VOLTAR

Conexão com Bases de Dados

Chegou a hora, finalmente, de trabalhamos com conexão a bancos de dados no Java. Para isso, usaremos o MySql como sistema gerenciador de banco de dados. Você aprenderá aqui como conectar ao banco, inserir um valor e consultar valores. Prepare-se para programar de verdade agora.

NESTE TÓPICO

O que preciso saber, antes?

Antes de programar em Java com conexões à bancos de dados, é preciso saber algumas coisas importantes, sem as quais, não se começa ou não se programa.

Você precisa saber (lembrar) que o Java é uma linguagem de programação. O banco de dados é um banco de dados (sim!) ou seja, são coisas diferentes. São tecnologias diferentes. São, normalmente, de fabricantes diferentes.

Por isso, a primeira coisa que você precisa saber para trabalhar com bacos de dados em Java é que será preciso usar um driver de conexão, implementado através de uma API que, normalmente, é fornecida pelo fabricante do banco de dado.

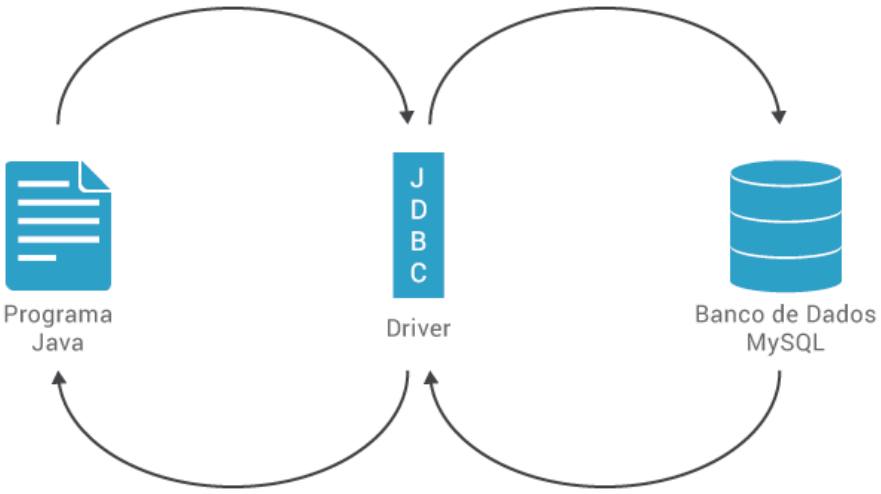
Outra coisa importante é que os conceitos de tratamento de exceções devem estar muito claros para você, pois toda comunicação entre a programação e o banco, correrá dentro de um try..catch, inclusive a abertura e fechamento de conexões.

E claro, quando falamos de bancos de dados, estamos falando de coleção de dados, ou seja, você precisa estar dominando algumas coisas da linguagem, como loops (laço) e coleções de dados (vetor, array etc.)

Por fim, mas não menos importante, você precisa, claro, conhecer um pouco de bancos de dados: insert, update, select, select com join etc., para poder compreender os exemplos e exercícios mostrados aqui.

Drive de conexão

Como mencionado acima para cada fabricante de bancos de dados, é preciso usar um driver de conexão entre o Java e a linguagem. Esse driver é um conjunto de código que possui os principais métodos de conexão, inserção, atualização, deleção etc., para serem implementados pela linguagem.



Relação Java X Driver JDBC

O drive de conexão que usaremos se chama JDBC, que significa *Java Database Connectivity* ou, Conectividade para Bancos de Dados do Java, em português. O JDBC para MySQL já vem no NetBeans. Se você precisar utilizar um outro conector (driver), será preciso entrar no site do fabricante do Banco de Dados para baixa-lo e acrescenta-lo ao seu projeto (isso será mostrado em instantes).

Para trabalharmos com Java e MySQL, instale em sua máquina (se ainda não o fez) o MySQL Server. Trabalharemos com um banco de dados local. Para baixar o MySQL, que atualmente pertence a Oracle (assim como o Java), entre em http://dev.mysql.com/downloads (http://dev.mysql.com/downloads/) e faça o download da versão *Community Server*. É preciso realizar um pequeno cadastro para poder fazer o download.

