 1, novamente.



3. Criando e testando o primeiro site

Passo 1 – Localize a pasta “webapps” do Tomcat (c:\arquivos de programas\Apache Software\Tomcat x.x).

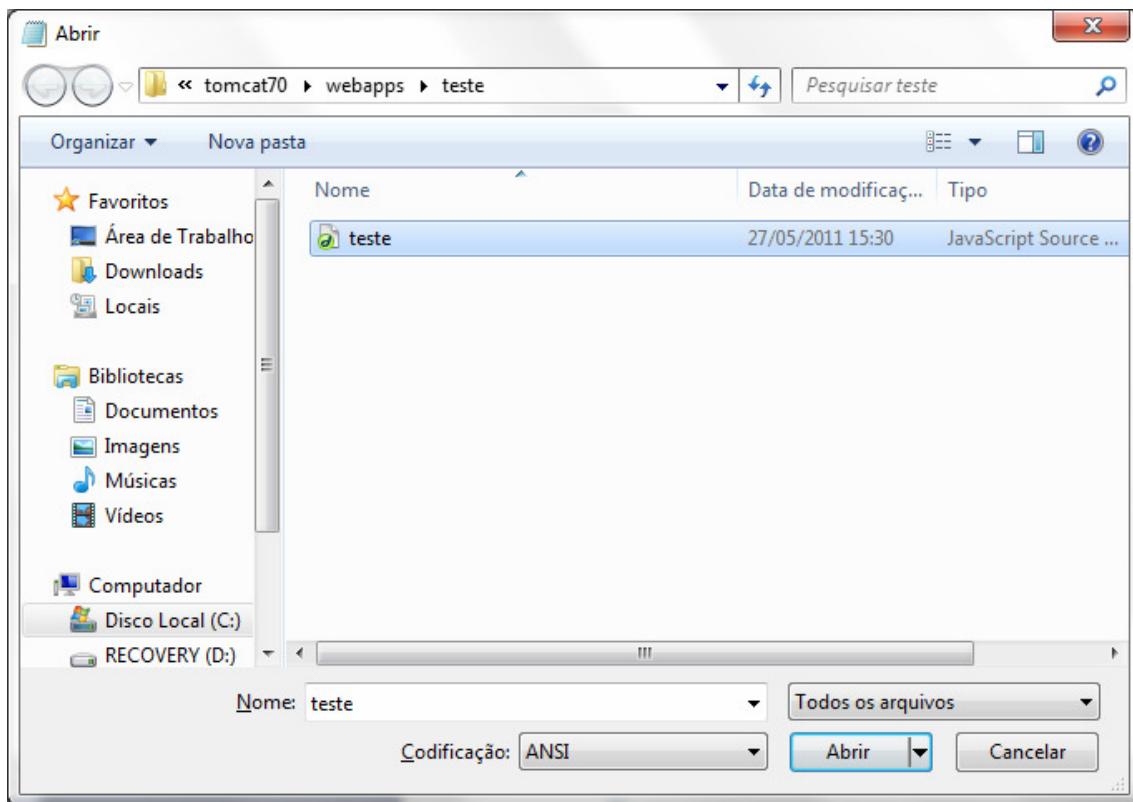
Passo 2 – Dentro da pasta “webapps” crie uma pasta chamada teste (este será o seu site).

Passo 3 – Dentro do seu site (**teste**), crie a pasta WEB-INF (tem que ser em maiúscula, do jeito que está escrito aqui).

Passo 4 – Abra o bloco de notas, ou DreamWeaver, ou outro editor HTML, e digite o código abaixo:

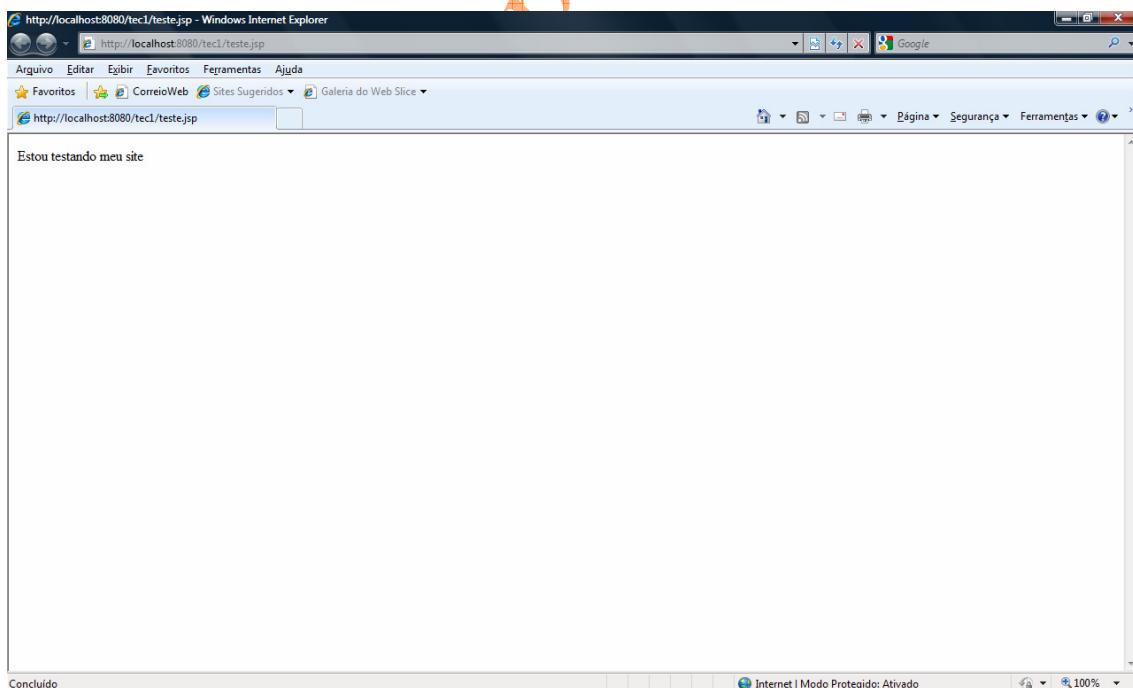
```
<html>
  <body>
    <%>
      out.print("Estou testando meu site");
    <%>
  </body>
</html>
```

Passo 5 – Salve-o, na pasta **teste**, criada no passo 2, com o nome de teste.jsp, conforme mostra figura a seguir. Lembre-se, Tipo deverá estar “Todos os Arquivos”.



Passo 6 – Digite no browser o seguinte endereço:

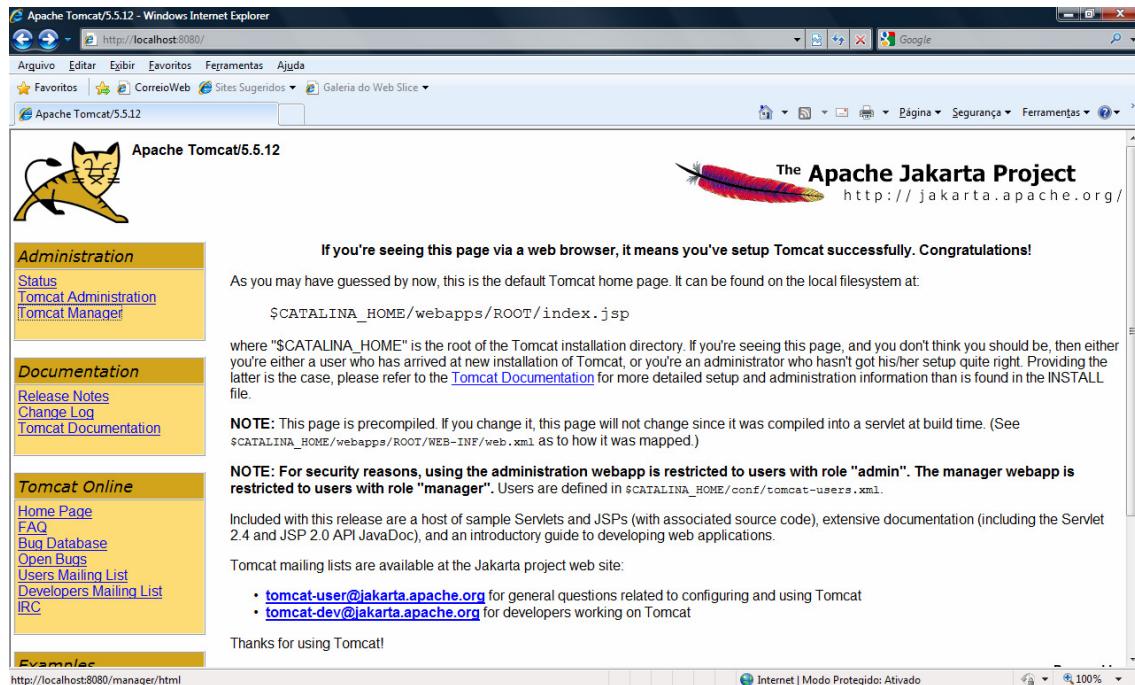
<http://localhost:8080/teste/teste.jsp> e tecle enter. A figura a seguir deverá aparecer.



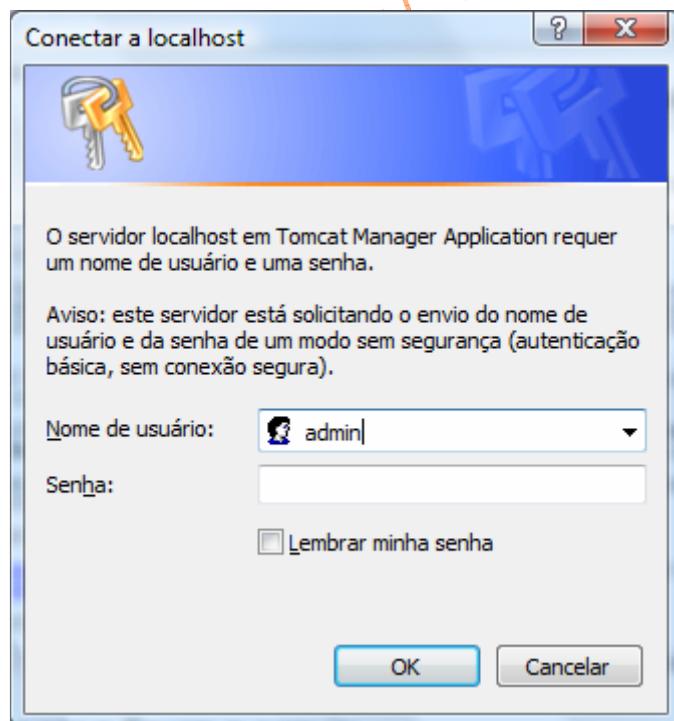
4. Verificando os sites

Passo 1 – Digite o seguinte endereço, na barra de endereço do browser:
<http://localhost:8080/>

Passo 2 – Click em “Tomcat Manager”, conforme figura a seguir.



Passo 2 – Ao aparecer a figura a seguir, digite admin (em minúsculas), no nome do usuário e click em ok, ou o nome e senha que você colocou quando instalou o tomcat.



Se não for esse usuário ou você não se lembra, siga o procedimento abaixo:

- acesse a pasta conf (c:\arquivos de programas\Apache Software\Tomcat x.x\conf);
- edit o arquivo tomcat-users (botão direito -> abrir com -> bloco de notas);
- coloque o trecho de código a seguir no corpo do arquivo (logo abaixo do comando <tomcat-users>);

```
<user name="admin" password="" roles="admin-gui,manager-gui" />
```

- reinicie o tomcat (click em "stop" e depois em "start");
- volte ao passo 2.

Passo 3 – Observe o site (teste) conforme destacado na figura a seguir. Aqui você pode administrá-lo, ou seja, parar (stop), recarregar (reload) ou iniciar (start).

Obs.: Toda vez que você colocar uma classe (.class) novo no seu site, é necessário recarregá-lo (reload).



Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true		Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true		Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/teste	None specified		true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

5. Criando o banco de dados

Agora que temos certeza de que o nosso servidor está funcionando, vamos criar o nosso banco de dados. Neste banco colocaremos apenas duas tabelas, por questões didáticas. Se você não tem familiaridade com banco de dados, aconselho fazer um curso, recomendo o curso da pusivus (<http://www.pusivus.com.br>).

Crie um banco chamado tutorial e crie as tabelas abaixo (você pode rodar o script apresentado em azul, logo abaixo do código das tabelas):

```
CREATE TABLE Proprietario (
    codigo INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
```

```

        nome VARCHAR(50) NULL,
        cidade VARCHAR(30) NULL,
        PRIMARY KEY(codigo)
    ) ;

CREATE TABLE carro (
    placa INTEGER NOT NUL,
    codigo INTEGER UNSIGNED NOT NULL,
    cor VARCHAR(20) NULL,
    descricao VARCHAR(60) NULL,
    PRIMARY KEY(placa),
    INDEX veiculo_FKIndex1(codigo),
    FOREIGN KEY(codigo)
        REFERENCES Proprietario(codigo)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ;

-- MySQL Administrator dump 1.4
--
-- Server version      5.0.16-nt

/*!40101
@OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */; SET
/*!40101
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */; SET
/*!40101
@OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */; SET
/*!40101 SET NAMES utf8 */;

/*!40014      SET          @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0 */; SET
/*!40014      SET          @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */; SET
/*!40101      SET          @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */; SET

-- Create schema tutorial
--

CREATE DATABASE /*!32312 IF NOT EXISTS*/ tutorial;
USE tutorial;

-- Table structure for table `tutorial`.`carro`
--
```

```

DROP TABLE IF EXISTS `carro`;
CREATE TABLE `carro` (
  `placa` varchar(8) NOT NULL,
  `codigo` int(10) unsigned NOT NULL,
  `cor` varchar(20) default NULL,
  `descricao` varchar(60) default NULL,
  PRIMARY KEY (`placa`),
  KEY `veiculo_FKIndex1` (`codigo`),
  CONSTRAINT `carro_ibfk_1` FOREIGN KEY (`codigo`)
  REFERENCES `proprietario` (`codigo`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- 
-- Dumping data for table `tutorial`.`carro`
-- 

/*!40000 ALTER TABLE `carro` DISABLE KEYS */;
/*!40000 ALTER TABLE `carro` ENABLE KEYS */;

-- 
-- Table structure for table `tutorial`.`proprietario`
-- 

DROP TABLE IF EXISTS `proprietario`;
CREATE TABLE `proprietario`(
  `codigo` int(10) unsigned NOT NULL auto_increment,
  `nome` varchar(30) default NULL,
  `cidade` varchar(30) default NULL,
  PRIMARY KEY (`codigo`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- 
-- Dumping data for table `tutorial`.`proprietario`
-- 

/*!40000 ALTER TABLE `proprietario` DISABLE KEYS */;
/*!40000 ALTER TABLE `proprietario` ENABLE KEYS */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */; SET
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;

```

6. Copiando o conector para a pasta lib do tomcat

Se você ainda não baixou o conector (driver) do MySQL, faça agora, acesse: <http://dev.mysql.com/downloads/connector/j/>

Após baixar o conector (vem zipado), descompacte-o e copie o arquivo mysql-connector-java-5.0.4-bin.jar para a pasta lib do tomcat (c:\arquivos de programas\Apache Software\Tomcat x.x\lib).

