

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 I

1. Introdução

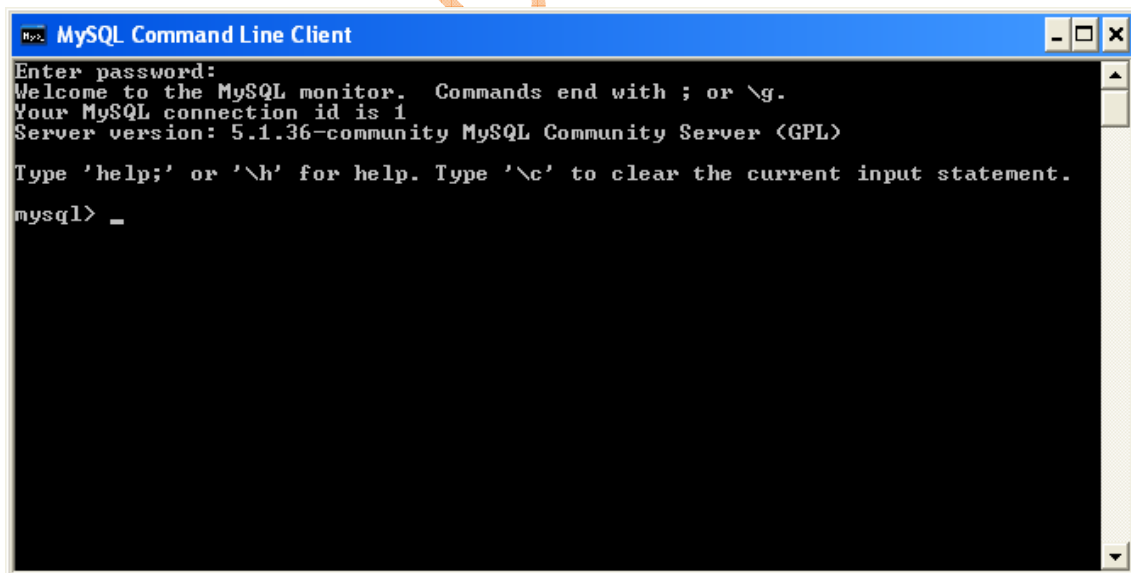
Este tutorial não tem como objetivo treinar um programador Java e sim mostrar como criar uma pequena aplicação usando alguns recursos do Java tendo como SGBD o MySQL. Para usar este tutorial será necessário ter os seguintes softwares instalados: netbeans 6.8 ou superior (http://netbeans.org/index_pt_BR.html), MySQL Server 5.0 ou superior (<http://dev.mysql.com/downloads/>) e o driver (conector) do MySQL (<http://dev.mysql.com/downloads/connector/j/>).

2. Criação do banco de dados

Por questões didáticas, o banco de dados criado para este tutorial se limita a uma tabela, com apenas 3 campos. O objetivo desse banco é manter dados dos carros que passam por determinado estacionamento. Bom, vamos à nossa primeira tarefa, que é a criação do banco.

Primeiro passo: acesse o MySQL Command Line Client (iniciar->programas->MySQL->MySQL Server 5.x->MySQL Command Line Client).

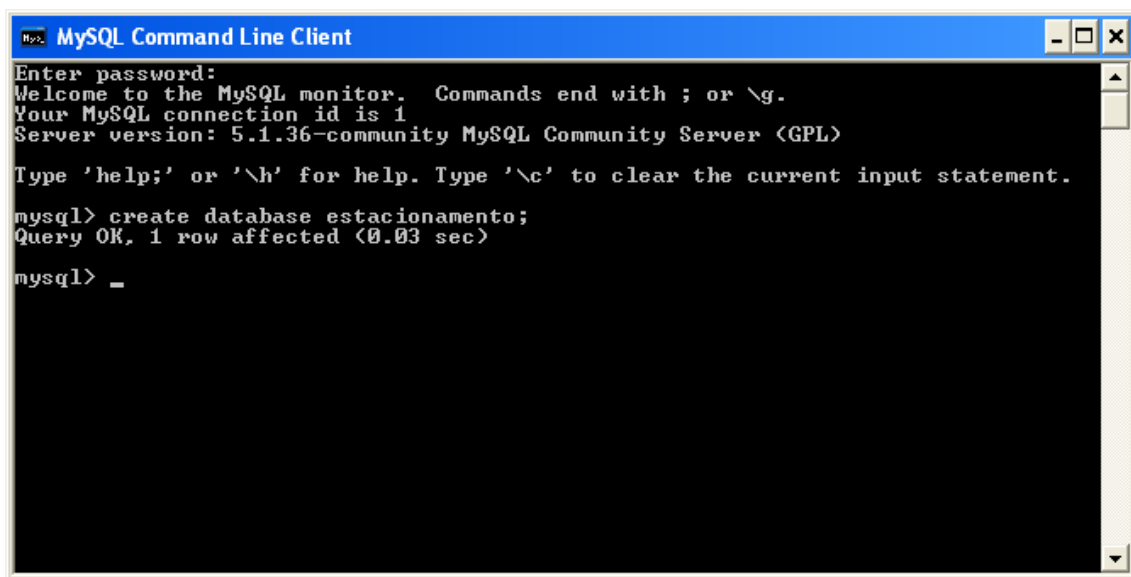
Segundo passo: entre com a senha (aquela que você colocou quando fez a instalação). Se você não alterou as configurações, a senha padrão é em branco para o usuário root. Se tudo estiver ok, você verá a janela com aparência da figura abaixo.



Terceiro passo: criar o banco (vamos chamar de estacionamento) propriamente dito, para isso digite o comando abaixo e tecla enter.

```
create database estacionamento;
```

Se o banco foi criado com sucesso a sua janela deverá se parecer com a figura abaixo.



```
MySQL Command Line Client
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.36-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

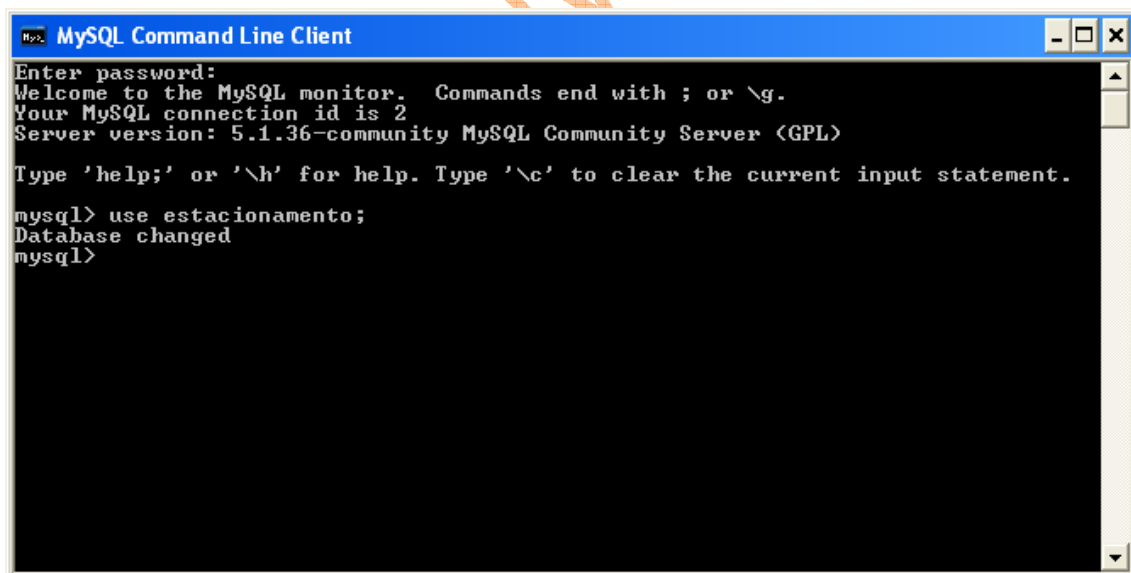
mysql> create database estacionamento;
Query OK, 1 row affected (0.03 sec)

mysql> _
```