Se você preferir, instale também (depois de instalar o servidor) o *MySQL Workbench*, um gerenciador visual para bancos MySQL, com muitos recursos interessantes e poderosos. Para baixar o Workbench, acesse: http://dev.mysql.com/downloads/workbench/ (http://dev.mysql.com/downloads/workbench/)(também da Oracle).

CRIANDO UM NOVO USUÁRIO NO BANCO

Dica: Depois que o servidor de banco é instalado em sua máquina, é preciso trocar a senha de root, se não foi definida pela instalação. Não deixe de troca-la. Por segurança e para a programação, usaremos um usuário diferente do usuário “root”. Não deixe de criar um novo usuário em seu banco para acesso aos recursos.

Para criar um novo usuário no MySQL, você pode utilizar a seguinte linha de comando no console do servidor:

```
CREATE USER 'USUARIO'@'localhost' IDENTIFIED BY 'SENHA';
```

Não esqueça de substituir a palavra “usuário” pelo nome de usuário que deseja criar e o mesmo para a palavra “senha”.

Tendo em vista que é necessário o uso de uma API externa para trabalharmos com bancos de dados temos, então, que acrescentar a API de programação ao nosso projeto. Depois de criar um projeto, veja em sua estrutura uma pasta a qual ainda não falamos, chamada “bibliotecas”. É nessa pasta onde colocamos bibliotecas (APIs) e programas externos a nossa aplicação para utilizar seus recursos. Note que a API do próprio Java já está dentro dessa biblioteca.

Para acrescentar (e referenciar) o drive de conexão com o MySQL em seu programa, clique com o botão direito do mouse sobre essa pasta (“bibliotecas”) e clique em “Adicionar Biblioteca”. No assistente que abrir, selecione “Driver JDBC do MySQL”. O vídeo abaixo mostra como fazer isso:

./videos/370386473.mp4

Hora de programar, mas antes, vamos preparar a base de dados

Com tudo pronto e um usuário do banco de dados criado, podemos começar a programar. Contudo, precisamos preparar uma base de dados para fazer isso.

Imagine que você foi contratado para desenvolver um pequeno sistema para uma concessionária de veículos, ou seja, um sistema que permite você acrescentar, editar e remover veículos de uma base de dados da empresa. Por enquanto vamos trabalhar com uma única tabela no banco de dados: a tabela de carros.

A tabela de carros é composta pelas características relevantes do carro para um sistema de vendas: Fabricante, modelo, ano de fabricação, cor, quilometragem e valor. Podemos representar a tabela de carros da seguinte forma visual:

tb_Carros
- idCarro: int (PK, AI) - fabricante: varchar(32) - modelo: varchar(45) - anoFabricacao: integer - cor: varchar(16) - km: decimal(8,2) - valor: decimal (10,2)

Estrutura da tabela de Carros

Por sugestão, crie um novo schema em seu banco de dados, ou seja, depois de instalar, ele vem com um único schema que é o “sys”. Não use este pois é reservado para tabelas do próprio servidor.

CRIANDO UM NOVO SCHEMA

Para criar um schema novo, digite, no console do servidor: CREATE SCHEMA `NOMEDOSCHEMA` ;

Substitua a palavra “NOMEDOSCHEMA” pelo nome do schema que você quiser.

Depois criar e setar seu schema como padrão (onde você irá trabalhar), vamos criar a tabela de carros. O script de criação da tabela pode ser visto abaixo:

```
CREATE TABLE `tb_carros` (
  `idCarro` INT NOT NULL AUTO_INCREMENT,
  `fabricante` VARCHAR(32) NULL,
  `modelo` VARCHAR(45) NULL,
  `anoFabricacao` INT NULL,
  `cor` VARCHAR(16) NULL,
  `km` DECIMAL(8,2) NULL,
  `valor` DECIMAL(10,2) NULL,
  PRIMARY KEY (`idCarro`));
```