7. Criando o projeto para as classes java

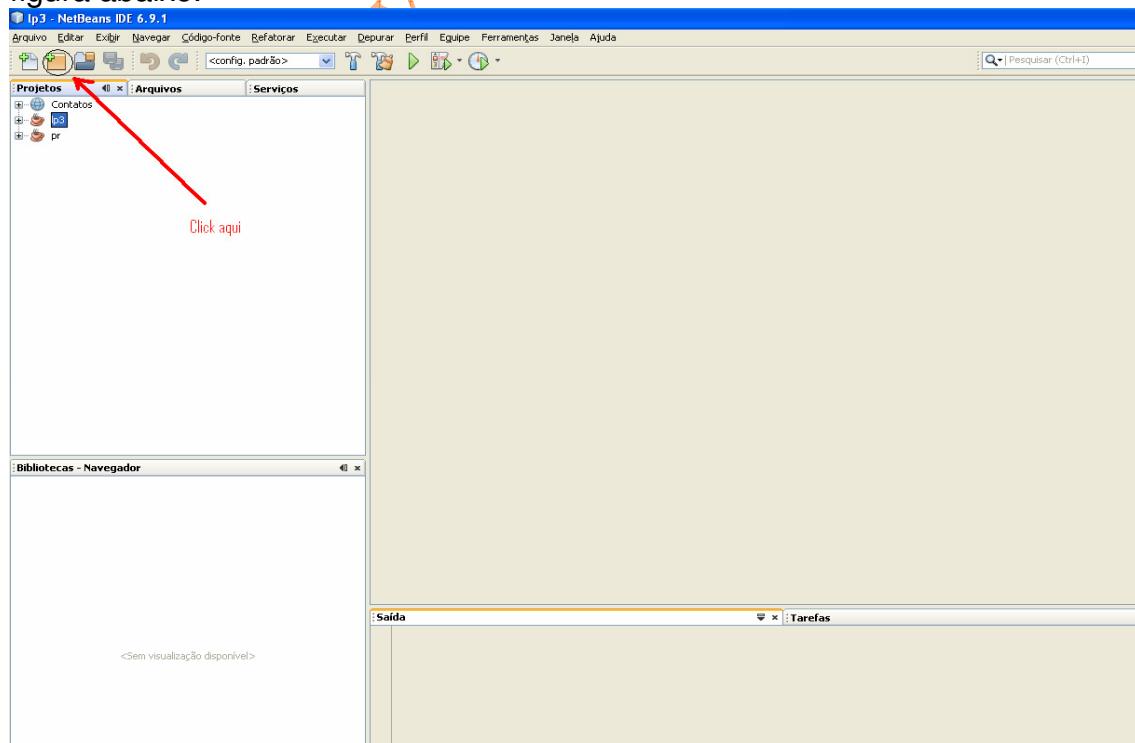
Até o momento temos o banco de dados, o conector e todas as ferramentas necessárias para desenvolvermos nossa aplicação, então vamos criar o nosso projeto para criação das nossas classes.

Obs.: neste tutorial o projeto para a criação das classes será separado do projeto web, tudo por uma questão de escolha.

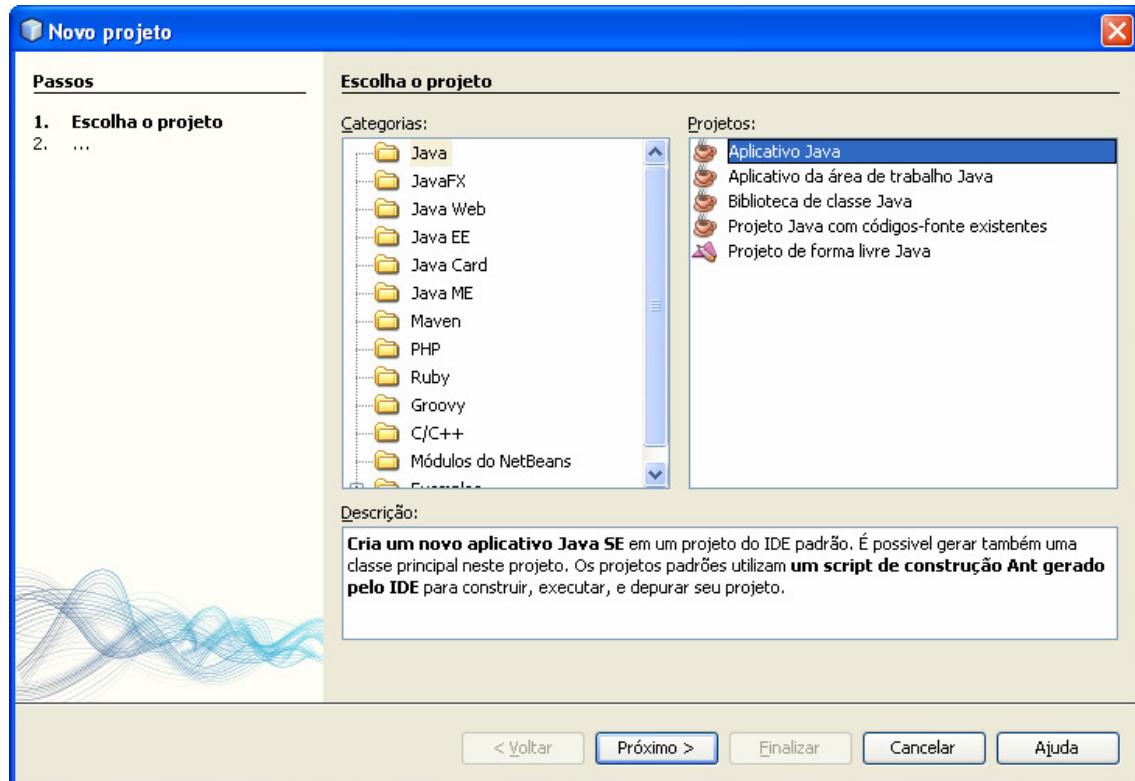
Para esta etapa é necessário que você já tenha o netbeans instalado na sua máquina.

Primeiro passo: Acesse o netbeans (iniciar->netbeans->netbeans 6.x.x). Você verá uma janela semelhante à apresentada na figura abaixo.

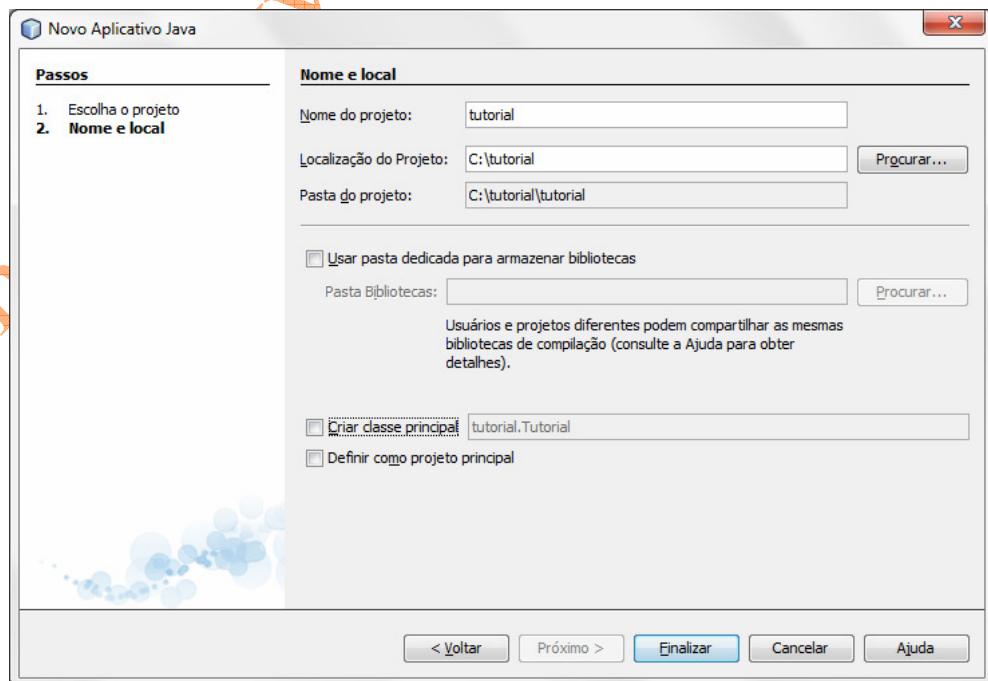
Segundo passo: criar um novo projeto. Para isso, click no ícone mostrado na figura abaixo.



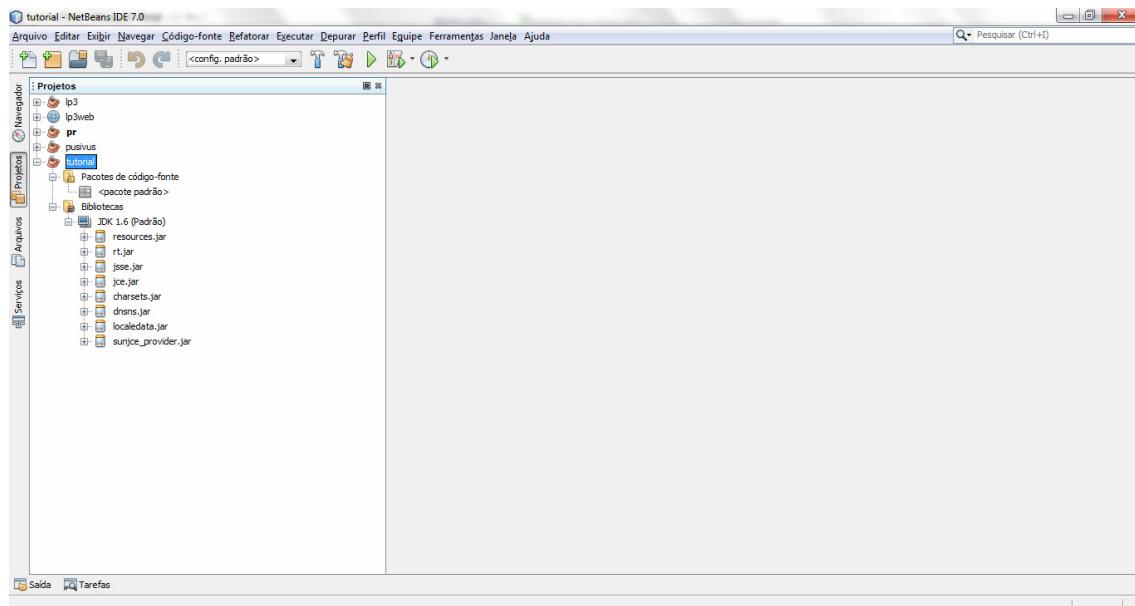
Ao aparecer uma nova janela, selecione Java em categorias e aplicativo Java em projetos, conforme figura abaixo, e click em próximo.



Ao aparecer uma nova janela, coloque o nome tutorial em nome do projeto; click em procurar e aponte para o diretório que se deseja criar o projeto. Neste tutorial foi criada a pasta tutorial,diretamente no diretório raiz, para facilitar o acesso. Desmarque os dois últimos checkbox, conforme mostra a figura abaixo, e click em finalizar.

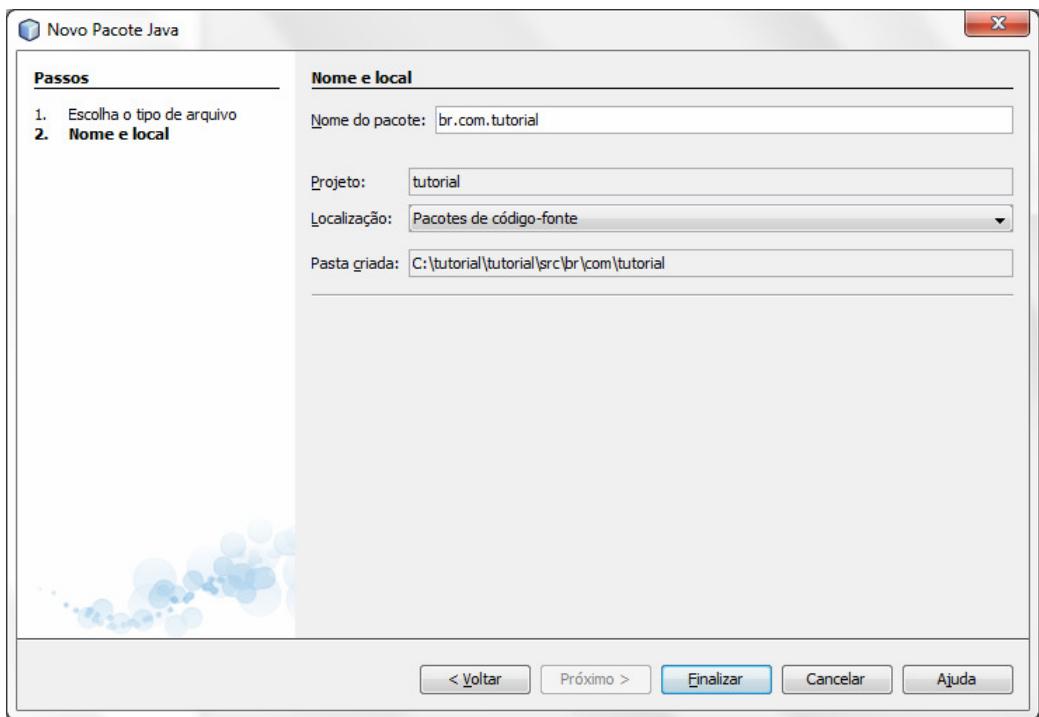


O novo projeto já vem com algumas pastas: pacotes de código fonte (onde guardaremos as nossas classes), pacote de testes, bibliotecas (onde colocaremos nossas bibliotecas externas) e bibliotecas de testes. Só vamos nos preocupar com a primeira e terceira pasta. Observe que temos um pacote vazio nas pastas de pacotes. Já nas pastas de bibliotecas, temos algumas previamente adicionadas pelo netbeans. Não entrarei em detalhe a respeito de cada pasta ou biblioteca, porque está fora do escopo deste tutorial.



8. Criando o pacote

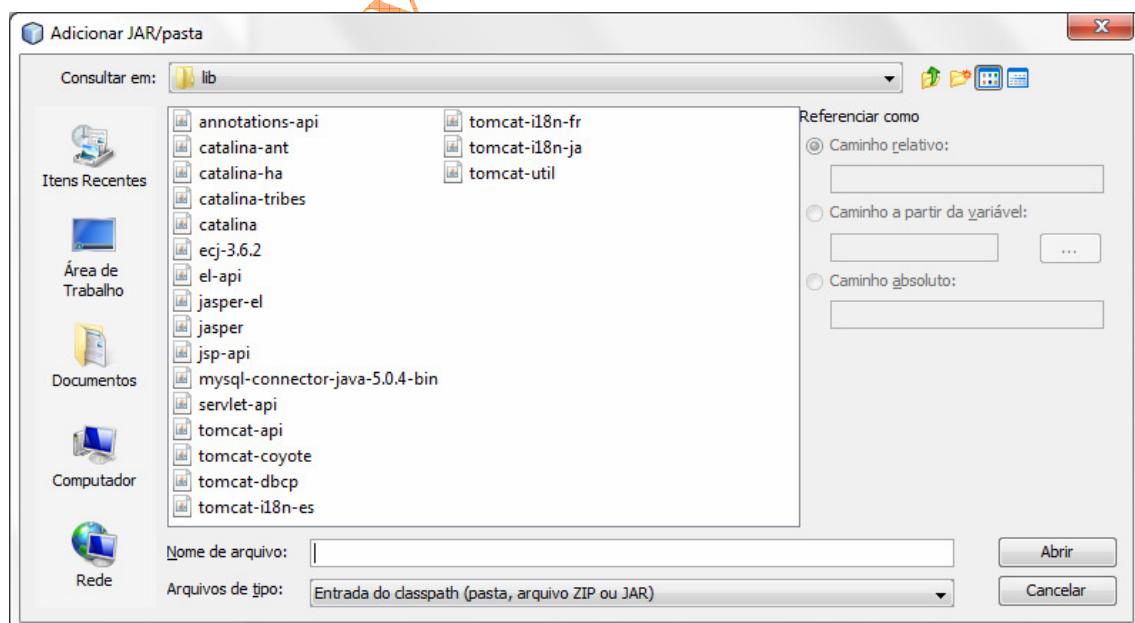
É sempre recomendável que criemos um pacote para colocarmos nossas classes. Para isso, click com o botão direito do mouse sobre a pasta “Pacotes de código fonte”, escolha novo->pacote. Uma nova janela aparecerá preencha os dados conforme figura abaixo. Coloque o nome `br.com.tutorial` para o nome do pacote.