Quarto passo: dizer para o MySQL que você deseja usar esse banco que acaba de criar. Para isso digite o comando abaixo e tecla enter.

```
use estacionamento;
```

Se o banco foi mudado com sucesso a sua janela deverá se parecer com a figura abaixo.



```
MySQL Command Line Client
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.1.36-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use estacionamento;
Database changed
mysql>
```

Quinto passo: criação da tabela. Para isso digite os comandos a seguir, linha a linha e tecla enter a cada linha. Só coloque o ponto-e-vírgula na última linha, pois este (ponto-e-vírgula) diz para o MySQL que o comando terminou.

```
create table carro(
placa char(7) not null primary key,
```

```
cor varchar(20),  
descricao varchar(100)  
);
```

Obs.: usei `char` para placa porque tenho certeza de que todo o espaço alocado (7) será usado. Já para cor e descricao, usei `varchar` porque não sei exatamente quanto desse espaço será usado.

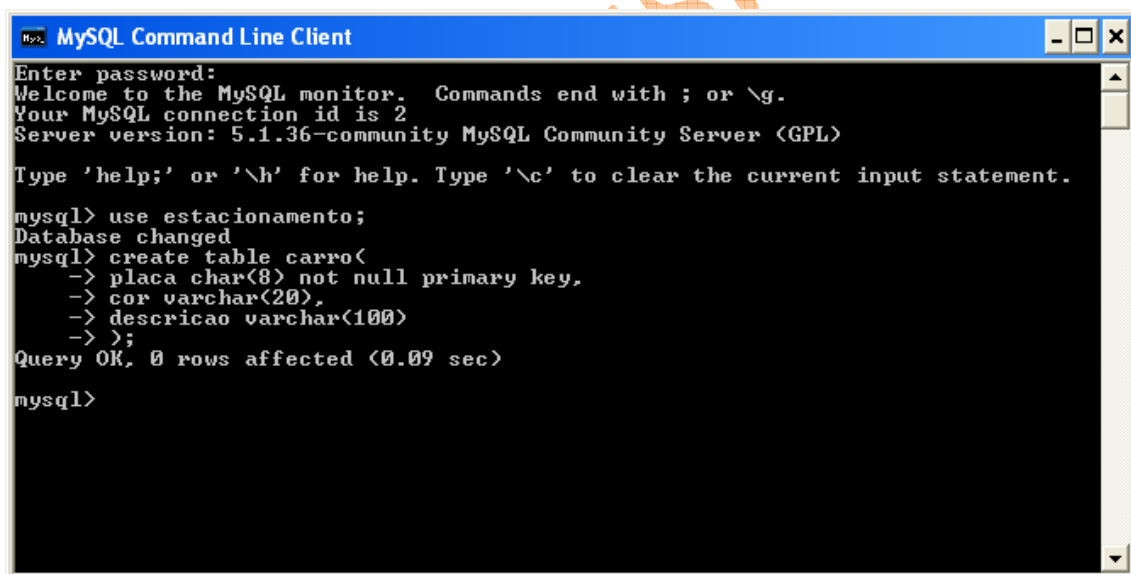
Pense assim, quando coloco o tipo `char`, eu digo para o SGBD que eu quero aquele espaço para minha variável e ele (SGBD) tem que deixar exatamente esse tamanho para o meu uso, mesmo que eu não use todo ele.

No caso de `varchar`, eu digo que preciso de no máximo aquele tamanho, assim o MySQL vai alocando os espaços à medida que eu precisar.

Em suma, o `char` é mais rápido, mas ocupa espaço desnecessário, já o `varchar` é mais lento, porém não ocupa espaço desnecessário.

O uso de um ou de outro depende do problema de cada desenvolvedor.

Se tudo deu certo, a sua janela deverá se parecer com a figura abaixo. Não se preocupe com a mensagem "0 rows affected", isso se dá porque a tabela está vazia.



```
MySQL Command Line Client  
Enter password:  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 2  
Server version: 5.1.36-community MySQL Community Server (GPL)  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
mysql> use estacionamento;  
Database changed  
mysql> create table carro(  
-> placa char(8) not null primary key,  
-> cor varchar(20),  
-> descricao varchar(100)  
-> );  
Query OK, 0 rows affected (0.09 sec)  
  
mysql>
```

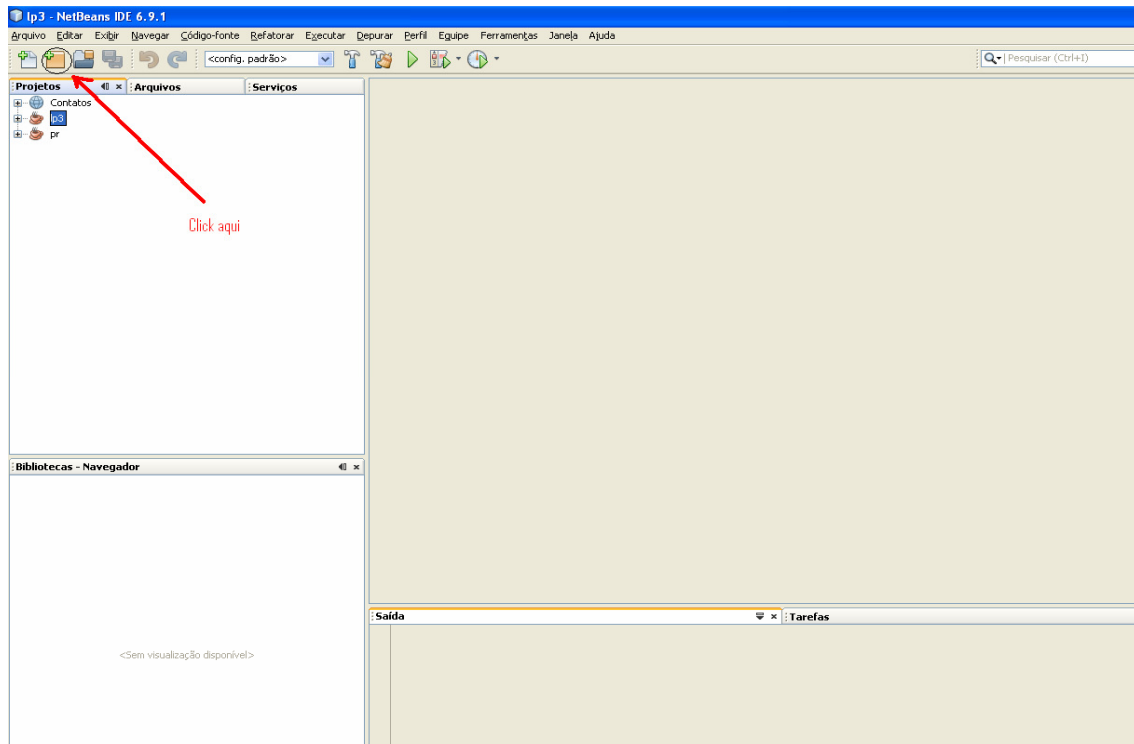
Pronto! Seu banco de dados está criado e você tem uma tabela nele e é com essa tabela que vamos trabalhar. Vamos para outra etapa.