Vamos inserir alguns carros na tabela, para já trabalharmos com uma base com dados, com os seguintes comando:

```
INSERT INTO `tb_carros` (`fabricante`, `modelo`, `anoFabricacao`, `cor`, `km`, `valo
r`)
      VALUES ('Audi', 'R8', '2017', 'Azul', '0', '100000');

INSERT INTO `tb_carros` (`fabricante`, `modelo`, `anoFabricacao`, `cor`, `km`, `valo
r`)
      VALUES ('Ferrari', 'F250', '2001', 'Vermelha', '20000', '800000');

INSERT INTO `tb_carros` (`fabricante`, `modelo`, `anoFabricacao`, `cor`, `km`, `valo
r`)
      VALUES ('BMW', 'M6', '2016', 'Preta', '14500', '450000');

INSERT INTO `tb_carros` (`fabricante`, `modelo`, `anoFabricacao`, `cor`, `km`, `valo
r`)
      VALUES ('Volvo', 'XC90', '2016', 'Branca', '100', '600000');
```

Pronto. Inserimos dados de 4 carros para trabalharmos. Se quiser testar e verificar se é tabela de carros está completa, faça uma seleção dos dados com o comando: *select * from tb_carros;* o resultado deve ser como visto abaixo:

```
mysql> select * from tb_carros;
+---------+------------+--------+---------------+----------+----------+-----------+
| idCarro | fabricante | modelo | anoFabricacao | cor      | km       | valor     |
+---------+------------+--------+---------------+----------+----------+-----------+
|       1 | Audi       | R8     |          2017 | Azul     |     0.00 | 100000.00 |
|       2 | Ferrari    | F250   |          2001 | Vermelha | 20000.00 | 800000.00 |
|       3 | BMW        | M6     |          2016 | Preta    | 14500.00 | 450000.00 |
|       4 | Volvo      | XC90   |          2016 | Branca   |   100.00 | 600000.00 |
+---------+------------+--------+---------------+----------+----------+-----------+
4 rows in set (0,00 sec)

mysql>
```

Resultado do select na tabela de carros.

Bem, bom a base pronta, podemos voltar ao nosso projeto Java. Fazermos uma aplicação que consulta, cadastra e altera dados dos veículos.

Programando pra bancos de dados

Para este programa, teremos uma única classe para fazer a conexão com o banco de dados, que será reaproveitada pelas demais classes de manipulação de dados.

Essa classe chama-se “Conexao” e pode ser vista abaixo. Leia atentamente cada linha de código e veja os comentários, explicando cada trecho. É muito importante o entendimento de como um objeto dessa classe funciona.

```
1.  //Pacote
2.  package br.uninove.poo.dao;
3.
4.  //Importações
5.  import com.mysql.jdbc.Connection;
6.  import java.sql.DriverManager;
7.  import java.sql.SQLException;
8.
9.  public class Conexao {
10.
11.     //Status da conexão, para consultas
12.     private String statusConexao = "Não conectado";
13.
14.     public Connection getConexao() {
15.         //Atributo para conexão
16.         Connection conexao = null;
17.
18.         //Começa a tentativa de conexão
19.         try {
20.
21.             //Designa o driver de conexão padrão para essa conexão:
22.             String driverName = "com.mysql.jdbc.Driver";
23.             Class.forName(driverName);
24.
25.             //Seta os parâmetros da conexão ao banco -IMPORTANTE AQUI!!!-
26.             String servidor = "localhost"; //Caminho de rede do BD
27.             String schema = "uninove";   //Nome do schema que será usado
28.
29.             //URL para o driver JDBC:
30.             String url = "jdbc:mysql://" + servidor + "/" + schema;
31.
32.             //Parâmetros da conexão (usuário e senha)
33.             //Atenção: COLOQUE AQUI O SEU USUÁRIO E A SUA SENHA!
34.             String usuario = "uninove";    //Usuário da base, criado anteriormente
35.             String senha = "Senha123";   //Senha do usuário
36.
37.             //seta a conexão:
38.             conexao = (Connection) DriverManager.getConnection(url, usuario, senha);
39.
40.             //Testando a conexão
41.             if (conexao != null) {   //Se aconexão for ok, ela não é nula...
42.                 //Troca o status da conexão
43.                 statusConexao = ("Conetado");
44.             } else {
45.                 statusConexao = ("Não conectado");
46.             }
47.
48.             //retorna a conexão para quem for utiliza-la
49.             return conexao;
50.
51.         } catch (ClassNotFoundException e) {  //Caso não encontre o driver de conexã
    o
52.             System.out.println("Driver de conexão não encontrado    ");
53.             return null;
54.         } catch (SQLException e) { //Caso não consiga conectar
55.             System.out.println("Falha na conexão: ");
56.             System.out.println(e.getMessage());
57.             return null;
58.         }
59.     }
60.
61.     //Retorna o status da conexão
62.     public String getStatusConexao() {
63.         return statusConexao;
64.     }
65.
66.     //Método para fechar (encerrar) a conexão
67.     public boolean FechaConexao() {
68.         try {
69.             getConexao().close(); //Fecha
70.             statusConexao = "Conexão fechada"; //Atualiza o status
```

```
71.             return true; //Retorna verdadeiro = conseguiu fechar!
72.         } catch (SQLException e) {
73.             //Se der algum erro, imprime
74.             System.out.println(e.getMessage());
75.             return false;
76.         }
77.     }
78.
79.     //caso precise reiniciar sua conexão
80.     public Connection ReiniciaConexao() {
81.         FechaConexao(); //Fecha
82.         return getConexao(); //Abre e retorna
83.     }
84. }
```