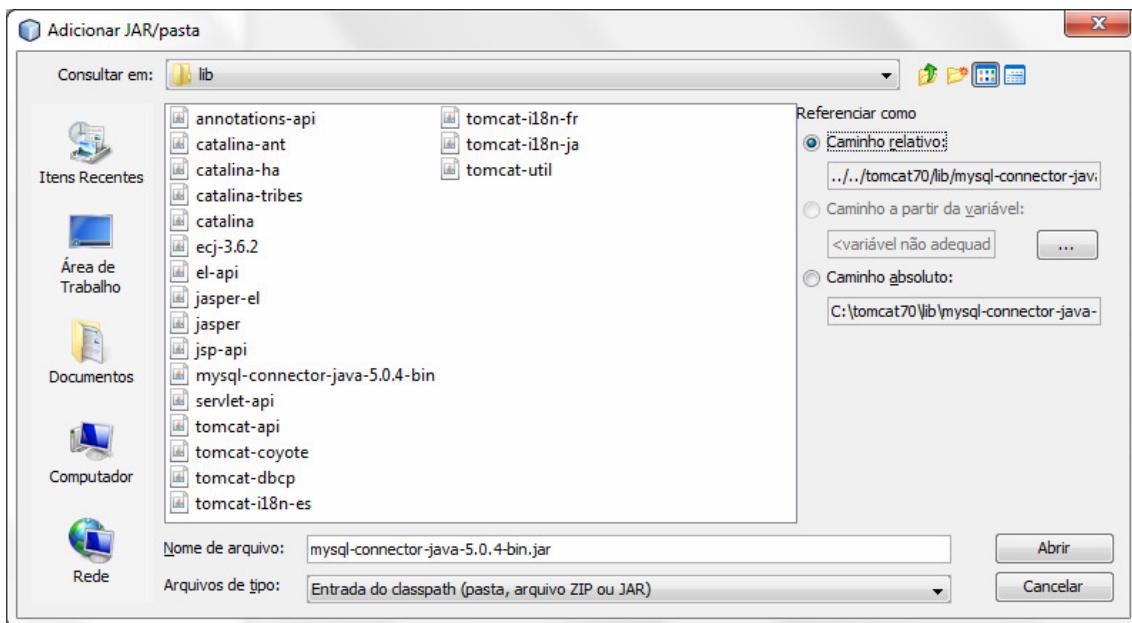
9. Adicionando o conector

Agora precisamos dizer para o netbeans onde está o nosso conector, para fazer isso siga os passos abaixo.

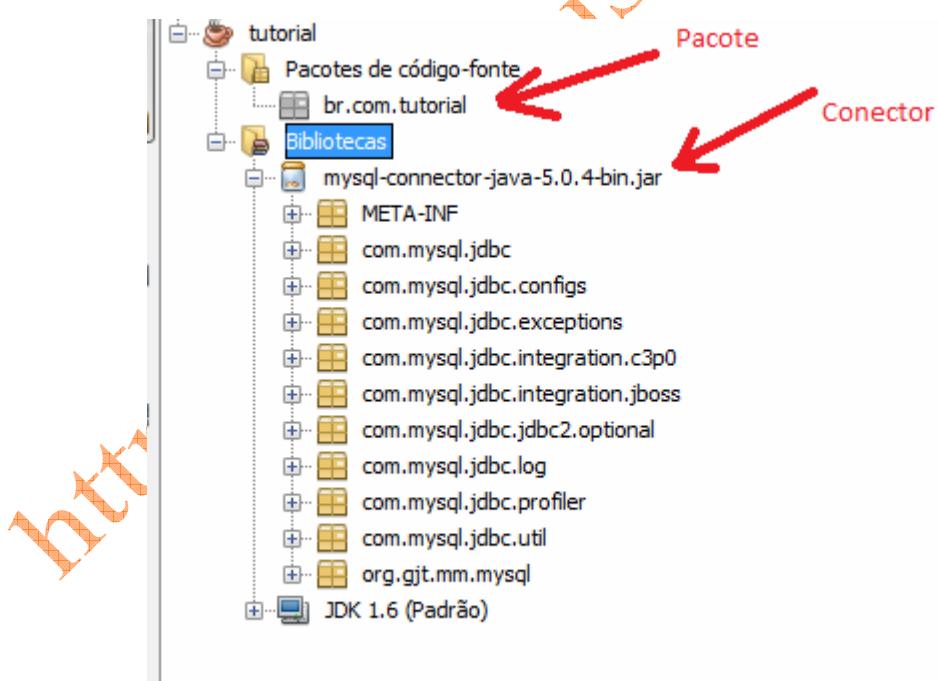
Primeiro passo: click com o botão direito do mouse sobre a pasta “Bibliotecas” e selecione a opção “Adicionar JAR/pasta”. Quando aparecer uma nova janela, navegue por ela até localizar o conector, conforme mostra a figura abaixo.



Segundo passo: selecione o conector e click em “caminho relativo”, conforme figura abaixo, isso diz para o netbeans que ele não deve ficar preso à pasta que se encontra o conector atualmente. Feito isso, click em open (abrir).



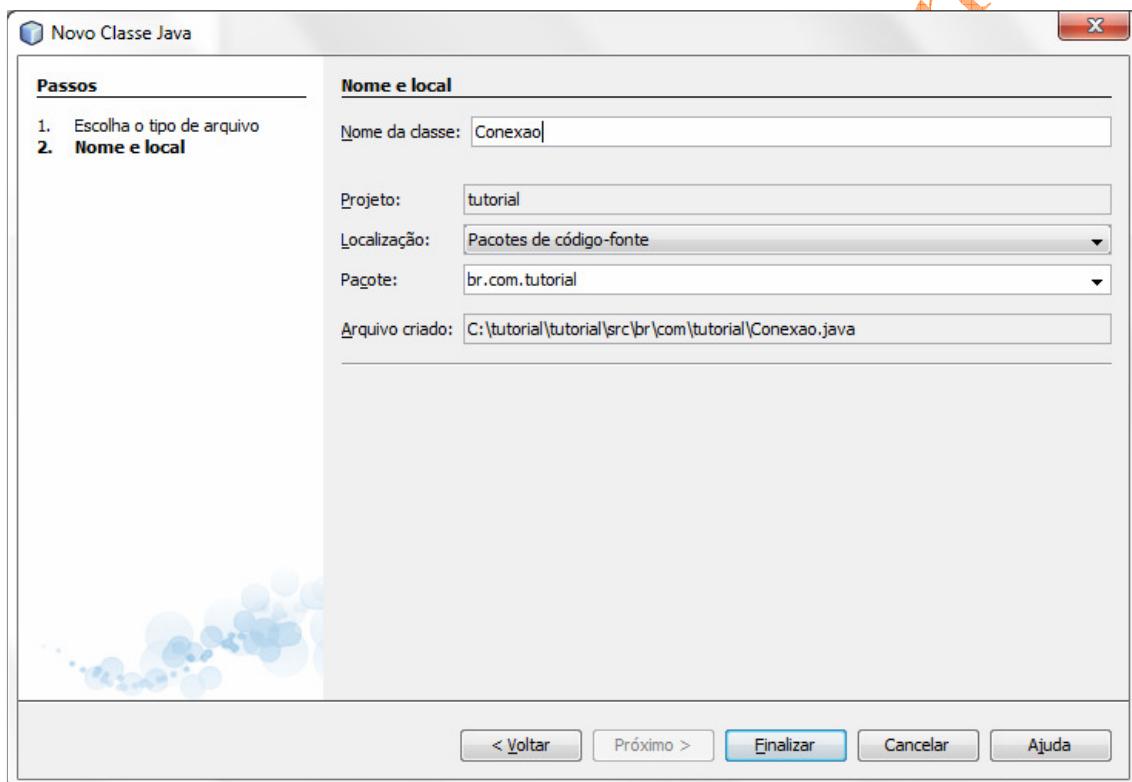
Pronto! Você agora tem um pacote para trabalhar e já tem o conector para o banco de dados, conforme mostra a figura abaixo.



10. Criando as classes para trabalhar com o banco

Bom, já temos o banco, o conector, o projeto e o pacote. Vamos agora criar as nossas classes JAVA para manutenção do nosso banco. Para fazermos isso, precisaremos de 6 classes: CarroBean (onde colocaremos todos os atributos e os métodos de acesso a esses atributos), CarroDAO (onde colocaremos todos os métodos de acesso aos dados), ProprietarioBean (onde colocaremos todos os atributos e os métodos de acesso a esses atributos), ProprietarioDAO (onde colocaremos todos os métodos de acesso aos dados), Conexao (responsável por abrir e fechar as conexões) e Teste (para testarmos as nossas classes).

Primeiro passo: criação da classe Conexao. Para isso, click com o botão direito sobre o pacote criado anteriormente, quando uma nova janela aparecer (figura abaixo), coloque o nome Conexao e click em finalizar.



Uma classe será criada, devemos acrescentar os seguintes códigos nela.

```
package br.com.tutorial; //Este não é necessário, pois já é colocado automaticamente
```

Em uma linha abaixo de package, devemos importar o seguinte pacote para usarmos a classe Connection.

```
import java.sql.*;
```

A linha a seguir aparece automaticamente, pois é o nome da classe

```
public class Conexao {
```

Dentro da classe, vamos criar um método (`abrirConexao`) estático (não é necessário instanciar a classe para utilizá-lo) para abrir uma conexão com o banco de dados. Esse método deverá retornar um objeto do tipo `Connection` e não recebe parâmetro.

```
public static Connection abrirConexao() {
```

Dentro do método (`abrirConexao`), vamos criar uma variável de método do tipo `Connection` e atribuir o valor `null` para ela, pois variável de método tem que inicializada.

```
Connection con = null;
```

Ainda dentro do método (`abrirConexao`), vamos abrir um bloco `try` colocar o código para conexão com o banco. Esse tratamento de exceção é necessário porque podem ocorrer as seguintes exceções: `SQLException` e `ClassNotFoundException`.

```
try {
```

Agora precisamos registrar o driver, ou seja, precisamos dizer para o Java que esse é o driver que iremos usar para as conexões.

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

Depois do driver registrado, podemos fazer a conexão usando o método estático (não é necessário instanciar a classe para usa-lo) `getConnnection` da classe `DriverManager`. Esse método recebe uma `String` como argumento, que é o endereço do banco, bem assim o usuário e senha. Por questão de legibilidade do código, criaremos uma variável chamada `url` para colocar essa `String`.

```
String url = "";  
url += "jdbc:mysql://127.0.0.1/tutorial?";  
url += "user=root&password=";
```

Antes de passar a `url` (poderia se qualquer nome) como parâmetro do método `getConnnection`, vamos analisar cada componente dela (`url`). O que é feito a seguir.

`jdbc:mysql://127.0.0.1` – endereço do servidor de banco de dados, neste caso é o `localhost`, ou seja, nossa própria máquina.

`tutorial` – nome do nosso banco de dados.

`? – indica que iremos passar um conjunto de parâmetro.`

`user=root` – usuário do banco de dados, neste caso, usuário administrador.

password= - senha do usuário, neste caso está em branco porque não colocamos senha para o usuário root na instalação do MySQL.

Obs.: Não pode haver espaço algum entre esses componentes do parâmetro.

Bom, já que temos a nossa String de conexão montada, vamos fazer a nossa conexão e passar o valor para a nossa variável criada no início do método.

```
con = DriverManager.getConnection(url);
```

Se a conexão for realizada com sucesso, podemos avisar usuário que essa foi estabelecida.

```
System.out.println("Conexão aberta.");
```

Como nós abrimos um bloco `try`, devemos ter pelo menos um `catch` ou `finally` para finaliza-lo, no nosso caso teremos 3 `catches`, um para `SQLException` (exceções que podem ocorrer durante a conexão com o banco), outra para `ClassNotFoundException` (exceção que pode ocorrer durante o processo de registro do driver) e outra para `Exception` (exceções quaisquer que possam ocorrer).

```
} catch (SQLException e) {  
    System.out.println(e.getMessage());  
} catch (ClassNotFoundException e) {  
    System.out.println(e.getMessage());  
} catch (Exception e) {  
    System.out.println(e.getMessage());  
}
```

Agora que já temos a nossa conexão, devemos retorna-la para quem a solicitou.

```
return con;
```

Podemos fechar o método, então.

```
}
```

Ainda dentro da classe, vamos criar um método (`fecharConexao`) estático (não é necessário instanciar a classe para utilizá-lo) para fechar uma conexão com o banco de dados. Esse método não tem retorno, mas recebe um objeto do tipo `Connection`, que é a conexão que ele deve fechar.

```
public static void fecharConexao(Connection con) {
```

Ainda dentro do método (`fecharConexao`), vamos abrir um bloco `try` colocar o código para fechar a conexão com o banco. Esse tratamento de exceção é necessário porque pode ocorrer a seguinte exceção: `SQLException`.

```
try {
```

Agora vamos fechar a conexão

```
con.close();
```

Depois da conexão fechada vamos dar um aviso ao usuário de que essa (conexão) foi fechada.

```
System.out.println("Conexão fechada.");
```

Como nós abrimos um bloco `try`, devemos ter pelo menos um `catch` ou `finally` para finaliza-lo, no nosso caso teremos 2 `catches`, um para `SQLException` (exceções que podem ocorrer durante a conexão com o banco) e outra para `Exception` (exceções quaisquer que possam ocorrer).

```
} catch (SQLException e) {  
    System.out.println(e.getMessage());  
} catch (Exception e) {  
    System.out.println(e.getMessage());  
}
```

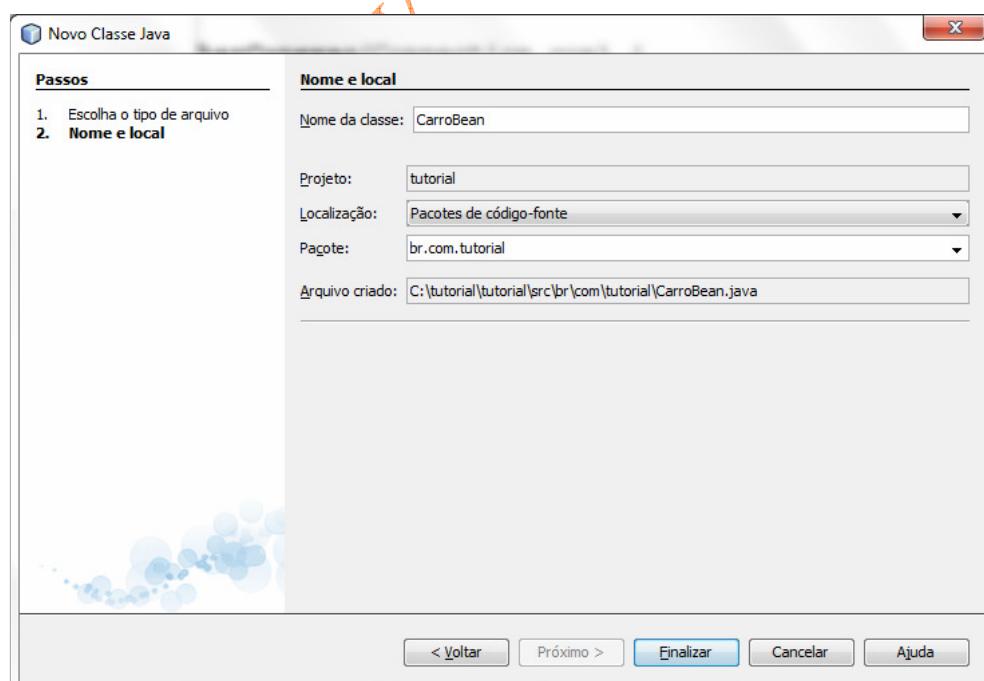
Podemos fechar o método, então.

```
}
```

Agora fechamos a classe.

```
}
```

Segundo passo: criação da classe CarroBean. Para isso, click com o botão direito sobre o pacote criado anteriormente, quando uma nova janela aparecer (figura abaixo), coloque o nome CarroBean e click em finalizar.



Uma classe será criada, devemos acrescentar os seguintes códigos nela.

```
package br.com.tutorial; //Este não é necessário, pois já é colocado automaticamente.
```

```
public class CarroBean {
```

Aqui colocamos os atributos, todos encapsulados com `private` para que não sejam acessados diretamente.

```
private String placa;  
private String cor;  
private String descricao;
```

```
private Integer codigo; //Campo de ligação com o proprietário
```

Aqui colocamos os métodos de acesso aos atributos `private` da classe. Todos os atributos tem um par de métodos, um `get` (responsável por devolver o valor atual do atributo) e um `set` (responsável por setar valor para o atributo). Esses métodos são a interface para acesso aos valores desses atributos.

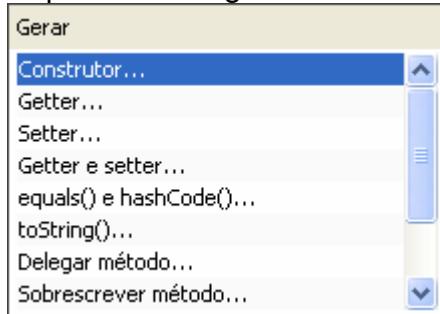
```
public String getCor() {  
    return cor;  
}  
public void setCor(String cor) {  
    this.cor = cor;  
}  
public String getDescricao() {  
    return descricao;  
}  
public void setDescricao(String descricao) {  
    this.descricao = descricao;  
}  
public String getPlaca() {  
    return placa;  
}  
public void setPlaca(String placa) {  
    this.placa = placa;  
}  
public Integer getCodigo() {  
    return codigo;  
}  
public void setCodigo(Integer codigo) {  
    this.codigo = codigo;  
}
```

Precisamos fechar a classe.