3. Criação do projeto

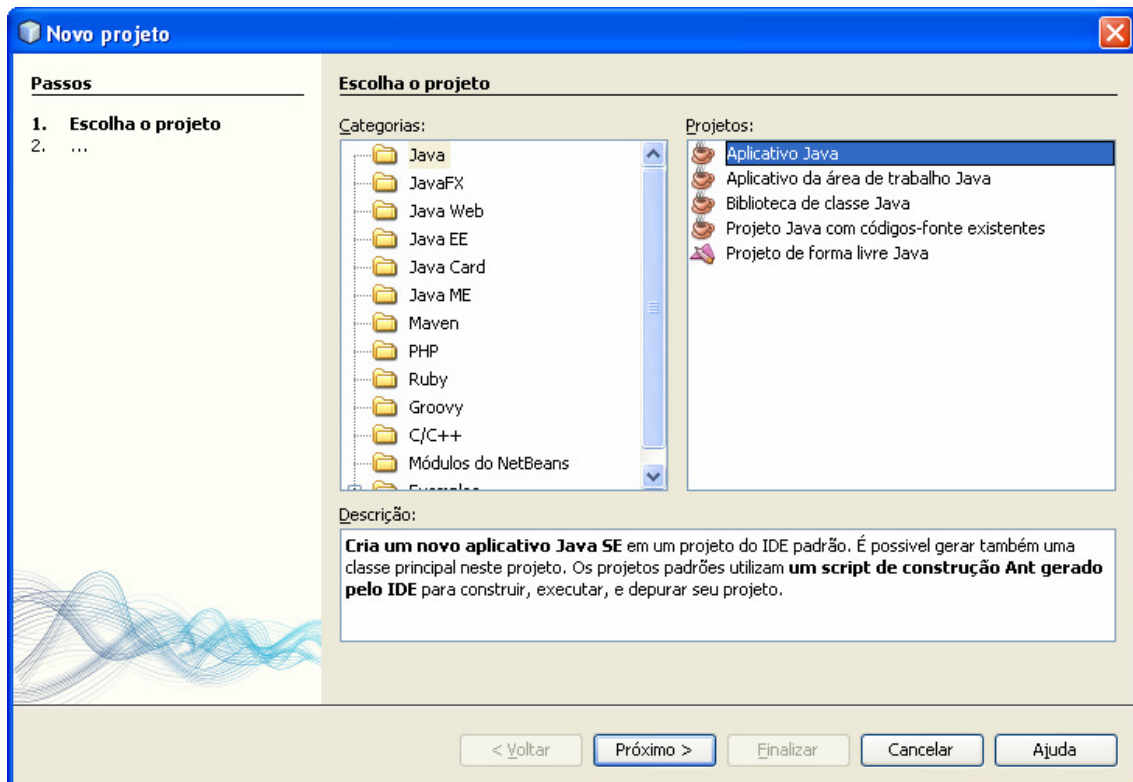
Para esta etapa é necessário que você já tenha o netbeans instalado na sua máquina.

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

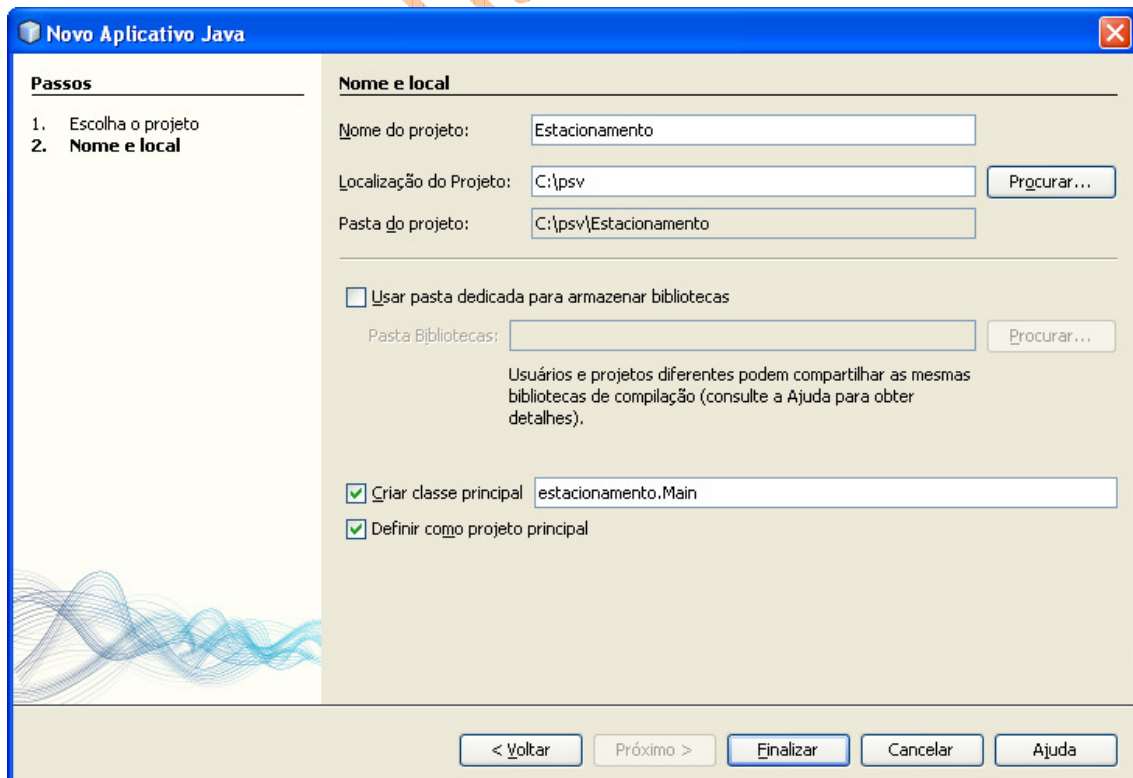
Segundo passo: criar um novo projeto. Para isso, click no ícone mostrado na figura abaixo.



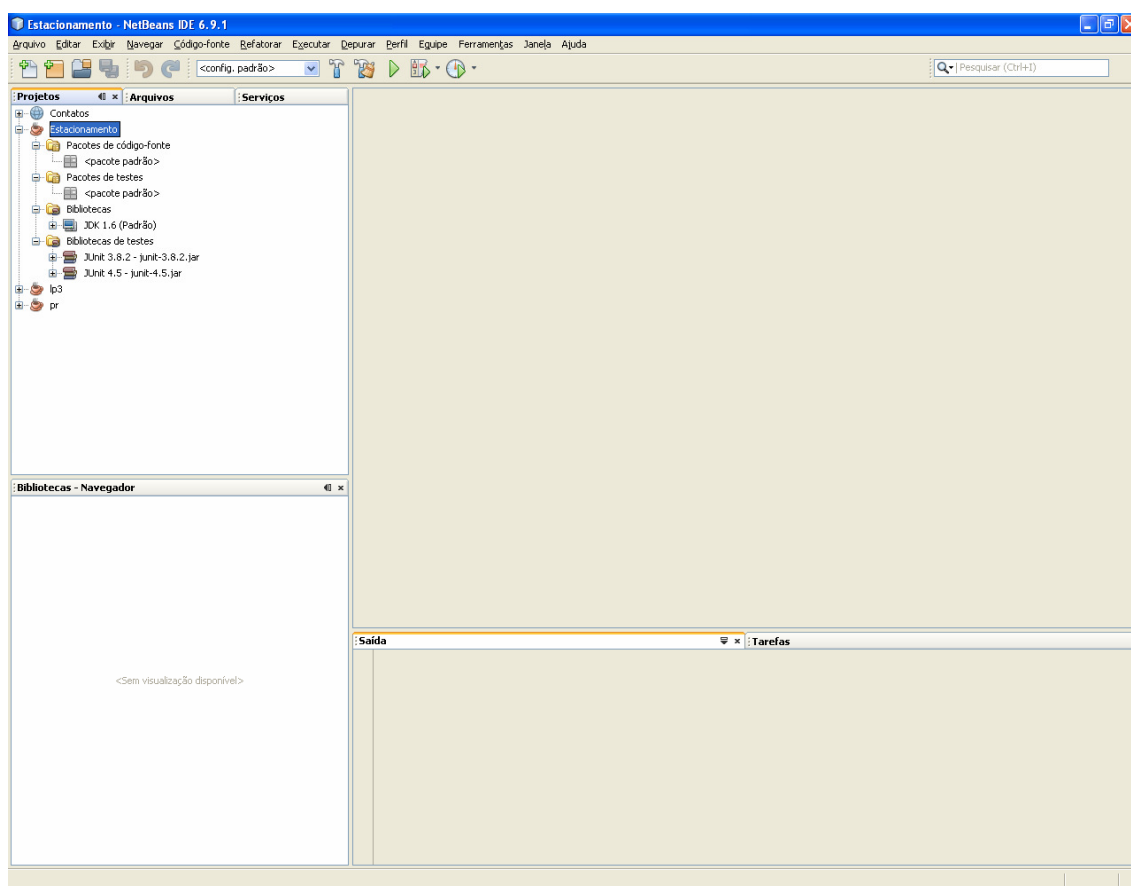
Ao aparecer uma nova janela, selecione Java em categorias e aplicativo Java em projetos, conforme figura abaixo, e click em próximo.