Note, na classe acima que o schema utilizado, assim como o usuário e senha do banco deste exemplo estão no código. Não se esqueça de alterar para o nome do schema que você criou e alterar, também, o usuário e a senha no método de obtenção da conexão.

É muito importante organizar o código, pois teremos projetos cada vez maiores agora. A organização é primordial para lhe auxiliar na manutenção e alteração de seu projeto. Por isso, não deixe de notar que a classe "Conexao" foi criar em um pacote chamado "br.uninove.poo.dao".

DAO, em "Javanês", quer dizer "*Data Access Object*", ou ainda, Objeto de Acesso a Dados. Criou-se um pacote chamado "dao" para organização das classes de manipulação de dados. As demais classes que envolverem acesso a manipulação ao banco de dados, devem ficar neste pacote.

Antes de programarmos a classe de manipulação (consulta, inserção e alteração) da tabela de carros, precisaremos de uma classe que representa um carro, pois lembre-se que Java é uma linguagem orientada a objetos e tudo é tratado como um objeto. Isso quer dizer que cada linha na tabela (que representa um carro) terá os seus dados mareados para uma classe que representa este carro. Essa classe precisa ser encapsulada, como boas práticas de programação.

Portanto, vamos criar um outro pacote, para colocar nossas classes que fazem parte da lógica de negócio. Neste caso, o pacote criado chama-se "br.uninove.poo.negocio" e a casse "Carro" pode ser vista abaixo. Veja atentamente a classe, pois os comentários explicam sobre seu funcionamento.

```java
package br.uninove.poo.negocio;

public class Carro {

    //Atributos possíveis de um carro para este sistema
    private String fabricante, modelo, cor;
    private int ano, idCarro;
    private double km, valor;

    //Sets e gets dos atributos...:
    public String getFabricante() {
        return fabricante;
    }

    public void setFabricante(String fabricante) {
        this.fabricante = fabricante;
    }

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

    public String getCor() {
        return cor;
    }

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

    public int getAno() {
        return ano;
    }

    public void setAno(int ano) {
        this.ano = ano;
    }

    public int getIdCarro() {
        return idCarro;
    }

    public void setIdCarro(int idCarro) {
        this.idCarro = idCarro;
    }

    public double getKm() {
        return km;
    }

    public void setKm(double km) {
        this.km = km;
    }

    public double getValor() {
        return valor;
    }

    public void setValor(double valor) {
        this.valor = valor;
    }
    //fim dos sets e gets
}
```

Pronto, temos um carro implementado e temos uma classe de conexão. Vamos implementar agora a classe que faz a manipulação dos carros. Para uma primeira versão dessa classe (ou seja, teremos modificações nela adiante), considere apenas o método que retorna a lista completa de carros. Lembre-se que, como Java é orientado a objetos, precisaremos trabalhar com objetos do tipo lista de carros.