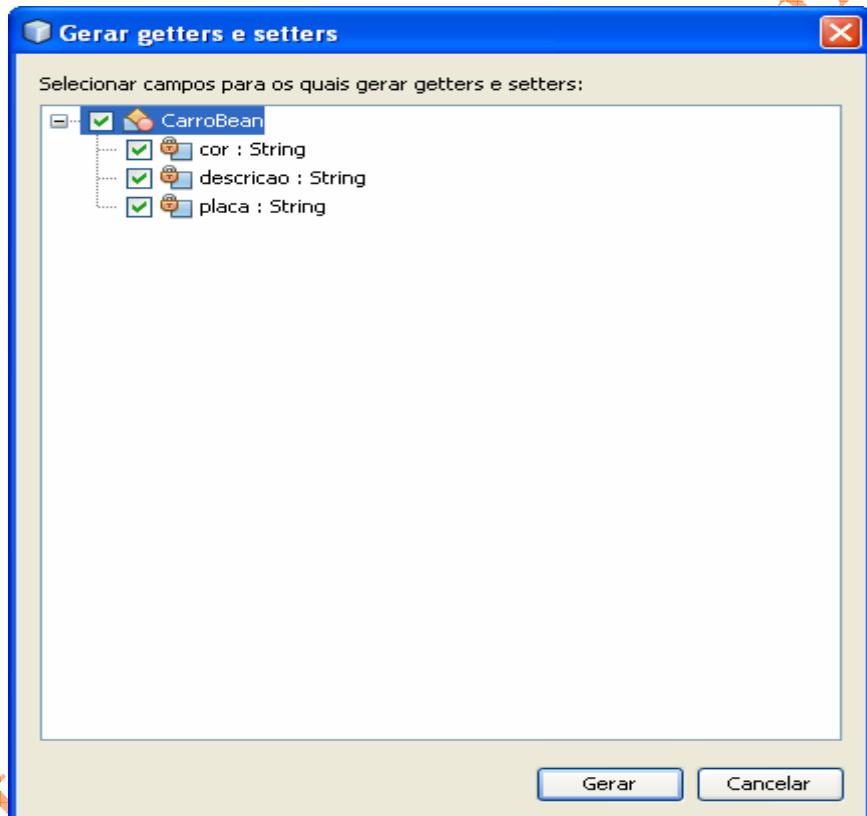
```
}
```

Dica: No netbeans podemos criar os `gets` e `sets` de forma rápida usando o procedimento a seguir.

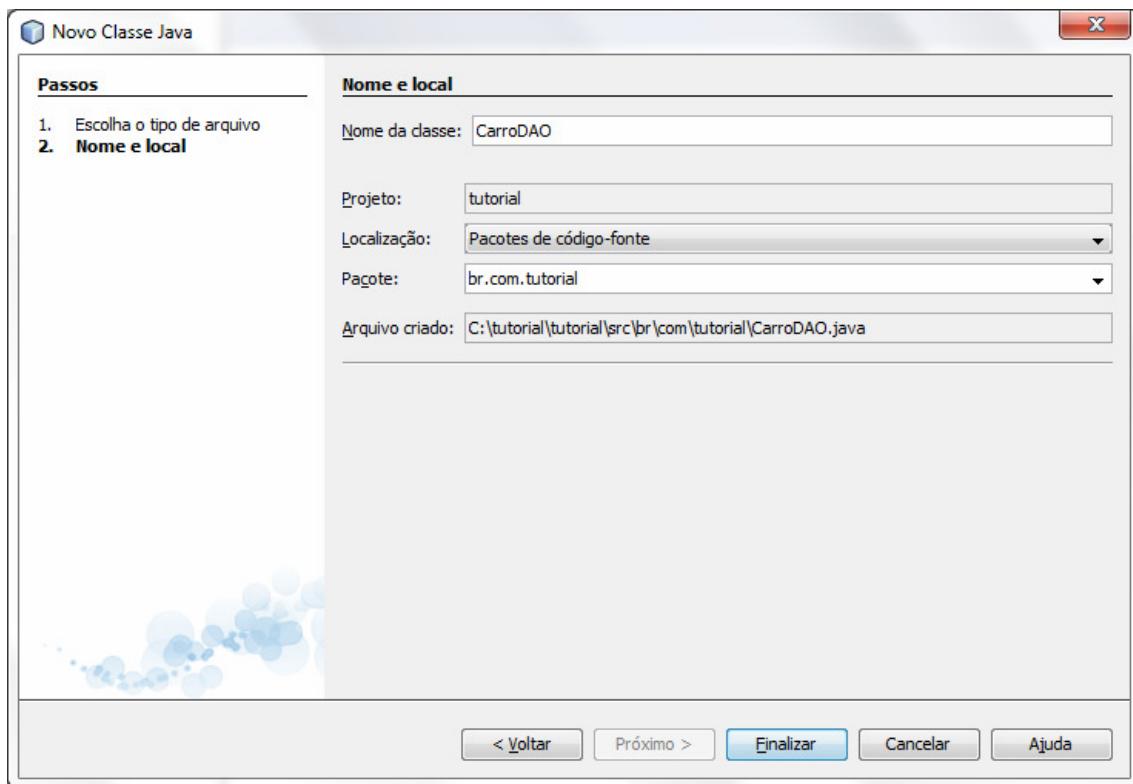
1 – pressione alt+insert e aparecerá a figura abaixo.



2 – Selecione “Getter e setter” e pressione enter. Quando aparecer a janela “Gerar getters e setters”, selecione os atributos conforme figura abaixo e click em gerar. Pronto! Todos os seus métodos de acesso aos atributos estão feitos.



Terceiro passo: Criação da classe CarroDAO. Para isso, click com o botão direito sobre o pacote criado anteriormente, quando uma nova janela aparecer (figura abaixo), coloque o nome CarroDAO e click em finalizar.



Uma classe será criada, devemos acrescentar os seguintes códigos nela. Essa classe será responsável pelo contato com o banco de dados nosso, ou seja, ela será responsável por incluir, alterar, pesquisar e excluir dados da tabela (CRUD).

Primeiro vamos criar a base da classe, com um atributo do tipo `Connection`, que será o responsável por guardar a conexão usada durante o processo. Os componentes da classe serão descritos a seguir.

```
package br.com.tutorial; //Este não é necessário, pois já é colocado automaticamente.
```

Precisamos importar dois pacotes, o `sql` e o `util`. O primeiro contém classes para manipulação banco de dados e o segundo contém classes para trabalharmos com `collections`.

```
import java.sql.*;  
import java.util.*;
```

```
public class CarroDAO {
```

Criando o atributo da classe `Connection`.

```
private Connection con;
```

Vamos criar agora o construtor da classe que receberá como argumento uma conexão e a passará ao nosso atributo `con`.

```
public CarroDAO(Connection con) {  
    setCon(con);  
}
```

Podemos, então criar os métodos get e set para o nosso atributo con.

```
public Connection getCon() {  
    return con;  
}  
public void setCon(Connection con) {  
    this.con = con;  
}
```

Precisamos fechar a classe.

```
}
```

Bom, agora que temos a nossa classe base, podemos criar os métodos CRUD. Começaremos com o método de inserção (inserir).

Este método receberá um objeto CarroBean como parâmetro e fará a persistência deste no banco de dados, isto é, fará a gravação efetiva desses dados na nossa tabela carro. Cabe observar que não está no escopo deste tutorial tratar de transações em banco de dados, não nos preocuparemos com isso aqui, no futuro em outro tutorial tratarei disso.

Vamos criar a assinatura do método, que tem como retorno uma String para avisar ao usuário se tudo ocorreu bem ou se deu algum erro.

```
public String inserir(CarroBean carro) {
```

Precisamos agora de uma variável do tipo String para passarmos o comando sql para o banco. Observe que, com exceção das interrogações (que serão substituídas pelos valores dos campos), são comandos padrões de SQL que, para o Java é apenas uma String.

```
String sql = "insert into  
carro(placa, codigo, cor, descricao) values(?, ?, ?, ?)";
```

Ainda dentro do método (inserir), vamos abrir um bloco *try* colocar o código para enviar dados para o banco. Esse tratamento de exceção é necessário porque pode ocorrer a seguinte exceção: SQLException.

```
try {
```

Agora precisamos preparar o nosso comando para enviar ao banco. Para isso usaremos a interface PreparedStatement.

```
PreparedStatement ps = getCon().prepareStatement(sql);
```

Após preparar o comando, podemos substituir as interrogações pelos valores dos campos correspondentes. A sequência aqui é muito importante, pois a primeira interrogação corresponde ao campo número 1, a segunda, ao campo número 2 e assim por diante.

```
ps.setString(1, carro.getPlaca());
ps.setInt(2, carro.getCodigo());
ps.setString(3, carro.getCor());
ps.setString(4, carro.getDescricao());
```

Feito isso vamos efetivar a inserção no banco, para isso podemos usar o método `execute` (retorna um boolean) ou `executeUpdate` (retorna um inteiro com o número de linhas afetadas pela operação), vamos usar o segundo.

Vamos executar o comando e fazer o teste ao mesmo tempo para verificar se alguma linha foi afetada para podermos retornar uma mensagem de sucesso ou insucesso.

```
if (ps.executeUpdate() > 0) {
    return "Inserido com sucesso.";
} else {
    return "Erro ao inserir";
}
```

Como nós abrimos um bloco `try`, devemos ter pelo menos um `catch` ou `finally` para finalizá-lo, no nosso caso teremos 1 `catch` para `SQLException` (exceções que podem ocorrer durante a conexão com o banco).

```
} catch (SQLException e) {
    return e.getMessage();
}
```

Podemos fechar o método, então.

```
}
```

O próximo método é o de alteração (alterar).

Este método receberá um objeto CarroBean como parâmetro e fará a alteração dos dados na nossa tabela carro. Este método não tem diferenças significativas em relação ao método inserir, as únicas exceções são a String SQL e a sequência dos parâmetros. Lembre-se, o campo chave (placa) não pode ser alterado.

```
public String alterar(CarroBean carro) {
    String sql = "update carro set codigo = ?, cor =
?, descricao = ?";
    sql += " where placa = ?";
```

```

try {
    PreparedStatement ps =
getCon().prepareStatement(sql);

    ps.setInt(1, carro.getCodigo());
    ps.setString(2, carro.getCor());
    ps.setString(3, carro.getDescricao());
    ps.setString(4, carro.getPlaca());

    if (ps.executeUpdate() > 0) {
        return "Alterado com sucesso.";
    } else {
        return "Erro ao alterar";
    }
} catch (SQLException e) {
    return e.getMessage();
}
}

```

O próximo método é o de exclusão (excluir).

Este método receberá um objeto CarroBean como parâmetro e fará a exclusão dos dados na nossa tabela carro. Este método não tem diferenças significativas em relação ao método alterar, as únicas exceções são a String SQL e que ele só tem um parâmetro.

```

public String excluir(CarroBean carro) {
    String sql = "delete from carro where placa = ?";

    try {
        PreparedStatement ps =
getCon().prepareStatement(sql);

        ps.setString(1, carro.getPlaca());

        if (ps.executeUpdate() > 0) {
            return "Excluído com sucesso.";
        } else {
            return "Erro ao excluir";
        }
    } catch (SQLException e) {
        return e.getMessage();
    }
}

```

E agora vamos ao ultimo método, `IstarTodos` (**deixo a cargo do leitor o aperfeiçoamento deste método para pesquisar por parâmetro**). Ele tem várias diferenças dos demais, por isso será explicado em detalhe.

Precisamos criar um método que retorne uma coleção de objetos, pois ele irá ao banco e trará todos os dados que se encontram na tabela carro. Para isso usaremos uma implementação da interface `List`, a `ArrayList`. A escolha por

tal implementação é meramente didática, sem levar em consideração as vantagens ou desvantagens dela. Dessa forma, a assinatura do método fica assim:

```
public List<CarroBean> listarTodos() {
```

Como vamos listar todas as linhas da tabela, a nossa String SQL fica bem simples.

```
String sql = "select * from carro ";
```

Vamos criar uma lista de carros para armazenar os objetos carros que retornarão da nossa consulta.

```
List<CarroBean> listaCarro = new ArrayList<CarroBean>();
```

As duas linhas a seguir já foram explicadas anteriormente.

```
try {
    PreparedStatement ps =
getCon().prepareStatement(sql);
```

Agora precisamos de um objeto da classe `ResultSet` para armazenar o resultado vindo do banco. Observe que, diferentemente dos outros três métodos, neste usamos o `executeQuery` para efetivar a nossa operação.

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

Um objeto `ResultSet` tem acesso ao método `next` que permite percorrer todos os dados nele contido. Porém se a consulta não retornar dados e tentarmos usá-lo, pode ocorrer uma exceção, por isso é necessário fazer essa verificação antes.

```
if (rs != null) {
```

Se a consulta retornou dados, podemos percorrê-los e fazer a atribuição a um objeto e guardar este na nossa lista. Dentro no laço (`while`) um objeto `CarroBean` é criado e os seus atributos são preenchidos com os dados dos campos correspondentes da tabela. Observe que os números dentro dos parênteses correspondem à ordem dos campos da tabela, alternativamente podemos usar os nomes dos campos.

```
while (rs.next()) {
    CarroBean cb = new CarroBean();
    cb.setPlaca(rs.getString(1));
    cb.setCodigo(rs.getInt(2));
    cb.setCor(rs.getString(3));
    cb.setDescricao(rs.getString(4));
    listaCarro.add(cb);
}
return listaCarro;
```

```
} else {
    return null;
}
```

As linhas a seguir já foram explicadas anteriormente.

```
        } catch (SQLException e) {
            return null;
        }
    }
```

Procedendo da mesma forma que procedemos para criar as classes ProprietarioBean e ProprietarioDAO. O código é mostrado a seguir (o código completo de CarroBean, CarroDAO e Conexao também é colocado a seguir).

```
package br.com.tutorial;

import java.sql.*;
import java.util.List;

public class Teste {
    public static void main(String[] args) {
        //Criando conexão com o banco
        Connection con = Conexao.abrirConexao();

        //Criando objetos das classes ProprietarioBean e
ProprietarioDAO
        ProprietarioBean pb = new ProprietarioBean();
        ProprietarioDAO pd = new ProprietarioDAO(con);

        //Atribuindo valor para os campos
        /*pb.setNome("João");
        pb.setCidade("Brasília");

        pb.setNome("Maria");
        pb.setCidade("Brasília");*/
        //Testando o método de inserção
        //System.out.println(pd.inserir(pb));

        //Atribuindo valor para os campos
        /*pb.setCodigo(2);
        pb.setNome("Maria");
        pb.setCidade("Rio de Janeiro");*/

        //Testando o método de alteração
        //System.out.println(pd.alterar(pb));

        //Atribuindo valor para o campo
        // pb.setCodigo(2);

        //Testando Exclusão
    }
}
```

```

//System.out.println(pd.excluir(pb));

//Testando pesquisa
List<ProprietarioBean> lp = pd.listarTodos();
for(ProprietarioBean p : lp){
    System.out.println("Codigo: "+p.getCodigo());
    System.out.println("Nome: "+p.getNome());
    System.out.println("Cidade: "+p.getCidade());
}

}

package br.com.tutorial;

public class ProprietarioBean {
    private Integer codigo;
    private String nome;
    private String cidade;

    public String getCidade() {
        return cidade;
    }

    public void setCidade(String cidade) {
        this.cidade = cidade;
    }

    public Integer getCodigo() {
        return codigo;
    }

    public void setCodigo(Integer codigo) {
        this.codigo = codigo;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}

package br.com.tutorial;

import java.sql.*;
import java.util.*;

```

```

public class ProprietarioDAO {
    private Connection con;

    public ProprietarioDAO(Connection con) {
        setCon(con);
    }

    public String inserir(ProprietarioBean proprietario) {
        String sql = "insert into proprietario(nome,cidade)values(?,?)";

        try {
            PreparedStatement ps = getCon().prepareStatement(sql);
            ps.setString(1, proprietario.getNome());
            ps.setString(2, proprietario.getCidade());

            if (ps.executeUpdate() > 0) {
                return "Inserido com sucesso.";
            } else {
                return "Erro ao inserir";
            }
        } catch (SQLException e) {
            return e.getMessage();
        }
    }

    public String alterar(ProprietarioBean proprietario) {
        String sql = "update proprietario set nome = ?,cidade = ?";
        sql += " where codigo = ?";

        try {
            PreparedStatement ps = getCon().prepareStatement(sql);
            ps.setString(1, proprietario.getNome());
            ps.setString(2, proprietario.getCidade());
            ps.setInt(3, proprietario.getCodigo());

            if (ps.executeUpdate() > 0) {
                return "Alterado com sucesso.";
            } else {
                return "Erro ao alterar";
            }
        } catch (SQLException e) {
            return e.getMessage();
        }
    }
}