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

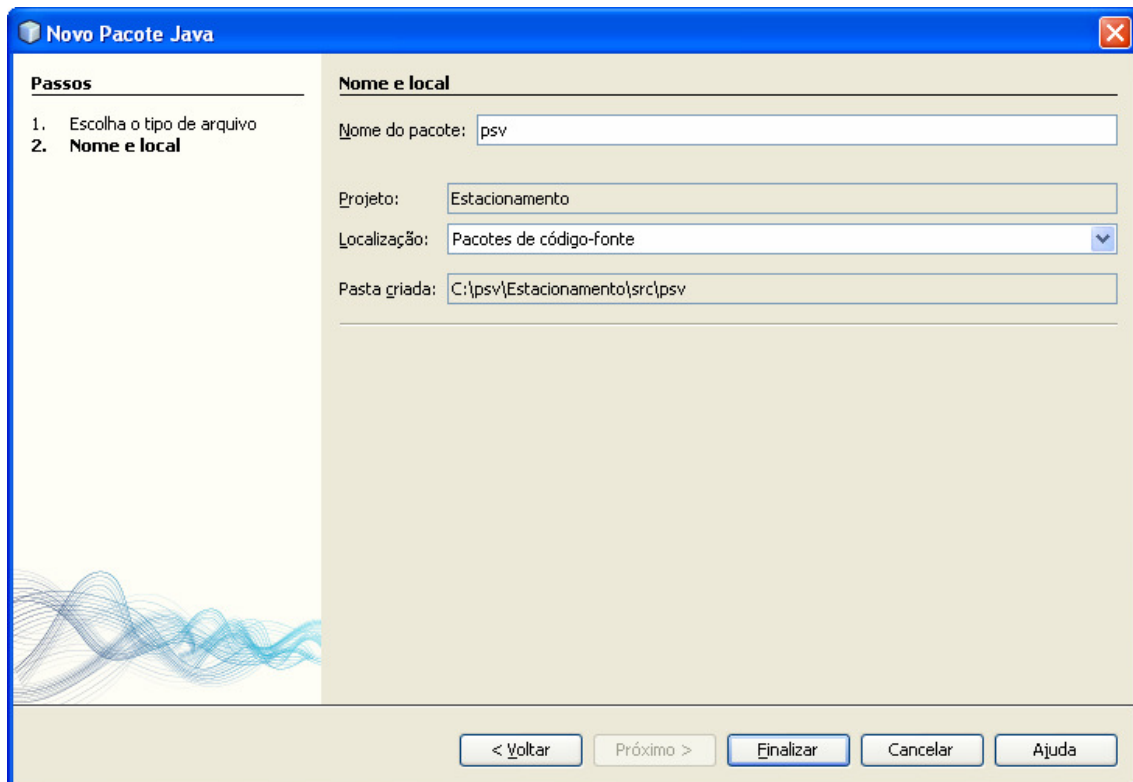


O novo projeto já vem com algumas pastas: pacotes de código fonte (onde guardaremos as nossas classes), pacote de testes, bibliotecas (onde colocaremos nossas bibliotecas externas) e bibliotecas de testes. Só vamos nos preocupar com a primeira e terceira pasta. Normalmente o netbeans já cria um pacote com o nome da pasta onde foi criado o projeto, se isso não acontecer, crie um pacote seguindo orientações da seção 4. 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.



4. Criação do pacote

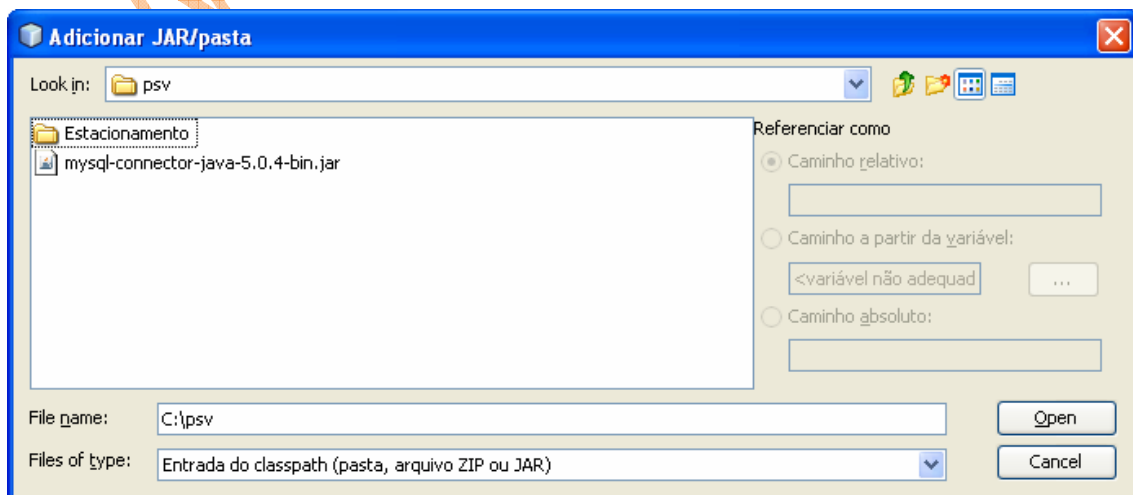
É sempre recomendável que criemos um pacote para colocarmos nossas classes. Para isso, click com o botão direito do mouse sobre a pasta "Pacotes de código fonte", escolha novo->pacote. Uma nova janela aparecerá preencha os dados conforme figura abaixo. Coloque o nome `psv` para o nome do pacote.