```

```

public String excluir(ProprietarioBean proprietario) {
    String sql = "delete from proprietario where codigo
= ?";

    try {
        PreparedStatement ps =
getCon().prepareStatement(sql);

        ps.setInt(1, proprietario.getCodigo());

        if (ps.executeUpdate() > 0) {
            return "Excluído com sucesso.";
        } else {
            return "Erro ao excluir";
        }
    } catch (SQLException e) {
        return e.getMessage();
    }
}

public List<ProprietarioBean> listarTodos() {
    String sql = "select * from proprietario ";
    List<ProprietarioBean> listaCarro = new
ArrayList<ProprietarioBean>();
    try {
        PreparedStatement ps =
getCon().prepareStatement(sql);

        ResultSet rs = ps.executeQuery();

        if (rs != null) {
            while (rs.next()) {
                ProprietarioBean pb =
new
ProprietarioBean();
                pb.setCodigo(rs.getInt(1));
                pb.setNome(rs.getString(2));
                pb.setCidade(rs.getString(3));
                listaCarro.add(pb);
            }
        }
        return listaCarro;
    } else {
        return null;
    }
} catch (SQLException e) {
    return null;
}
}

public Connection getCon() {
    return con;
}

```

```

        public void setCon(Connection con) {
            this.con = con;
        }

    }

package br.com.tutorial;

public class CarroBean {

    private String placa;
    private String cor;
    private String descricao;
    private Integer codigo;

    public Integer getCodigo() {
        return codigo;
    }

    public void setCodigo(Integer codigo) {
        this.codigo = codigo;
    }

    public String getCor() {
        return cor;
    }

    public void setCor(String cor) {
        this.cor = cor;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public String getPlaca() {
        return placa;
    }

    public void setPlaca(String placa) {
        this.placa = placa;
    }
}

```

```

package br.com.tutorial;

import java.sql.*;
import java.util.*;

public class CarroDAO {

    private Connection con;

    public CarroDAO(Connection con) {
        setCon(con);
    }

    public String inserir(CarroBean carro) {
        String sql = "insert carro(placa,codigo,cor,descricao)values(?, ?, ?, ?)";
        try {
            PreparedStatement ps = getCon().prepareStatement(sql);
            ps.setString(1, carro.getPlaca());
            ps.setInt(2, carro.getCodigo());
            ps.setString(3, carro.getCor());
            ps.setString(4, carro.getDescricao());

            if (ps.executeUpdate() > 0) {
                return "Inserido com sucesso.";
            } else {
                return "Erro ao inserir";
            }
        } catch (SQLException e) {
            return e.getMessage();
        }
    }

    public String alterar(CarroBean carro) {
        String sql = "update carro set codigo = ?, cor = ? ,descricao = ? ";
        sql += " where placa = ?";

        try {
            PreparedStatement ps = getCon().prepareStatement(sql);
            ps.setInt(1, carro.getCodigo());
            ps.setString(2, carro.getCor());
            ps.setString(3, carro.getDescricao());
            ps.setString(4, carro.getPlaca());
        }
    }
}

```

```

        if (ps.executeUpdate() > 0) {
            return "Alterado com sucesso.";
        } else {
            return "Erro ao alterar";
        }
    } catch (SQLException e) {
        return e.getMessage();
    }
}

public String excluir(CarroBean carro) {
    String sql = "delete from carro where placa = ?";

    try {
        PreparedStatement ps = getCon().prepareStatement(sql);

        ps.setString(1, carro.getPlaca());

        if (ps.executeUpdate() > 0) {
            return "Excluído com sucesso.";
        } else {
            return "Erro ao excluir";
        }
    } catch (SQLException e) {
        return e.getMessage();
    }
}

public List<CarroBean> listarTodos() {
    String sql = "select * from carro ";
    List<CarroBean> listaCarro = new ArrayList<CarroBean>();
    try {
        PreparedStatement ps = getCon().prepareStatement(sql);

        ResultSet rs = ps.executeQuery();

        if (rs != null) {
            while (rs.next()) {
                CarroBean cb = new CarroBean();
                cb.setPlaca(rs.getString(1));
                cb.setCodigo(rs.getInt(2));
                cb.setCor(rs.getString(3));
                cb.setDescricao(rs.getString(4));

                listaCarro.add(cb);
            }
        }
    }
}

```

```

        return listaCarro;
    } else {
        return null;
    }

} catch (SQLException e) {
    return null;
}
}

public Connection getCon() {
    return con;
}

public void setCon(Connection con) {
    this.con = con;
}
}

package br.com.tutorial;

import java.sql.*; //Importação do pacote que contém a classe Connection
public class Conexao { //Nome da classe
    public static Connection abrirConexao() {
        Connection con = null;
        try {

Class.forName("com.mysql.jdbc.Driver").newInstance();
            String url = "";
            url += "jdbc:mysql://127.0.0.1/tutorial?";
            url += "user=root&password=123";
            con = DriverManager.getConnection(url);
            System.out.println("Conexão aberta.");
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        } catch (ClassNotFoundException e) {
            System.out.println(e.getMessage());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return con;
    }

    public static void fecharConexao(Connection con) {
        try {
            con.close();
            System.out.println("Conexão fechada.");
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
}

```

```
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Com isso temos todas as classes necessárias, só nos resta fazer um teste para ver se estão funcionando realmente. Para verificar se as classes estão executando corretamente, crie uma classe chamada Teste e faça um teste de todas as operações.

Lembre-se, insira dados primeiro na tabela proprietário e depois na tabela carro porque o campo "codigo" da tabela carro é uma chave estrangeira e precisa existir na tabela de origem. Veja como testar no parte I deste tutorial (Criando um aplicação simples com JAVA e MySQL usando NetBeans – Parte I), que se encontra no site <http://www.pusivus.com.br> na seção de tutoriais.

Neste momento temos cinco classes que são totalmente independentes da interface que vamos usar para criar seus objetos. Dessa forma, podemos tanto usar swing como jsp ou outra abordagem qualquer.

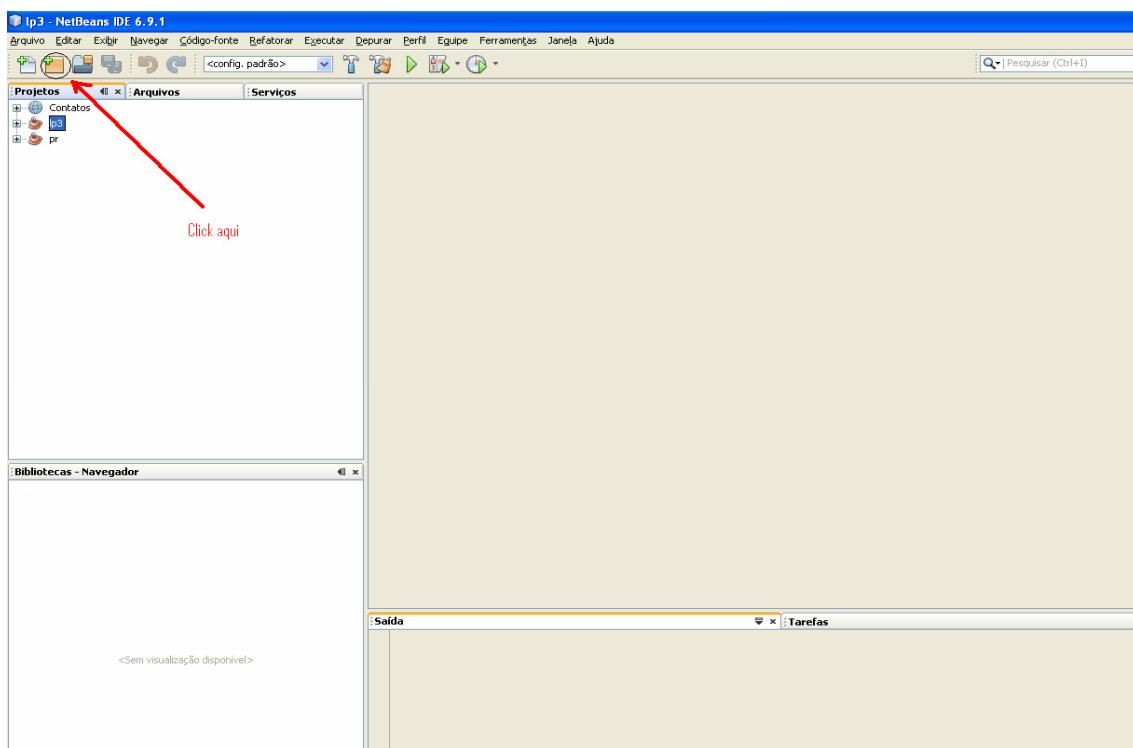
Neste tutorial não usaremos todos os recursos do netbeans para criação de aplicação web, usaremos apenas os recursos de criação de arquivos JSP. Cabe ainda ressaltar que não estamos usando nenhuma arquitetura específica, tipo MCV, por exemplo, estamos fazendo apenas aplicação simples usando recursos do Java e JSP. Embora o conhecimento adquirido com este conjunto de tutoriais seja suficiente para desenvolver pequenas aplicações, recomendo que você aprofunde seus estudos em Java e JSP para melhorar seu desempenho profissional.

Pode parecer estranho, mas já tive que fazer manutenção em sistemas (em uso em empresa de grande porte) feitos de forma muito mais amadora do que o que estamos fazendo aqui. Bom, chega de conversa, vamos em frente.

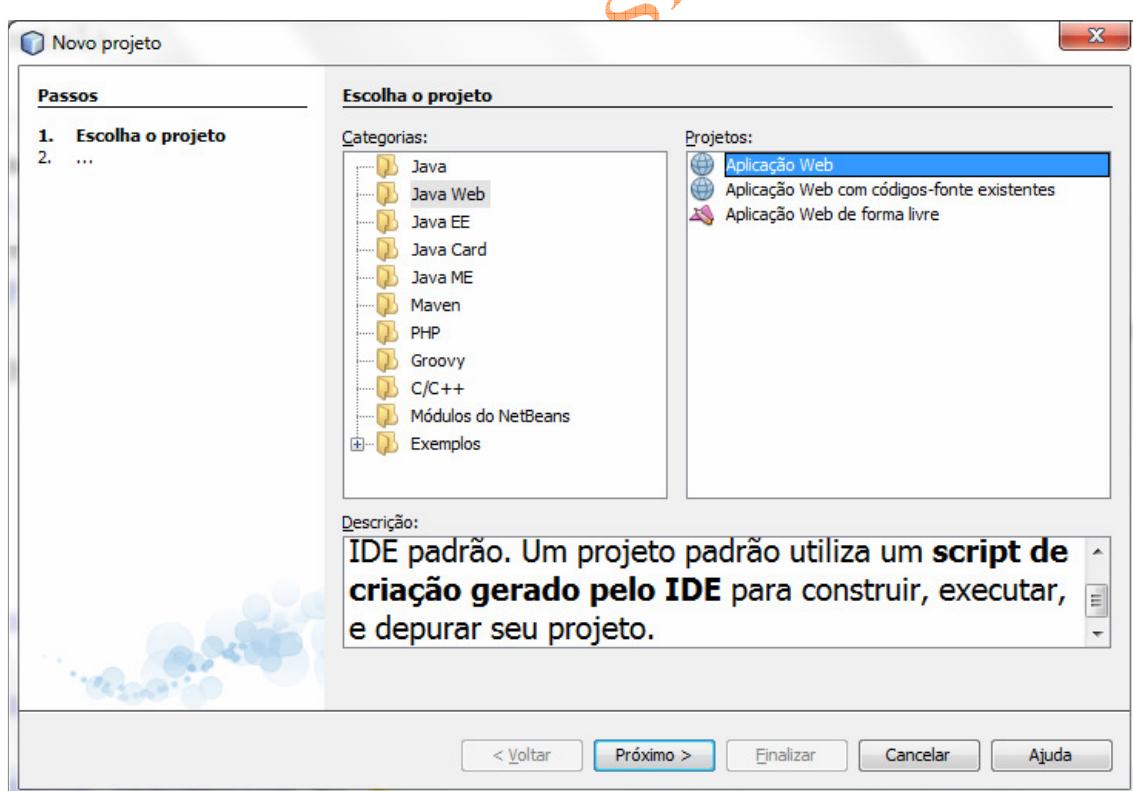
11. Criando o projeto web

Para criação do projeto web siga as orientações abaixo. Não rode o projeto diretamente no netbeans, use o browser, como é mostrado neste tutorial.

Primeiro passo: click no ícone novo projeto, conforme mostrado na figura abaixo.

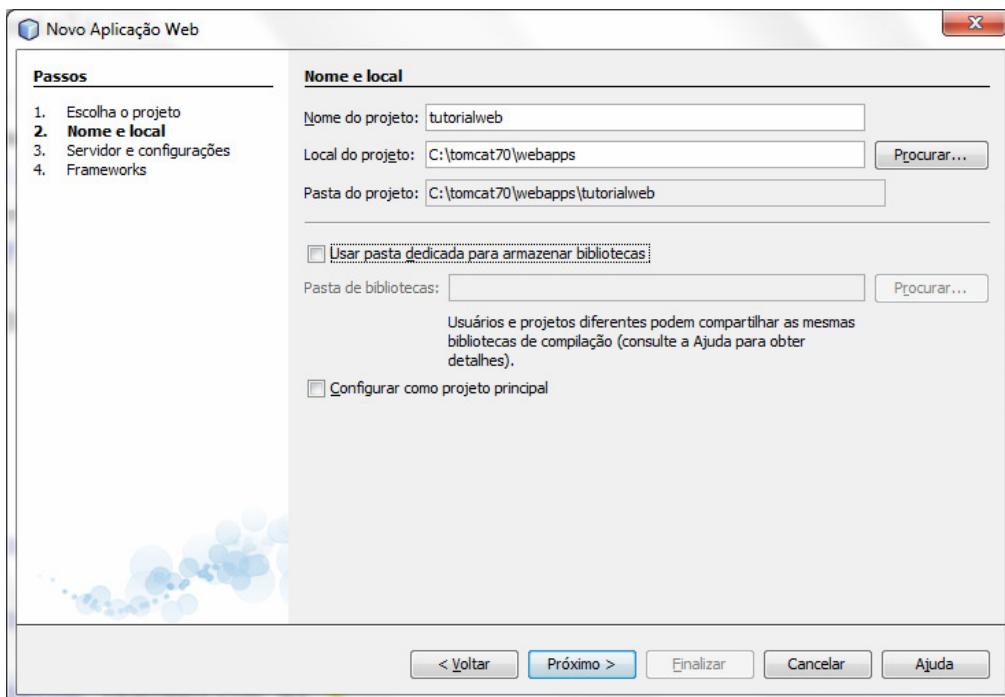


Segundo passo: ao aparecer uma nova janela, selecione "Java Web" em categorias e "Aplicação Web" em Projetos e click em "próximo" (next).

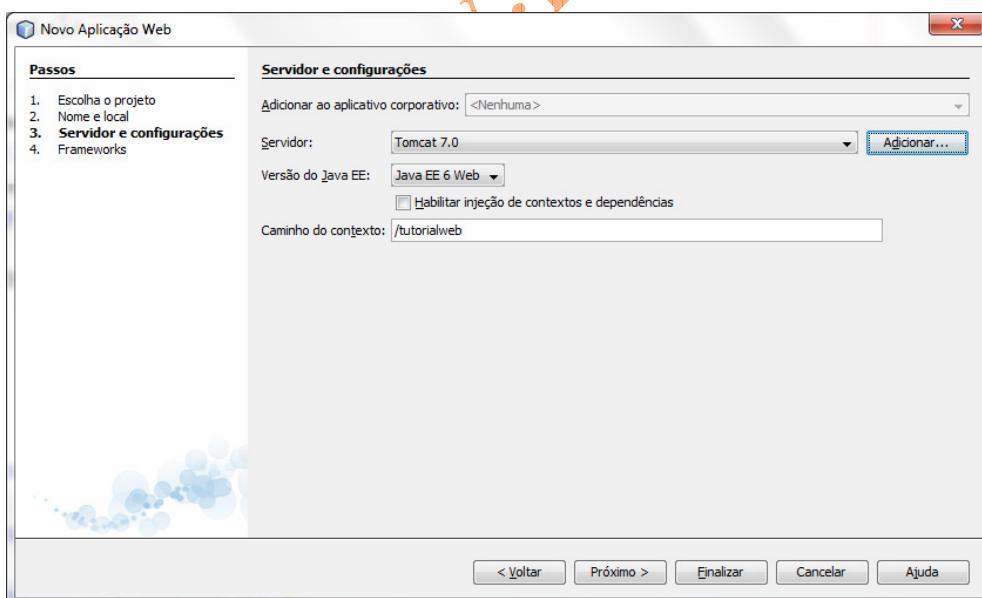


Terceiro passo: ao aparecer a nova janela, figura abaixo, escreva tutorialweb no campo "Nome do Projeto" e click em procurar e localize a pasta webapps do tomcat (c:\arquivos de programas\Apache Software\Tomcat x.x\webapps). Desmarque os dois checkbox, a sua janela deverá se parecer

com a figura abaixo, com diferença do caminho onde se encontra o tomcat, pois eu instalei diretamente na raiz (C:). Click em "próximo" (next).



Quarto passo: escolha o tomcat para servidor, escolha a versão 5 ou 6 para a versão Java EE e desmarque (se estiver marcado) o checkbox e click em próximo (next). A sua janela deverá se parecer a figura abaixo.

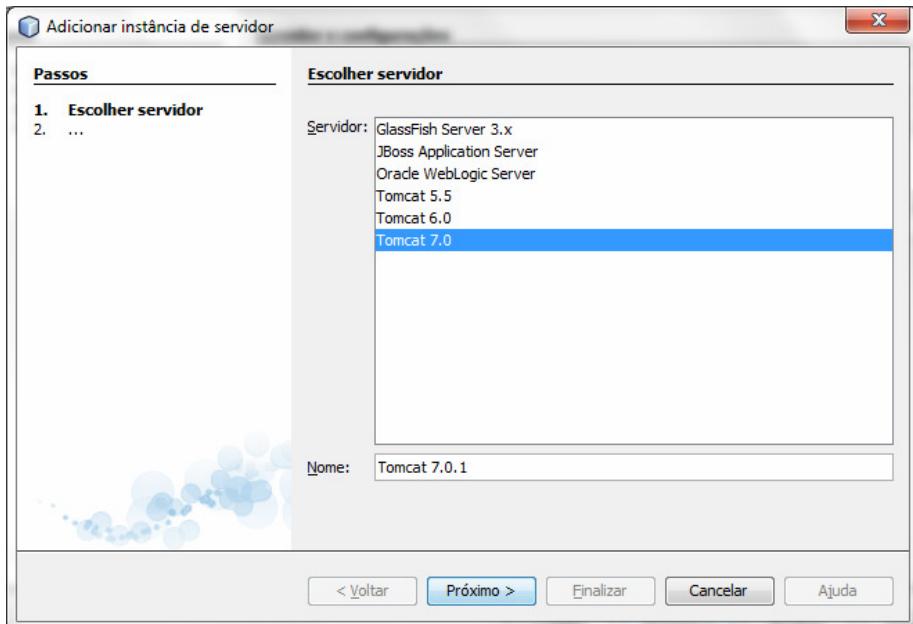


Se o tomcat não aparecer na lista, siga os passos abaixo para adicioná-lo.

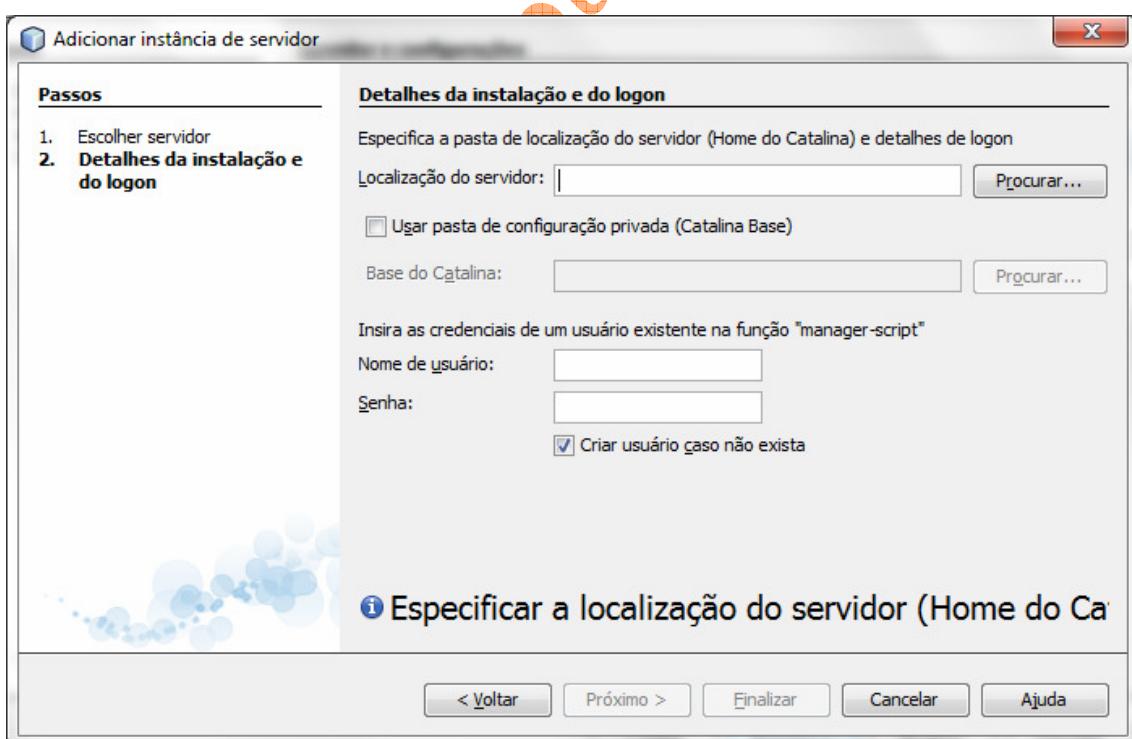
11.1. Adicionando o tomcat ao netbeans

a) click no botão adicionar, figura anterior.

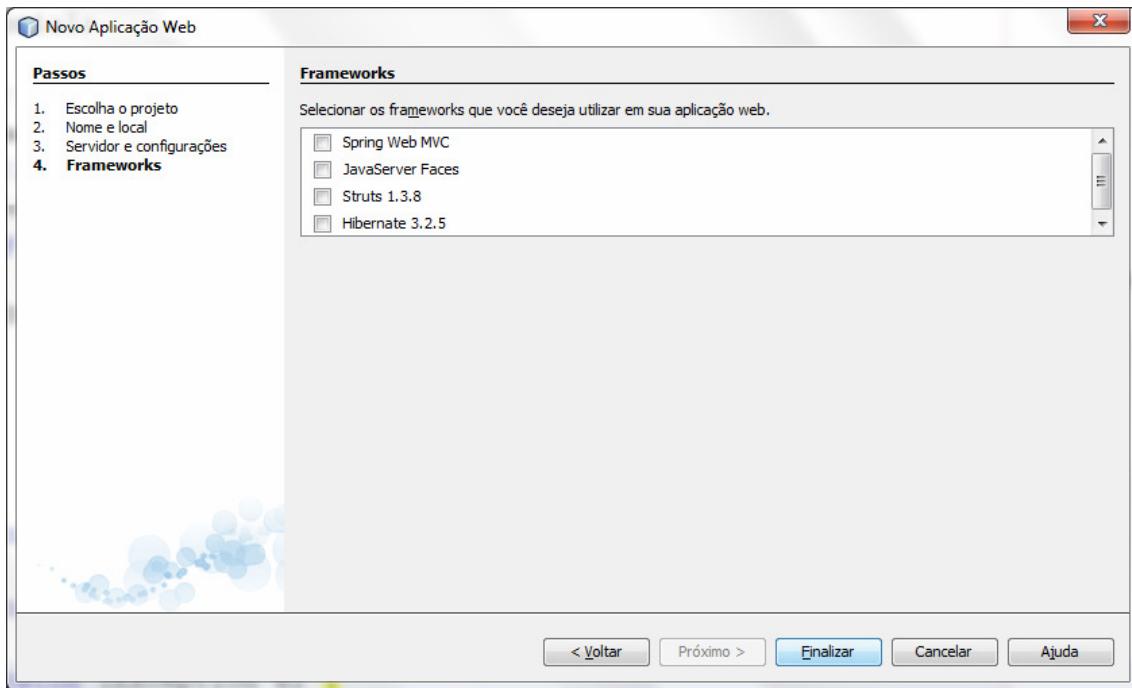
Segundo passo: quando a nova janela aparecer, selecione o tomcat que estiver instalado na sua máquina, figura abaixo. Click em próximo (next).



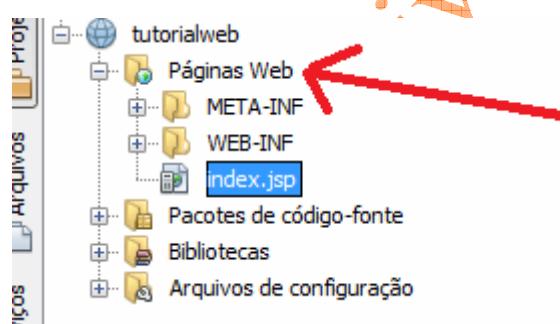
b) na próxima janela, click em procurar, localize a pasta onde está instalada o tomcat, dê um nome de usuário e senha e click em finalizar.



Quinto passo: ao aparecer a janela da figura abaixo, click em finalizar, sem escolher framework algum.



Pronto, o seu projeto web está criado, preste atenção na pasta "Páginas Web", figura abaixo, pois é a única que vamos usar.



12. Adicionando as classes ao projeto web

Até o momento temos nossas classes e nosso projeto web, o que precisamos agora é fazer com que eles se conheçam. Por padrão o tomcat procura nossos recursos (.class) em uma pasta chamada classes, que fica dentro da pasta WEB-INF (que deve estar na raiz do nosso projeto), pois bem, como você observa na figura acima, existe uma pasta WEB-INF, porém, ela não está na raiz do projeto, por isso, precisamos criar tal pasta para colocarmos nossos recursos nela. Siga os passos abaixo para executar esta tarefa.

Primeiro passo: vá até a pasta do projeto (c:\arquivos de programas\Apache Software\Tomcat x.x\webapps\tutorialweb), como mostra a figura abaixo.

	Nome	Data de modificaç...	Tipo	Tamanho
▷	nbproject	28/05/2011 13:19	Pasta de arquivos	
▷	src	28/05/2011 13:19	Pasta de arquivos	
▷	web	28/05/2011 13:19	Pasta de arquivos	
▷	build	28/05/2011 13:19	Documento XML	4 KB

Segundo passo: crie, dentro da pasta tutorialweb, a pasta WEB-INF (tem que ser em maiúscula), como mostra a figura abaixo.

	Nome	Data de modificaç...	Tipo	Tamanho
▷	nbproject	28/05/2011 13:19	Pasta de arquivos	
▷	src	28/05/2011 13:19	Pasta de arquivos	
▷	web	28/05/2011 13:19	Pasta de arquivos	
▷	WEB-INF	28/05/2011 14:06	Pasta de arquivos	
▷	build	28/05/2011 13:19	Documento XML	4 KB

Terceiro passo: copie a pasta classes para dentro da pasta WEB-INF (criada no passo anterior), que se encontra na pasta build (C:\tutorial\tutorial\build) do projeto criado no tópico 7 deste tutorial, veja figura abaixo.

	Nome	Data de modificaç...	Tipo
▷	classes	27/05/2011 20:56	Pasta de arquivos
▷	empty	27/05/2011 20:55	Pasta de arquivos
▷	generated-sources	27/05/2011 20:55	Pasta de arquivos

Obs.:

1 - o netbeans cria uma pasta chamada src para guardar os arquivos .java e outra chamada build para guardar os arquivos .class. Porém, a pasta build só é criada quando compilamos e rodamos uma classe no nosso projeto. Por isso, é importante que você crie a classe Teste (com o método main) para testar as classes, pois além de, com o teste, termos certeza de que as nossas classes estão funcionando direito, criamos, de fato, as nossas ".class", que são as que nos interessam. Veja nas duas figuras a seguir, os nossos arquivos.

	Nome	Data de modificação	Tipo
•	CarroBean.class	27/05/2011 20:56	Arquivo CLASS
•	CarroDAO.class	27/05/2011 20:56	Arquivo CLASS
•	Conexao.class	27/05/2011 20:56	Arquivo CLASS
•	ProprietarioBean.class	27/05/2011 20:56	Arquivo CLASS
•	ProprietarioDAO.class	27/05/2011 20:56	Arquivo CLASS
•	Teste.class	27/05/2011 21:47	Arquivo CLASS

Arquivos .class

	Nome	Data de modificação	Tipo
•	CarroBean	27/05/2011 20:46	Arquivo JAVA
•	CarroDAO	27/05/2011 20:55	Arquivo JAVA
•	Conexao	27/05/2011 17:14	Arquivo JAVA
•	ProprietarioBean	27/05/2011 20:21	Arquivo JAVA
•	ProprietarioDAO	27/05/2011 20:34	Arquivo JAVA
•	Teste	27/05/2011 21:47	Arquivo JAVA

Arquivos .java

2 - o pacote (br.com.tutorial) que criamos no netbeans se transformam em três pastas na seguinte hierarquia br -> com -> tutorial, como pode ser visto na figura acima.

Quarto passo (opcional): entre na pasta tutorial da WEB-INF, como mostra a figura abaixo e apague a o arquivo Teste.class, pois não precisamos dele.

	Nome	Data de modificação	Tipo
•	CarroBean.class	27/05/2011 20:56	Arquivo CLASS
•	CarroDAO.class	27/05/2011 20:56	Arquivo CLASS
•	Conexao.class	27/05/2011 20:56	Arquivo CLASS
•	ProprietarioBean.class	27/05/2011 20:56	Arquivo CLASS
•	ProprietarioDAO.class	27/05/2011 20:56	Arquivo CLASS
•	Teste.class	27/05/2011 21:47	Arquivo CLASS

Teste.class

Quinto passo: dê um reload na sua aplicação, como mostra a figura abaixo, para que o tomcat "veja" as suas modificações. Se você estiver em dúvida de como obter a tela da figura abaixo, volte ao item 4 deste tutorial.

/pusivus1	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/teste	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/tutorialweb	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

13. Criando as páginas JSP

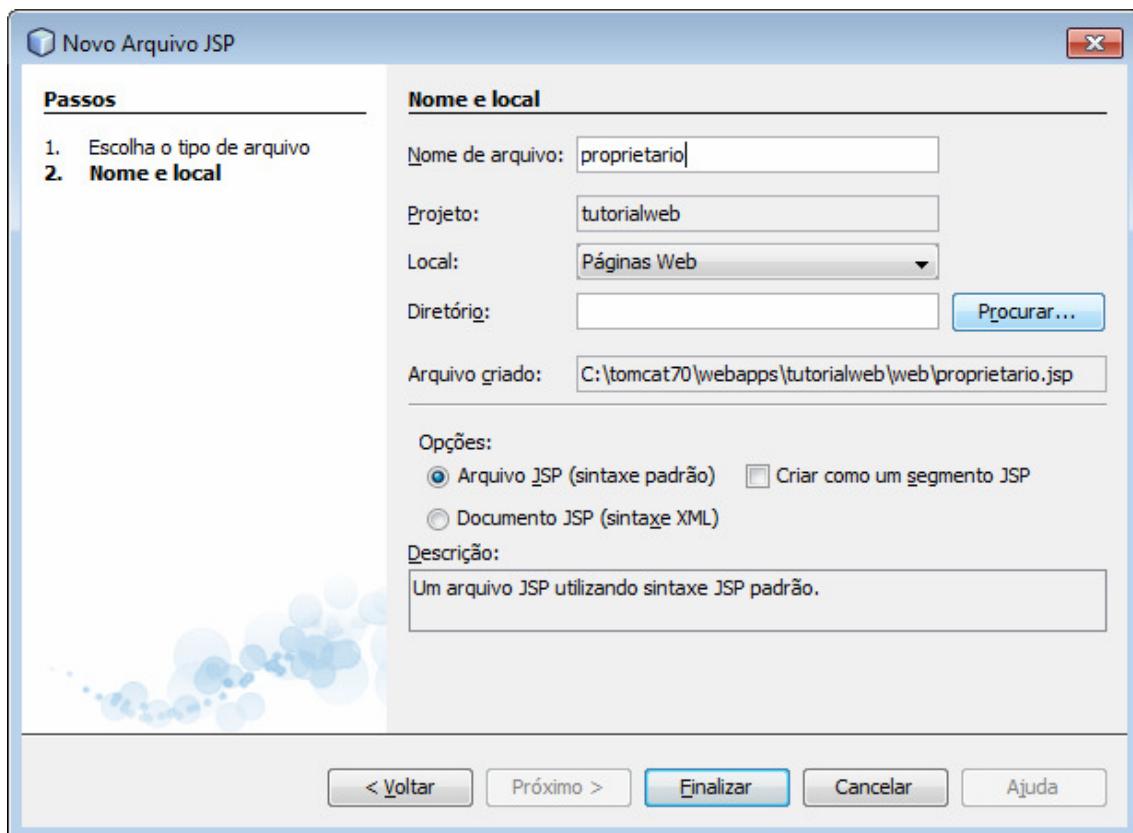
Finalmente, estamos prontos para criar a interface web para nossas classes. Vamos criar quatro páginas web, duas para o formulário de entrada de dados de proprietário e carro e duas para executar as operações CRUD¹.

13.1. Criando o formulário para proprietário

Para criação do formulário, siga os passos apresentados abaixo.

Primeiro passo: com o botão direito do mouse sobre a pasta "Páginas Web" click em novo -> JSP, a figura abaixo irá aparecer. Coloque o nome de proprietario e click em finalizar.

¹ CRUD significa Create (criar) Retrieve (recuperar) Update (atualizar) Delete (excluir). Basicamente corresponde a Insert, Select , Update e Delete no banco.



O netbeans cria um esqueleto da página como mostra a figura abaixo. Não faz parte do escopo deste tutorial explicar cada parte que componente da página.

Antes de ir para o segundo passo precisamos verificar se o nosso servidor web (tomcat) consegue localizar nosso pacote e nossas classes. Para tanto, vamos fazer a importação do nosso pacote e testar a conexão. Siga o procedimento abaixo.

a) Coloque, na linha 6 do arquivo mostrado na figura acima, a seguinte linha:

```
<%@page import="br.com.tutorial.* , java.sql.*" %>
```

Essa é a forma de fazer a importação dos pacotes para uma página JSP. Não se preocupe com os erros apontados pelo netbeans, pois ele não sabe onde está o nosso pacote, mas o tomcat sabe, pois contamos só para e é isso que nos interessa.

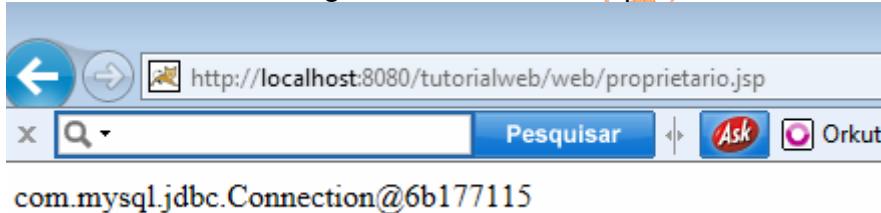
b) Apague a linha 15 (`<h1>Hello World!</h1>`) e coloque os comandos (JSP) abaixo:

```
<%  
    //Criando um objeto de conexão  
    Connection con = Conexao.abrirConexao();  
    //Exibindo o objeto no browser  
    out.print(con);  
  
    //Fechando a conexão  
    Conexao.fecharConexao(con);  
%>
```

c) Abra o browser e digite o comando abaixo.

`http://localhost:8080/tutorialweb/web/proprietario.jsp`

Se tudo estiver ok, a figura abaixo deverá aparecer.



Nota: os comandos JSP devem ser colocados dentro de scriptlets (`<% %>`). Os demais elementos são mostrados na tabela abaixo. Reforço que este não é um curso de JSP e sim um tutorial usando tal linguagem, portanto, não explicarei detalhe da linguagem.

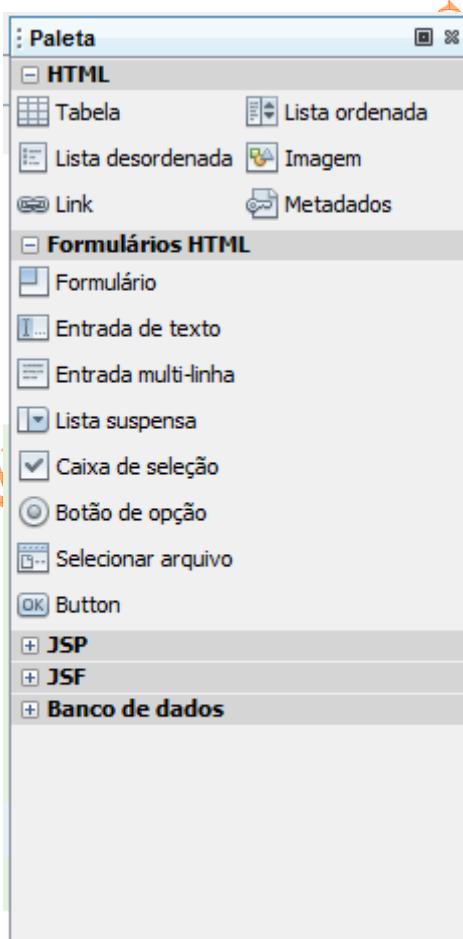
Elementos do JSP

As páginas JSPs podem conter uma série de elementos, dos quais merecem destaque:

Elemento	Exemplos	Descrição
Diretivas	<code><%@ page atributo="valor"%></code>	Define propriedades gerais em um arquivo JSP
Scriptlets	<code><% ... %></code>	Define trechos de códigos Java, a

		serem executados durante uma requisição.
Declarações	<%! ... %>	Declara variável e método.
expressões	<%= ... %>	Variáveis ou métodos java com retorno do tipo String enviados para uma página de saída (out)
ações	<jsp:.... ... />	Novas ações adicionadas à página c/ mecanismo de extensão de tags

Segundo passo: agora vamos criar o formulário propriamente dito. Para isso, você pode usar os recursos visuais do netbeans, selecione o meu janela -> paleta ou Ctrl+Shift+8, com isso surgirá a paleta com os componentes conforme mostra a figura abaixo.



Terceiro passo: monte seu formulário como mostra a figura abaixo e o código a seguir.

Código

Nome

Cidade

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Tela para manutenção dos dados de
proprietários</title>
    </head>

    <body>
        <script language="javascript">
            //Função para limpar os campos do formulário
            function limpar(){
                with(document.form1){
                    codigo.value = '';
                    nome.value = '';
                    cidade.value = '';
                }
            }
        </script>
        <%
            //Criação das variáveis para receber os dados
            //para preencher o formulário após a pesquisa
            //Para cada parâmetro passado é feito um teste
            //para verificar se esse veio nulo, se veio nulo,
            //é atribuído valor em branco.
            String codigo = ((request.getParameter("codigo"))
                != null) ? (request.getParameter("codigo")) : "0";
            String nome = ((request.getParameter("nome"))
                != null) ? (request.getParameter("nome")) : "";
            String cidade = ((request.getParameter("cidade"))
                != null) ? (request.getParameter("cidade")) : "";
        %>
    
```

```

        <!--Criação do formulário que enviará os dados para
a página proprietarioOp.jsp
        usando o método get-->
        <form    name="form1"    action="proprietarioOp.jsp"
method="get"><!--Tag de abertura do form-->
            <table><!--Tag de abertura da tabela para
organizar os componentes-->
                <tr>
                    <td>Código</td>
                    <td><input type="text"    name="codigo"
size="5" maxlength="5" value="<%.codigo%>" /></td>
                </tr>
                <tr>
                    <td>Nome</td>
                    <td><input type="text"    name="nome"
size="50" maxlength="50" value="<%.nome%>" /></td>
                </tr>
                <tr>
                    <td>Cidade</td>
                    <td><input type="text"    name="cidade"
size="30" maxlength="30" value="<%cidade%>" /></td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit"    name="crud"
value="Incluir"/>
                        <input type="submit"    name="crud"
value="Alterar"/>
                        <input type="submit"    name="crud"
value="Excluir"/>
                        <input type="submit"    name="crud"
value="Pesquisar"/>
                        <input type="button"    value="Limpar"
onclick="limpar()"/>
                    </td>
                </tr>
            </table><!--Tag para fechamento da tabela-->
        </form><!--Tag de fechamento do form-->
    </body>
</html>

```

Antes de prosseguir se faz necessário uma explicação a respeito do código acima (dentro do corpo do documento - <body></body>). Para tanto, o dividirei em 3 partes: o código javascript, o código jsp e o código html com jsp.

- Na primeira parte (<script language="javascript"></script>) temos apenas uma função para limpar os campos do formulário. Como pode ser observado, só estamos atribuindo valor vazio para cada campo. Cada campo html é formado da seguinte maneira: document.nome_do_form.nome_do_campo. Para atribuirmos valor a esses campos, usamos a "extensão" value, assim, para atribuirmos o valor vazio (em

branco) para o campo "codigo", precisamos fazer isto:
document.form1.codigo.value = ''.

Como document.form1 se repete em todos os campos, usamos o comando with (document.form1) para evitar essa repetição, ou seja, tudo que estiver dentro desse comando pode usar esse prefixo, dessa forma, quando colocarmos codigo.value, queremos executar o comando document.form1.codigo.value.

b) Na segunda parte (<% %>) temos os comandos jsp para teste e atribuição de valores. Para passarmos valores de um formulário a outro, precisamos usar o método getParameter que retorna uma String, por isso temos que trabalhar sempre com variáveis do tipo String. Mas, como faremos para mandar um valor inteiro ou em ponto flutuante para o banco? Devemos converter a String para o tipo desejado.

Para evitar exceções em nossos aplicativos, devemos ter o cuidado de não tentarmos converter um valor null ou vazio para inteiro ou double, ou mesmo usar o método equals para valores null. Por isso, devemos fazer em teste com os parâmetros passados para verificar se estão nulos, se estiverem, devemos atribuir vazio (se o atributo da nossa classe for String) ou zero (se o atributo da nossa classe for numérico). Os testes foram feitos usando o operador ternário do jsp, que tem a mesma sintaxe para o Java, C/C++.

A linha String codigo = ((request.getParameter("codigo") != null) ? (request.getParameter("codigo")) : "0"); pode ser substituída pelas seguintes linhas:

```
String codigo = "";
if (request.getParameter("codigo") != null){
    codigo = request.getParameter("codigo");
} else{
    codigo = "0";
}
```

Como se pode notar, o uso do operador ternário torna nosso código mais compacto. Porém, se você não se sente à vontade com ele, pode usar a outra maneira de fazer os testes. Cabe a mesma explicação para as linhas seguintes, exceto pelo fato de colocarmos vazio (porque correspondem a uma campo String em nossa classe) em vez de zero, quando o parâmetro vem nulo.

c) Na terceira parte (<form ...> </form>) temos uma mistura de código html com jsp. A necessidade disso se deve ao fato de que nosso formulário (proprietario.jsp) enviará dados para a página proprietarioOp.jsp e receberá dados deste. proprietario.jsp enviará dados para inclusão, alteração, exclusão e pesquisa para proprietarioOp.jsp e este enviará, de acordo com a escolha do usuário, dados de volta. Por isso, os valores dos campos do formulário deverão estar preparados para receber tais dados.

No `form` temos 3 campos e 5 botões, observe que cada campo tem um `name`, que o qual devemos nos referir para passagem e recepção de dados. Além disso, eles têm um `value` ao qual podemos atribuir valores. Observe que para o `value` de cada campo colocamos uma expressão `jsp`, pois os valores serão dinâmicos, dependendo dos valores retornados do banco de dados, ou seja, quando carregamos a página pela primeira vez, eles estarão vazios porque os parâmetros serão nulos e receberão vazio ou zero. Mas, quando fizermos uma pesquisa e selecionarmos uma linha, estes valores deverão preencher esses campos, como será visto mais adiante.

Em relação aos botões, quatro deles tem o mesmo nome, pois é necessário para garantirmos que apenas um componente submeterá o formulário. O que diferencia um do outro é o seu valor, por exemplo, quando o usuário clica em "Excluir", saberemos a sua intenção devido ao valor do botão, ou seja, um botão com mesmo nome pode ter valores diferentes.

O botão "limpar" chama a função `javascript` para limpar os campos, note que o tipo dele não é `submit`, pois ele não irão submeter o formulário, somente chamará função que o limpa.

13.2. Criando a página para as operações CRUD para proprietário

Nesta etapa faremos o código para executar as operações no banco de dados, basicamente criaremos objetos e enviaremos mensagens às nossas classes, conforme pode ser visto no código abaixo. Crie uma nova página chamada `proprietarioOp.jsp` e acrescente o código abaixo.

```
<%@page import="br.com.tutorial.* , java.sql.* , java.util.*"
%
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Operações no banco de dados</title>
    </head>
    <body>
        <%
            //Criação das variáveis para receber os dados
            //para preencher o formulário após a pesquisa
            //Para cada parâmetro passado é feito um teste
            //para verificar se esse veio nulo, se veio nulo,
            //é atribuído valor em branco.
            String codigo = ((request.getParameter("codigo") != null) ?
                request.getParameter("codigo")) : "0");
        %>
```

```

        String nome = ((request.getParameter("nome") != null) ? (request.getParameter("nome")) : "");
        String cidade = ((request.getParameter("cidade") != null) ? (request.getParameter("cidade")) : "");
        String crud = ((request.getParameter("crud") != null) ? (request.getParameter("crud")) : "");

        //Criando os objetos para comunicação com o banco de dados
        //O netbeans não reconhecerá as nossas classes, mas isso não é problema, pois o tomcat reconhece
        Connection con = Conexao.abrirConexao();
        ProprietarioBean pb = new ProprietarioBean();
        ProprietarioDAO pd = new ProprietarioDAO(con);

        //Atribuindo os valores ao objeto criado
        if(!codigo.equals("")){
            pb.setCodigo(Integer.parseInt(codigo));
        }else{
            pb.setCodigo(0);
        }

        pb.setNome(nome);
        pb.setCidade(cidade);

        //Verifica se o usuário clicou no botão Incluir
        if (crud.equals("Incluir")) {
            //Invoca o método inserir e imprime o seu retorno
            out.print(pd.inserir(pb));
        }

        //Verifica se o usuário clicou no botão Alterar
        if (crud.equals("Alterar")) {
            //Invoca o método alterar e imprime o seu retorno
            out.print(pd.alterar(pb));
        }

        //Verifica se o usuário clicou no botão Excluir
        if (crud.equals("Excluir")) {
            //Invoca o método excluir e imprime o seu retorno
            out.print(pd.excluir(pb));
        }

        //Verifica se o usuário clicou no botão Pesquisar
        if (crud.equals("Pesquisar")) { %>
<table border="0">

```

```

<tr bgcolor="#f2f2f2">
    <td>
        Código
    </td>
    <td>
        Nome
    </td>
    <td>
        Cidade
    </td>
</tr>
<%
    List<ProprietarioBean> lp =
pd.listarTodos();
%>

<%
    int x = 1;
    String cor = "#FFFFFF";
    for (ProprietarioBean p : lp) {
        if (x % 2 == 0) {
            cor = "#999999";
        } else {
            cor = "#FFFFFF";
        }
%>
<tr bgcolor="<%="cor%">">
    <td><a href="proprietario.jsp?codigo=<%=p.getCodigo()%>&nome=<%=p.
getNome()%>&cidade=<%=p.getCidade()%>"><%=p.getCodigo()%></
a></td>
    <td><%=p.getNome()%></td>
    <td><%=p.getCidade()%></td>
</tr>
<%
    x++;
%>

<%
    Conexao.fecharConexao(con);
%>

</body>
</html>

```

Vamos fazer uma análise dos pontos mais importantes desse código. Na linha abaixo estamos importando todos os pacotes necessário para executarmos as operações. Note que precisamos importar 3 pacotes, o que contém as nossas classes, o que contém as classes para comandos SQL e o que contém as

classes do framework collections. Observe que cada importação é separada por vírgula.

```
<%@page import="br.com.tutorial.* , java.sql.* , java.util.*" %>
```

O trecho de código abaixo declara as variáveis que receberão os parâmetros passados pelo formulário. Perceba que mantivemos os mesmos nomes dos objetos do formulário, que são os mesmos nomes dos atributos da classe e da tabela. Embora isso não seja necessário, mas é uma boa prática fazer isso, pois evita que nos percamos durante a fase desenvolvimento.

```
String codigo = ((request.getParameter("codigo") != null) ?  
    (request.getParameter("codigo")) : "0");  
String nome = ((request.getParameter("nome") != null) ?  
    (request.getParameter("nome")) : "");  
String cidade = ((request.getParameter("cidade") != null) ?  
    (request.getParameter("cidade")) : "");  
String crud = ((request.getParameter("crud") != null) ?  
    (request.getParameter("crud")) : "");
```

O trecho abaixo é a criação dos objetos que precisamos para as operações CRUD.

```
Connection con = Conexao.abrirConexao();  
ProprietarioBean pb = new ProprietarioBean();  
ProprietarioDAO pd = new ProprietarioDAO(con);
```

O trecho abaixo atribui os valores das variáveis para os atributos da classe. Note que o código foi preciso ser convertido em inteiro para que possa ficar compatível com o tipo de dado do atributo correspondente. Observe, ainda, que foi necessário fazer um teste antes para não ocorrer uma exceção caso tentemos converter uma String vazia.

```
if (!codigo.equals("")) {  
    pb.setCodigo(Integer.parseInt(codigo));  
} else {  
    pb.setCodigo(0);  
}  
  
pb.setNome(nome);  
pb.setCidade(cidade);
```

O trecho abaixo verifica se o usuário clicou em incluir, alterar ou excluir. Precisamos de apenas uma linha para executar os comandos e exibir a resposta vinda do servidor, dado que o retorno dos nossos métodos é uma String.

```
//Verifica se o usuário clicou no botão Incluir  
if (crud.equals("Incluir")) {  
    //Invoca o método inserir e imprime o seu retorno  
    out.print(pd.inserir(pb));  
}  
  
//Verifica se o usuário clicou no botão Alterar  
if (crud.equals("Alterar")) {
```

```

        //Invoca o método alterar e imprime o seu retorno
        out.print(pd.alterar(pb));
    }

    //Verifica se o usuário clicou no botão Excluir
    if (crud.equals("Excluir")) {
        //Invoca o método excluir e imprime o seu retorno
        out.print(pd.excluir(pb));
    }
}

```

Quando a intenção do usuário é pesquisar, o código é um pouco mais complexo, por isso será analisado por parte, como mostrado a seguir.