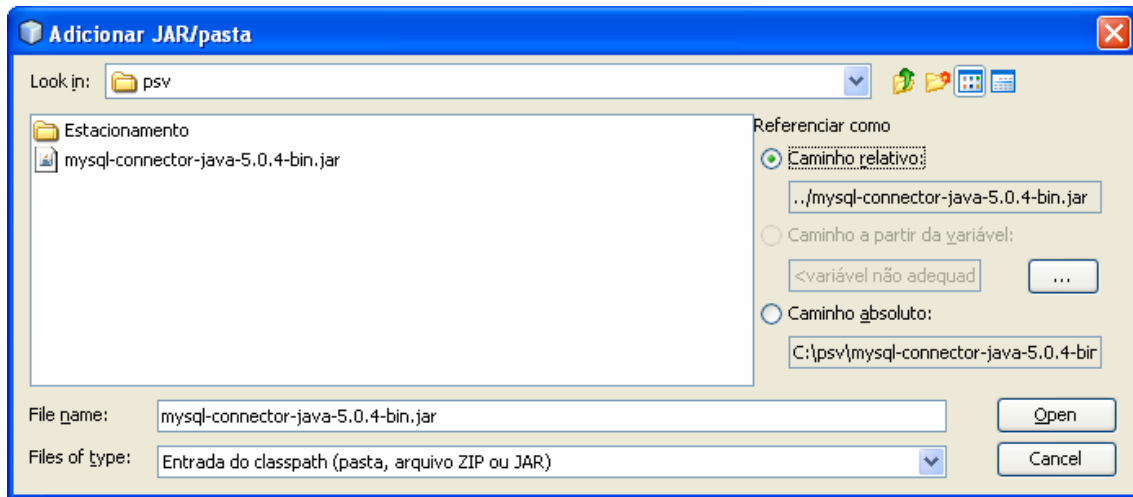
5. Adição do conector

Para esta etapa você precisará ter o conector em alguma pasta na sua máquina, se ainda não tem, faça download no seguinte endereço: <http://dev.mysql.com/downloads/connector/j/>, descompacte-o (normalmente vem zipado) em uma pasta de fácil acesso, de preferência, como por exemplo: C:\psv\mysql-connector-java-5.0.4-bin.jar

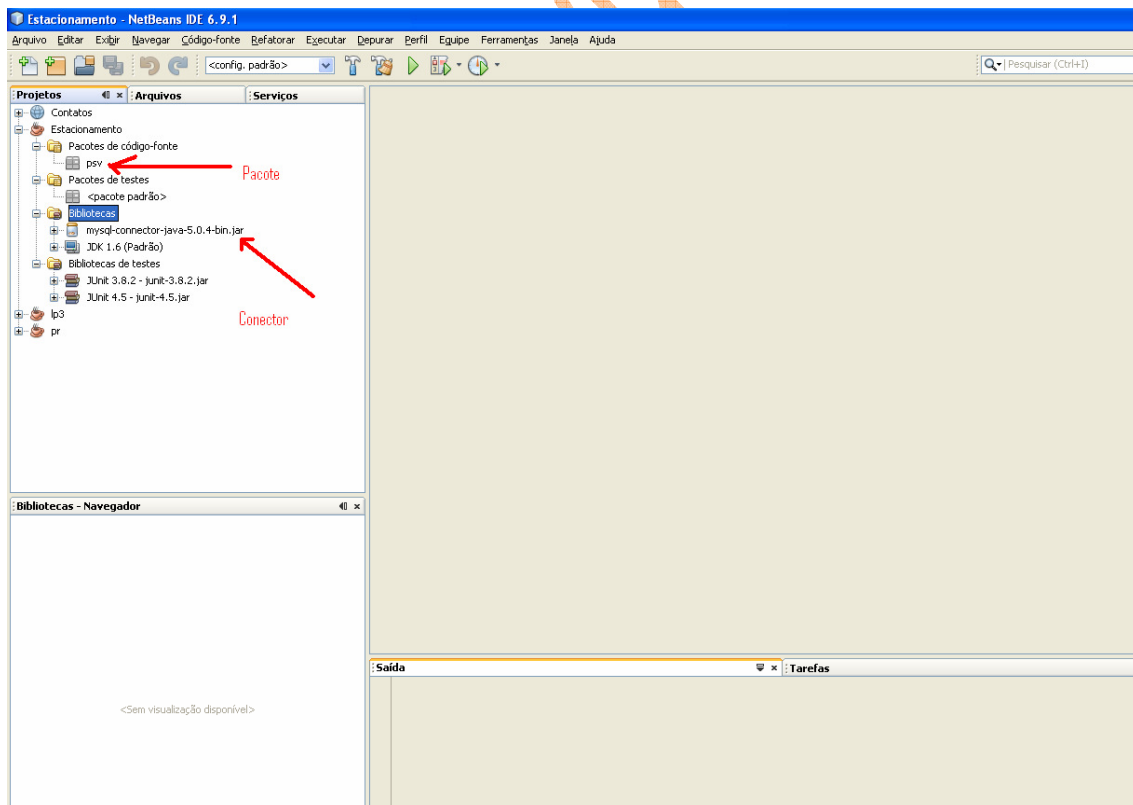
Primeiro passo: click com o botão direito do mouse sobre a pasta “Bibliotecas” e selecione a opção “Adicionar JAR/pasta”. Quando aparecer uma nova janela, navegue por ela até localizar o conector, conforme mostra a figura abaixo.



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.

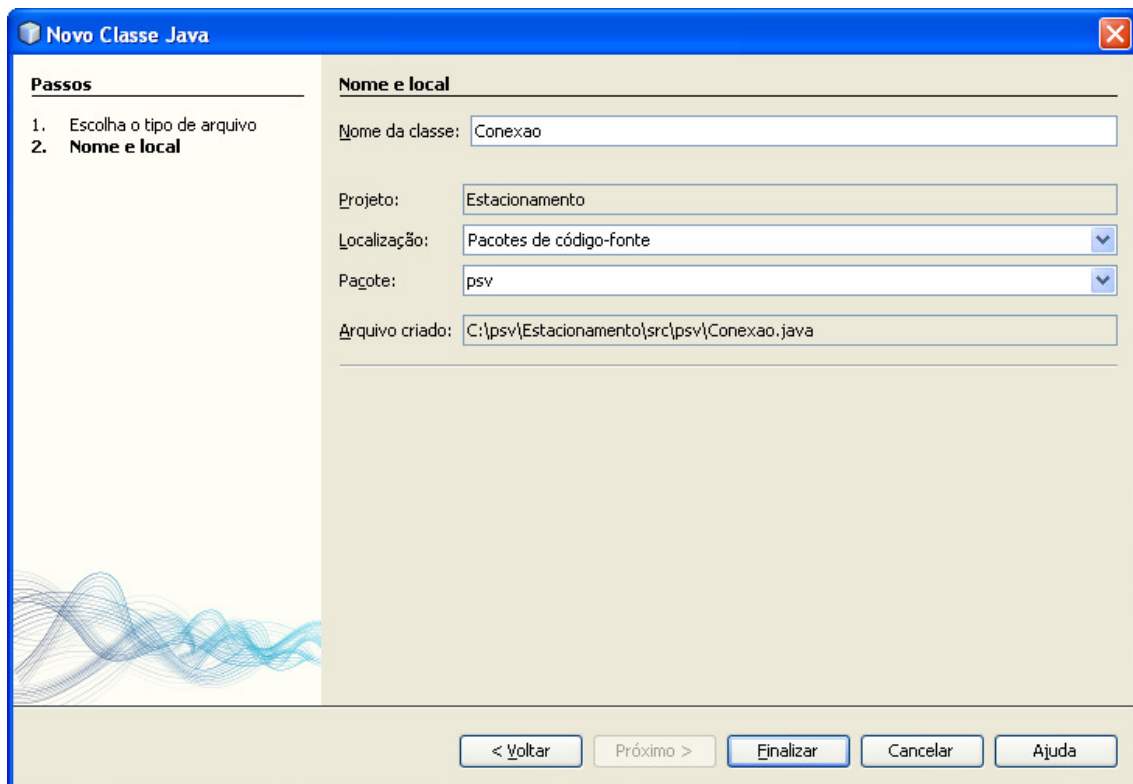


Pronto! Você agora tem um pacote para trabalhar e já tem o conector para o banco de dados, conforme mostra a figura abaixo.