`if (crud.equals("Pesquisar")) { %>` aqui interrompemos a tag jsp para incluirmos código html, podemos misturar html e jsp à vontade, desde que atentemos para a abertura e fechamento das tags jsp.

No trecho abaixo criamos uma tabela com borda zero e criamos o cabeçalho dela (um linha com 3 colunas) com os valores que não serão repetidos. As demais linhas serão inseridas dinamicamente dependendo da quantidade de linhas que virão do banco.

```

<table border="0">
<tr bgcolor="#f2f2f2">
    <td>
        Código
    </td>
    <td>
        Nome
    </td>
    <td>
        Cidade
    </td>
</tr>

```

Na linha abaixo criamos lista para receber o retorno do método `listarTodos` que estamos invocando da nossa classe.

```
List<ProprietarioBean> lp = pd.listarTodos();
```

No trecho abaixo estamos, de fato, listando os valores na nossa tabela. Note que criamos For-Each para percorrermos nossa lista e estamos alternando as cores das linhas entre cinza e branco. Além disso, colocamos link para o campo código e estamos passando os valores para o formulário, de acordo com a linha selecionada pelo usuário. Observe que dentro das aspas de `href` colocamos todos os campos a serem retornados ao formulário e fora colocamos os valores a serem exibidos, por isso há repetição dos valores.

```

int x = 1;
String cor = "#FFFFFF";
for (ProprietarioBean p : lp) {
    if (x % 2 == 0) {
        cor = "#999999";
    } else {

```

```

        cor = "#FFFFFF";
    }
%>

<tr bgcolor="<%=>cor%>">
    <td><a href="proprietario.jsp?codigo=<%=p.getCodigo()%>&nome=<%=p.getNome()%>&cidade=<%=p.getCidade()%>"><%=p.getCodigo()%></a></td>
        <td><%=p.getNome()%></td>
        <td><%=p.getCidade()%></td>
    </tr>

<% x++ ;
} %>

<% } %>

```

Bom, agora vamos ver esse código em ação, faremos uma inclusão, uma alteração e uma exclusão e, é claro que teremos que fazer pesquisa para executar as operações de alteração e exclusão.

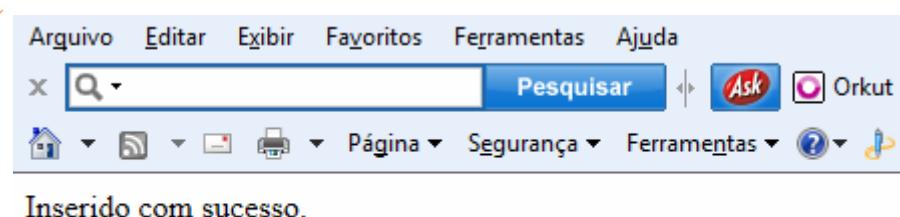
Abra o browser e digite o comando <http://localhost:8080/tutorialweb/web/proprietario.jsp>, preencha os dados como mostra a figura abaixo e click em "Incluir". Não é necessário preencher o campo código, pois este é automaticamente gerado.

Código

Nome

Cidade

Se você seguiu todos os passos corretamente, deverá ver a mensagem da figura abaixo.



Volte para o formulário, click em "Limpar" e Depois em "Pesquisar". Os dados constantes na tabela de proprietário serão listados conforme figura abaixo.

Código	Nome	Cidade
1	João	Brasília
2	Maria	Rio de Janeiro
3	Maria da Penha	Rio de Janeiro

Selecione o registro que você acabou de inserir, para isso click no código, isso levará você de volta ao formulário com os campos preenchidos, como mostra a figura abaixo.

Código	Nome	Cidade
3	Maria da Penha	Rio de Janeiro

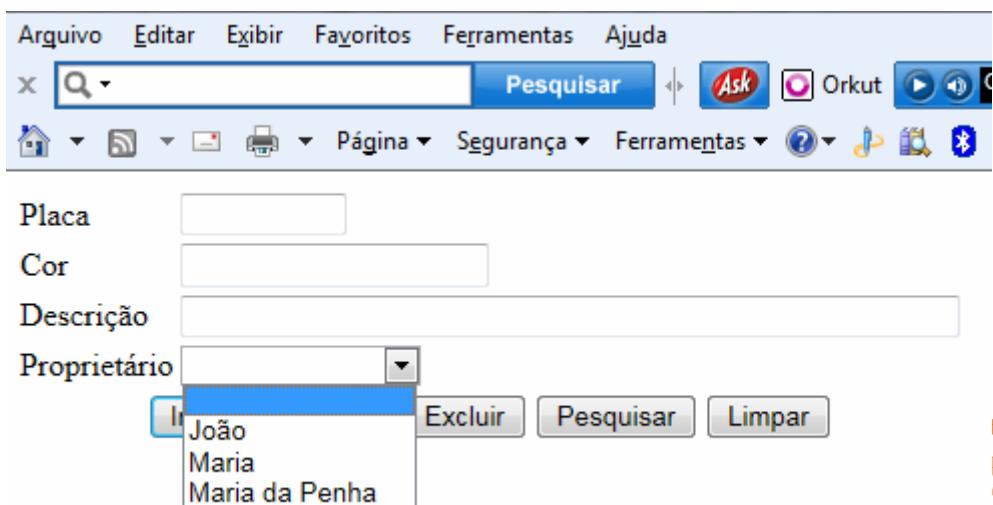
Incluir Alterar Excluir Pesquisar Limpar

Troque "Rio de Janeiro" por "Rio de Fevereiro" e click em alterar. Verifique o resultado, fazendo pesquisa. Selecione algum registro e click em excluir. Se tudo funcionou corretamente está na hora de fazermos as nossas telas para o CRUD da tabela de carros.

13.3. Criando o formulário para carro

O procedimento para criação deste formulário semelhante à criação do anterior, com pequenas diferenças que são explicadas na seqüência.

Primeiro devemos criar o formulário da figura abaixo, cujo código é colocado em seguida.



```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="br.com.tutorial.* , java.sql.* , java.util.*"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Tela para manutenção dos dados de
carros</title>
    </head>
    <body>
        <script language="javascript">
            //Função para limpar os campos do formulário
            function limpar(){
                with(document.form1){
                    codigo.value = '';
                    cor.value = '';
                    descricao.value = '';
                    placa.value = '';
                }
            }
        </script>
        <%
            //Criação das variáveis para receber os dados
            para preencher o formulário após a pesquisa
            //Para cada parâmetro passado é feito um teste
            para verificar se esse veio nulo, se veio nulo,
            //é atribuído valor em branco.
            String codigo = ((request.getParameter("codigo"))
                != null) ?
                (request.getParameter("codigo")) : "0";
            String placa = ((request.getParameter("placa"))
                != null) ? (request.getParameter("placa")) : "";
        %>
    </body>
</html>
```

```

        String cor = ((request.getParameter("cor") != null) ? (request.getParameter("cor")) : "");
        String descricao = ((request.getParameter("descricao") != null) ? (request.getParameter("descricao")) : "");

        Connection con = Conexao.abrirConexao();
        ProprietarioDAO pd = new ProprietarioDAO(con);

        //Nome do proprietário do carro
        String nome = "";

        if (!codigo.equals("") && !codigo.equals("0"))
{
            List<ProprietarioBean> listap = pd.listarTodos();

            for (ProprietarioBean p : listap) {
                if (p.getCodigo() == Integer.parseInt(codigo)) {
                    nome = p.getNome();
                }
            }
}
%>

        <form name="form1" action="carroOp.jsp"
method="get">
        <table border="0">
        <tr>
            <td>Placa</td>
            <td><input name="placa" size="8" maxlength="8" value="<%=placa%>" /></td>
        </tr>
        <tr>
            <td>Cor</td>
            <td><input name="cor" size="20" maxlength="20" value="<%=cor%>" /></td>
        </tr>
        <tr>
            <td>Descrição</td>
            <td><input name="descricao" size="60" maxlength="60" value="<%=descricao%>" /></td>
        </tr>
        <tr>
            <td>Proprietário</td>
            <td>
                <%>
                    List<ProprietarioBean> listap =
pd.listarTodos();
                <%>
            </td>
        </tr>
    </table>
</form>

```

```

        <select name="codigo">
            <option
value="<%="codigo%>"><%="nome%></option>
            <%
                for (ProprietarioBean pb      :
listap) {
                    %
                    <option
value="<%="pb.getCodigo () %>"><%="pb.getNome () %></option>
                    <%}>
            </select>
        </td>
    </tr>
    <tr>
        <td colspan="2" align="center">
            <input type="submit" name="crud"
value="Incluir"/>
            <input type="submit" name="crud"
value="Alterar"/>
            <input type="submit" name="crud"
value="Excluir"/>
            <input type="submit" name="crud"
value="Pesquisar"/>
            <input type="button" value="Limpar"
onclick="limpar ()"/>
        </td>
    </tr>
</table>
</form>
<% Conexao.fecharConexao (con);%>
</body>
</html>

```

O código acima tem duas pequenas diferenças em relação ao código do formulário de proprietários. A primeira delas está relacionado ao código abaixo, onde é preciso buscar o nome do proprietário para o código passado quando o usuário seleciona uma linha da lista de carros após a pesquisa. Isso deve ser feito para preenchermos o combobox com o valor correto.

```

Connection con = Conexao.abrirConexao();
ProprietarioDAO pd = new ProprietarioDAO(con);

//Nome do proprietário do carro
String nome = "";

if (!codigo.equals("") && !codigo.equals("0")) {
    List<ProprietarioBean> listp = pd.listarTodos();

    for (ProprietarioBean p : listp) {
        if (p.getCodigo() == Integer.parseInt(codigo)) {
            nome = p.getNome();
        }
    }
}

```

```
}
```

A segunda diz respeito ao combobox que deve ser preenchido dinamicamente, conforme pode ser visto no código abaixo. Note que a primeira linha do combobox é preenchida com o valor resultante da pesquisa realizado pelo código acima, as outras são preenchidas com os valores vindos da nova pesquisa, feita no código abaixo.

```
<tr>
    <td>Proprietário</td>
    <td>
        <%
            List<ProprietarioBean> listap = pd.listarTodos();
        %>
        <select name="codigo">
            <option value="<%.codigo%>"><%=nome%></option>
            <%
                for(ProprietarioBean pb : listap){
            %>
            <option value="<%pb.getCodigo()%>"><%=pb.getNome()%></option>
            <%}>
        </select>
    </td>
</tr>
```

13.4. Criando a página para as operações CRUD para carro

A criação desta página é idêntica à criação da página para CRUD de proprietário, como pode ser visto no código a seguir.

```
<%@page import="br.com.tutorial.* , java.sql.* , java.util.*"
%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Operações no banco de dados</title>
    </head>
    <body>
        <%
            //Criação das variáveis para receber os dados
            //para preencher o formulário após a pesquisa
            //Para cada parâmetro passado é feito um teste
            //para verificar se esse veio nulo, se veio nulo,
            //é atribuído valor em branco.
            String codigo = ((request.getParameter("codigo") != null) ?
            (request.getParameter("codigo")) : "0");
            String placa = ((request.getParameter("placa") != null) ? (request.getParameter("placa")) : "");
```

```

        String cor = ((request.getParameter("cor") != null) ? (request.getParameter("cor")) : "");
        String descricao = ((request.getParameter("descricao") != null) ? (request.getParameter("descricao")) : "");
        String crud = ((request.getParameter("crud") != null) ? (request.getParameter("crud")) : "");

        //Criando os objetos para comunicação com o banco de dados
        //O netbeans não reconhecerá as nossas classes, mas isso não é problema, pois o tomcat reconhece
        Connection con = Conexao.abrirConexao();
        CarroBean cb = new CarroBean();
        CarroDAO cd = new CarroDAO(con);

        //Atribuindo os valores ao objeto criado
        if(!codigo.equals("")){
            cb.setCodigo(Integer.parseInt(codigo));
        }else{
            cb.setCodigo(0);
        }

        cb.setPlaca(placa);
        cb.setCor(cor);
        cb.setDescricao(descricao);

        //Verifica se o usuário clicou no botão Incluir
        if (crud.equals("Incluir")) {
            //Invoca o método inserir e imprime o seu retorno
            out.print(cd.inserir(cb));
        }

        //Verifica se o usuário clicou no botão Alterar
        if (crud.equals("Alterar")) {
            //Invoca o método alterar e imprime o seu retorno
            out.print(cd.alterar(cb));
        }

        //Verifica se o usuário clicou no botão Excluir
        if (crud.equals("Excluir")) {
            //Invoca o método excluir e imprime o seu retorno
            out.print(cd.excluir(cb));
        }

        //Verifica se o usuário clicou no botão Pesquisar
        if (crud.equals("Pesquisar")) { %>

```

```

<table border="0">
    <tr bgcolor="#f2f2f2">
        <td>
            Placa
        </td>
        <td>
            Cor
        </td>
        <td>
            Descrição
        </td>
    </tr>
    <%
        List<CarroBean> lc = cd.listarTodos();
    %>

    <%
        int x = 1;
        String corLinha = "#FFFFFF";
        for (CarroBean c : lc) {
            if (x % 2 == 0) {
                corLinha = "#999999";
            } else {
                corLinha = "#FFFFFF";
            }
        }
    %>

    <tr bgcolor="<%="corLinha%>">
        <td><a href="carro.jsp?codigo=<%">c.getCodigo()%>&cor=<%">c.getCor()%>&descricao=<%">c.getDescricao()%>&placa=<%">c.getPlaca()%>"><%">c.getPlaca()%></a></td>
        <td><%">c.getCor()%></td>
        <td><%">c.getDescricao()%></td>
    </tr>
    <% x++;
    }%>

    <%}
    Conexao.fecharConexao(con);
%>

</body>
</html>

```

Este tutorial serve apenas de guia para desenvolvimento de uma simples aplicação. Se o leitor deseja entender todos os conceitos tratados aqui, sugiro que faça cursos de banco de dados, Java e JSP.

Com isso, terminamos a nossa série de tutoriais para desenvolvimento de uma simples aplicação. Para ver outros tutoriais ou apostilas, acesse o nosso site.

http://www.pusivus.com.br