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 4 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), 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 psv; //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) `getConnection` 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/estacionamento?";
```

```
url += "user=root&password=";
```

Antes de passar a `url` (poderia se qualquer nome) como parâmetro do método `getConnection`, 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.

`estacionamento` – 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`.

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 `catchs`, 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 `catchs`, 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());  
  
}  
}
```

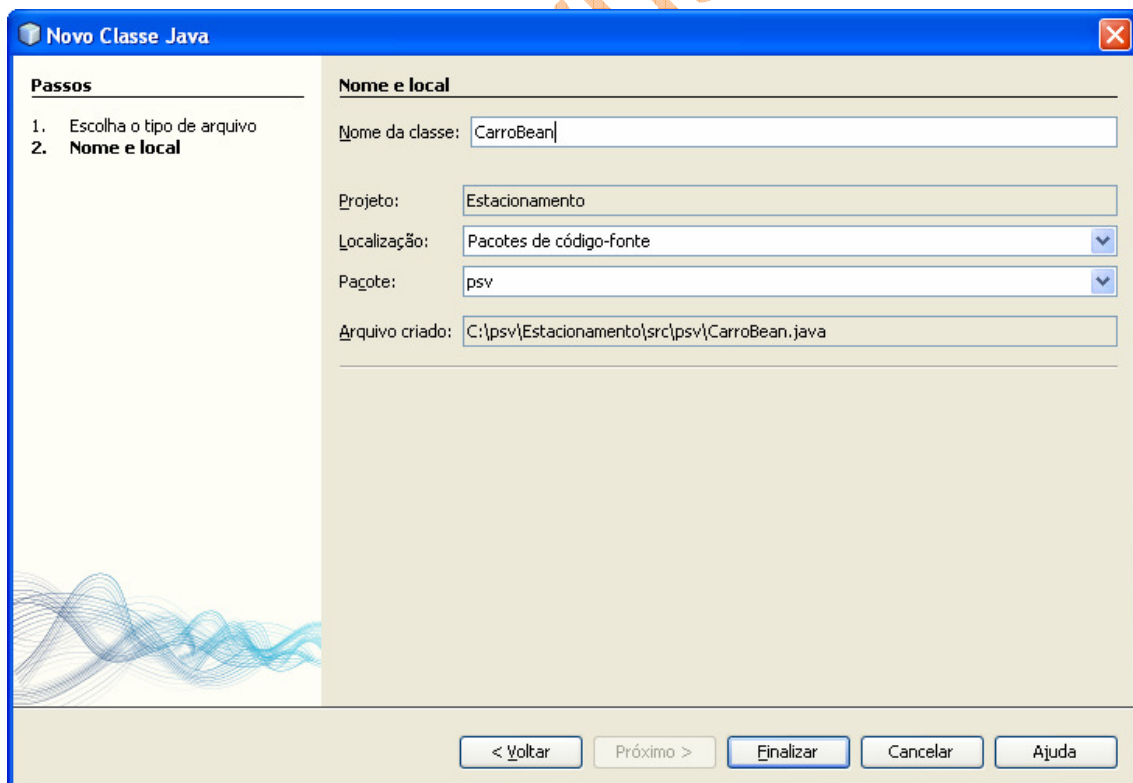
Podemos fechar o método, então.

```
}
```

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 psv; //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;
```

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.

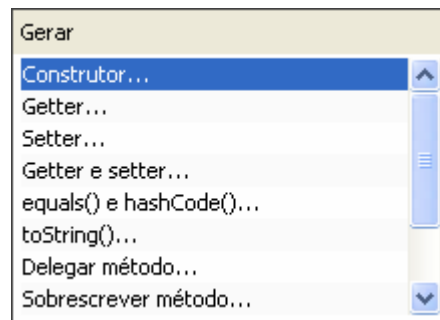
```
    public String getCor() {  
        return cor;  
    }  
    public void setCor(String cor) {  
        this.cor = cor;  
    }  
    public String getDescricao() {  
        return descricao;  
    }  
    public void setDescricao(String descricao) {  
        this.descricao = descricao;  
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

Precisamos fechar a classe.

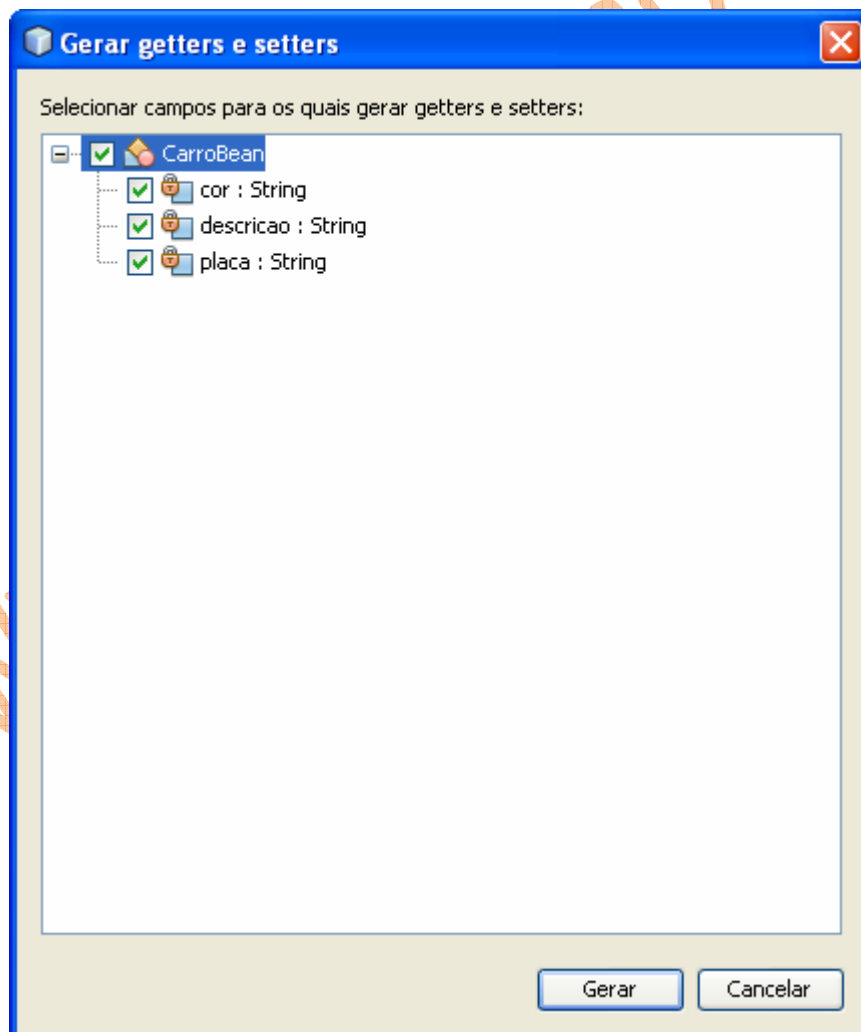
```
}
```

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

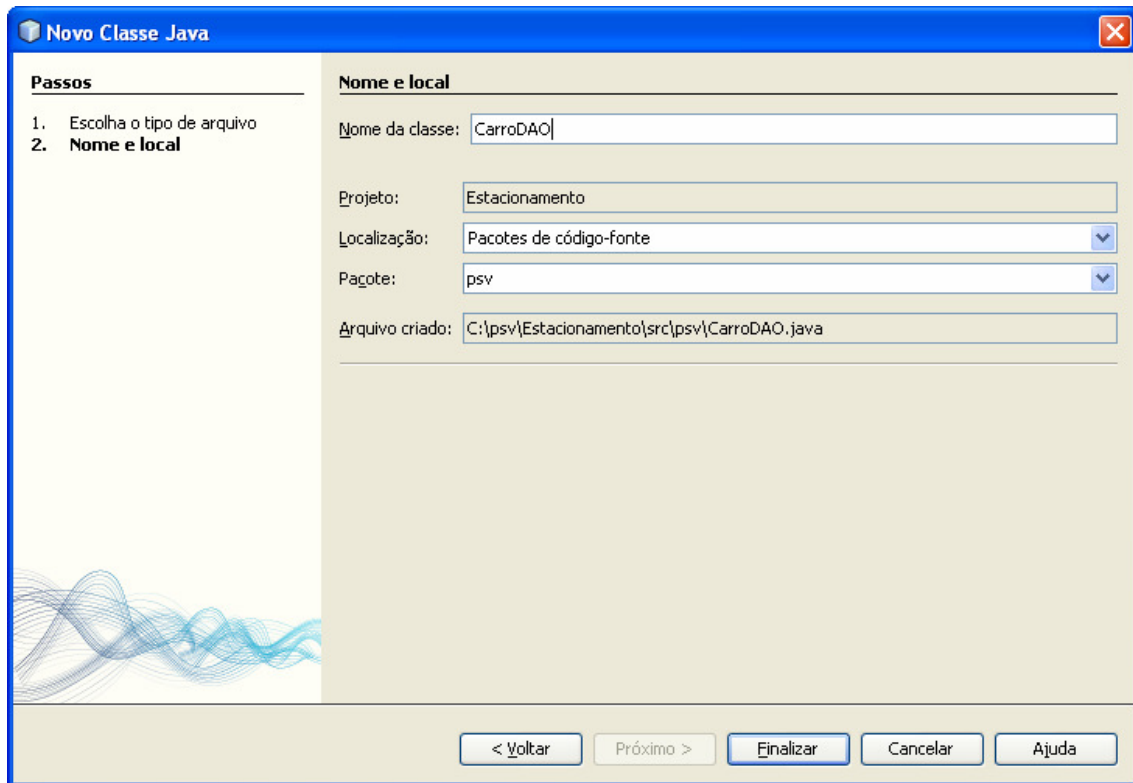
1 – pressione alt+insert e aparecerá a figura abaixo.



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 psv; //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 CarroBen 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,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.setString(2, carro.getCor());

ps.setString(3, 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 `CarroBen` 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 cor = ?,descricao = ?";

    sql += " where placa = ?";

    try {

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

```

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

        ps.setString(2, carro.getDescricao());

        ps.setString(3, 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 CarroBen 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, `listarTodos`. 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.setCor(rs.getString(2));

    cb.setDescricao(rs.getString(3));

    listaCarro.add(cb);

}

return listaCarro;

} else {

    return null;

}

```

As linhas a seguir já foram explicadas anteriormente.

```

    } catch (SQLException e) {

        return null;

    }

}

```

Com isso temos todas as classes necessárias, só nos resta fazer um teste para ver se estão funcionando realmente. Não criaremos interface ainda (ficará para a parte 2 deste tutorial), por isso vamos criar uma classe de teste (Teste.java).

Crie uma classe chamada Teste com o método main.

```

package psv;

import java.sql.*;

import java.util.*;

public class Teste {

    public static void main(String[] args) {

    }

}

```

Agora testaremos as funcionalidades uma a uma:

1 – testando a conexão: dentro do método main coloque a linha de comando abaixo e execute a classe (Shift+F6). Se você ver a mensagem “Conexão aberta” pode seguir em frente, se não, revise a classe Conexao e verifique os passos anteriores: banco, driver,...

```
Connection con = Conexao.abrirConexao();
```

2 – criando os objetos necessários para a manutenção da tabela:

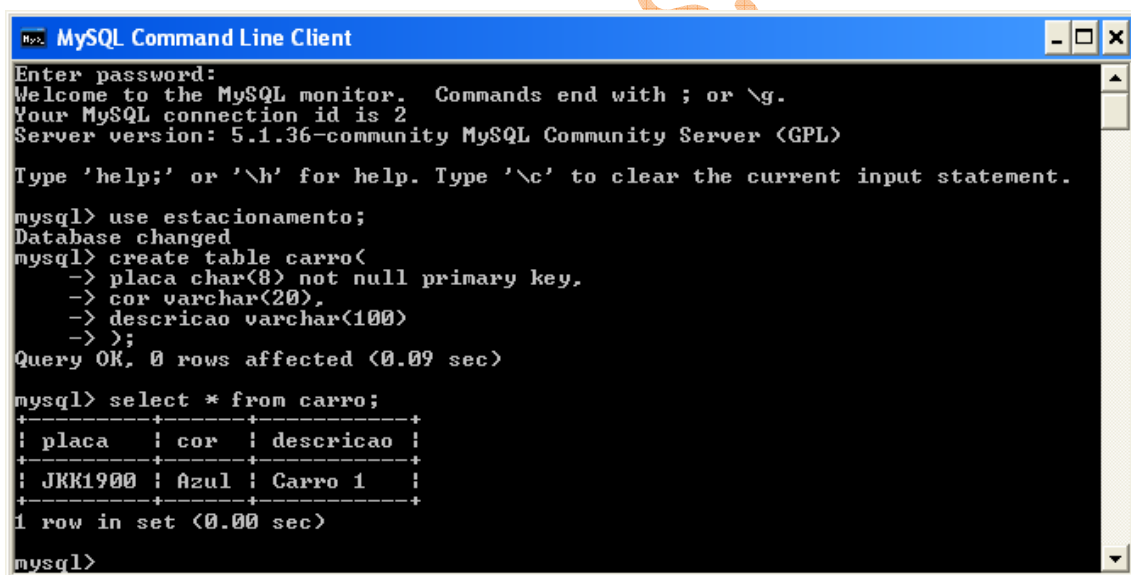
```
CarroBean cb = new CarroBean();
```

```
CarroDAO cd = new CarroDAO(con);
```

3 - Testando o método inserir: faça a seguinte operação.

```
cb.setPlaca("JKK1900");  
  
cb.setCor("Azul");  
  
cb.setDescricao("Carro 1");  
  
System.out.println(cd.inserir(cb));
```

Verifique no seu banco de dados se o registro foi inserido, conforme figura a seguir.



The screenshot shows a MySQL Command Line Client window. The title bar is blue with the text "MySQL Command Line Client". The main window has a black background with white text. The text inside the window is as follows:

```
Enter password:  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 2  
Server version: 5.1.36-community MySQL Community Server (GPL)  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
mysql> use estacionamento;  
Database changed  
mysql> create table carro(  
-> placa char(8) not null primary key,  
-> cor varchar(20),  
-> descricao varchar(100)  
-> );  
Query OK, 0 rows affected (0.09 sec)  
  
mysql> select * from carro;  
+-----+-----+-----+  
| placa | cor  | descricao |  
+-----+-----+-----+  
| JKK1900 | Azul | Carro 1  |  
+-----+-----+-----+  
1 row in set (0.00 sec)  
  
mysql>
```

Se o registro foi inserido, insira mais uns dois carros, mas lembre-se, não repita a mesma placa porque esse é um campo chave. Faça novamente a consulta.

```
MySQL Command Line Client
-> placa char(8) not null primary key,
-> cor varchar(20),
-> descricao varchar(100)
-> ;
Query OK, 0 rows affected (0.09 sec)

mysql> select * from carro;
+-----+-----+-----+
| placa | cor  | descricao |
+-----+-----+-----+
| JKK1900 | Azul | Carro 1 |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from carro;
+-----+-----+-----+
| placa | cor  | descricao |
+-----+-----+-----+
| JKE2013 | Preto | Carro 2 |
| JKK1900 | Azul | Carro 1 |
| JKL2897 | Verde | Carro 3 |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

4 - Testando o método alterar: escolha um dos carros e faça algumas alterações, observe que a placa não deve ser alterada e ela deve existir no banco.

```
cb.setPlaca("JKL2897");

cb.setCor("Amarelo");

cb.setDescricao("Carro 3");

System.out.println(cd.alterar(cb));
```

Faça a consulta novamente. Aqui o carro verde foi alterado para carro amarelo.

```
MySQL Command Line Client

+-----+-----+-----+
| JKK1900 | Azul | Carro 1 |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from carro;
+-----+-----+-----+
| placa | cor  | descricao |
+-----+-----+-----+
| JKE2013 | Preto | Carro 2 |
| JKK1900 | Azul | Carro 1 |
| JKL2897 | Verde | Carro 3 |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from carro;
+-----+-----+-----+
| placa | cor  | descricao |
+-----+-----+-----+
| JKE2013 | Preto | Carro 2 |
| JKK1900 | Azul | Carro 1 |
| JKL2897 | Amarelo | Carro 3 |
+-----+-----+-----+
3 rows in set (0.00 sec)

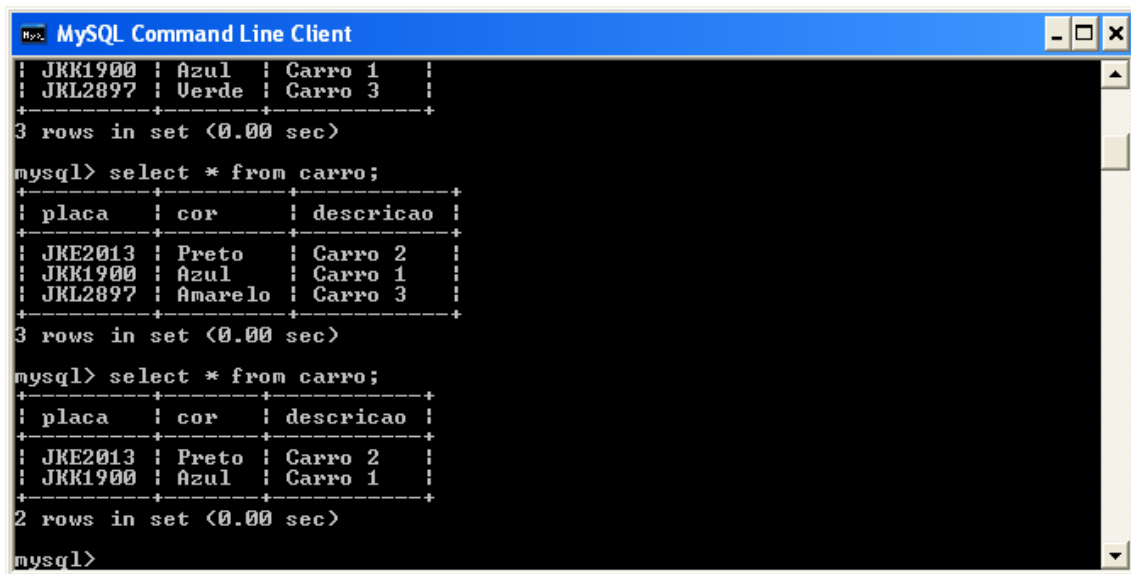
mysql>
```

5 - Testando o método excluir: vamos excluir o carro com placa JKL2897.

```
cb.setPlaca("JKL2897");

System.out.println(cd.excluir(cb));
```

Faça a consulta novamente.



```
MySQL Command Line Client
+----+-----+-----+
| JKK1900 | Azul  | Carro 1 |
| JKL2897 | Verde | Carro 3 |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from carro;
+----+-----+-----+
| placa | cor   | descricao |
+----+-----+-----+
| JKE2013 | Preto | Carro 2 |
| JKK1900 | Azul  | Carro 1 |
| JKL2897 | Amarelo | Carro 3 |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from carro;
+----+-----+-----+
| placa | cor   | descricao |
+----+-----+-----+
| JKE2013 | Preto | Carro 2 |
| JKK1900 | Azul  | Carro 1 |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

6 - Testando o método listarTodos: como o método retorna uma `List`, precisamos criar uma para receber o resultado. O código é mostrado a seguir, observe que estamos usando `ForEach` para percorrer a lista. Se você não está familiarizado com o assunto, pesquise material a respeito do `FrameWork Collection`.

```
List<CarroBean> lista = cd.listarTodos();

if(lista != null){
    for(CarroBean carro : lista){
        System.out.println("Placa: "+carro.getPlaca());
        System.out.println("Cor: "+carro.getCor());
        System.out.println("Descrição: "+carro.getDescricao());
    }
}
```

Resultado

Placa: JKE2013

Cor: Preto

Descrição: Carro 2

Placa: JKK1900

Cor: Azul

Descrição: Carro 1

Com isso você já está pronto para criar a interface para o usuário fazer manutenção da tabela carro.

6. Considerações finais

Existem várias maneiras de fazer o que fizemos aqui, isso depende do grau de conhecimento de cada um.

Na parte 2 deste tutorial, criaremos a interface (desktop) para trabalhar com essas classes.

O código completo das classes é colocado a seguir.

Classe Conexao.java

```
package psv;

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/estacionamento?";

            url += "user=root&password=";

            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());
        }
    }
}

```

Classe CarroBean.java

```

package psv;

public class CarroBean {
    private String placa;
    private String cor;
    private String descricao;

    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;

    }

}

```

Classe CarroDAO.java

```

package psv;

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 into
carro(placa,cor,descricao)values(?,?,?)";

        try {

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

```

```

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

        ps.setString(2, carro.getCor());

        ps.setString(3, 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 cor = ?,descricao = ?";

    sql += " where placa = ?";

    try {

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

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

        ps.setString(2, carro.getDescricao());

        ps.setString(3, 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.setCor(rs.getString(2));
                cb.setDescricao(rs.getString(3));
                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;
}
}
```

Classe Teste.java

```
package psv;

import java.sql.*;

import java.util.*;

public class Teste {

    public static void main(String[] args) {

        Connection con = Conexao.abrirConexao();

        CarroBean cb = new CarroBean();

        CarroDAO cd = new CarroDAO(con);

        //Testando método inserir
        /* cb.setPlaca("JKL2897");
        cb.setCor("Verde");
        cb.setDescricao("Carro 3");

        System.out.println(cd.inserir(cb));*/

        //Testando método alterar
        /*cb.setPlaca("JKL2897");
        cb.setCor("Amarelo");
        cb.setDescricao("Carro 3");

        System.out.println(cd.alterar(cb));*/

        //Testando excluir
        /* cb.setPlaca("JKL2897");

        System.out.println(cd.excluir(cb));*/
```

```
List<CarroBean> lista = cd.listarTodos();

if(lista != null){

    for(CarroBean carro : lista){

        System.out.println("Placa: "+carro.getPlaca());

        System.out.println("Cor: "+carro.getCor());

        System.out.println("Descrição:
"+carro.getDescricao());

    }

}

Conexao.fecharConexao(con);

}

}
```