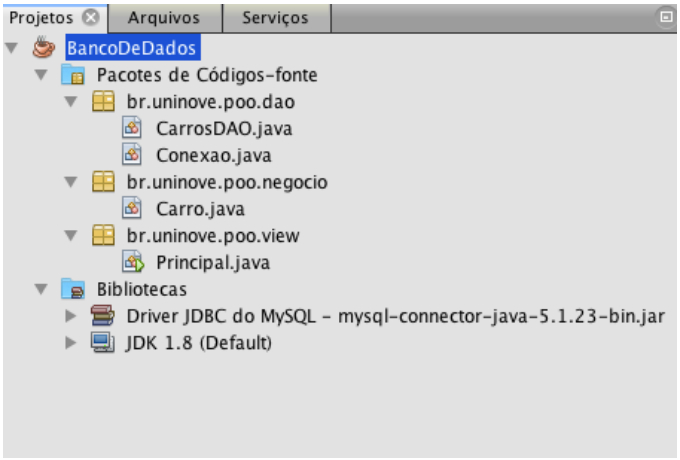
A classe abaixo chama-se “CarrosDAO”, pois representa a classe que é capaz de manipular o banco de dados. Onde essa classe será criada? Isso mesmo, no pacote “dao”.

```
1.  package br.uninove.poo.dao;
2.
3.  import br.uninove.poo.negocio.Carro;
4.  import java.sql.PreparedStatement;
5.  import java.sql.ResultSet;
6.  import java.util.ArrayList;
7.  import java.util.List;
8.
9.  public class CarrosDAO {
10.
11.     //Método que retorna a lsta de carros, com todos os carros do sistema
12.     public List<Carro> getListaDeCarros() {
13.
14.         try {
15.             //Conecta ao banco através da classe de conexão:
16.             Conexao con = new Conexao();
17.             con.getConexao();
18.
19.             //Select na tablea de carros:
20.             String sql = "select * from tb_carros;";
21.
22.             //Executa a query
23.             PreparedStatement comando = con.getConexao().prepareStatement(sql);
24.             ResultSet resultado = comando.executeQuery();
25.
26.             //Prepara a lista de carros para retornar
27.             List<Carro> listaDeCarros = new ArrayList<Carro>();
28.
29.             //para cada item retonrado no select, faça...
30.             while (resultado.next()) {
31.                 Carro c = new Carro(); //Cria um carro novo na memória
32.                 c.setIdCarro(resultado.getInt("idCarro")); //Seta o ID dele
33.                  c.setFabricante(resultado.getString("fabricante")); //Seta o fabrica
    nte
34.                 c.setModelo(resultado.getString("modelo")); //Seta o modelo
35.                 c.setCor(resultado.getString("cor")); //Seta a cor
36.                 c.setKm(resultado.getDouble("km")); //Seta a KM
37.                 c.setValor(resultado.getDouble("valor")); //Seta o valor do carro
38.                  c.setAno(resultado.getInt("anoFabricacao")); //Seta o ano de fabrica
    ção
39.
40.                 //insere o carro na lista local
41.                 listaDeCarros.add(c);
42.             }
43.
44.             //Ao terminar o laço, fecha a conexão
45.             resultado.close();
46.             comando.close();
47.             con.getConexao().close();
48.
49.             //Retorna a lista de carros
50.             return listaDeCarros;
51.
52.         } catch (Exception e) { //Se der algum exessão...
53.             System.out.println(e.getMessage());
54.             return null;
55.         }
56.     }
57. }
```

Quase tudo está pronto. Vamos criar uma classe principal, agora, que é capaz de listar todos os carros cadastrados no sistema e mostrar no console.

Para mantemos a organização, vamos criar um novo pacote de código-fonte. Vamos chama-lo de “view”, pois irá conter as classes que são responsáveis pela parte “visual” do projeto. Para mantermos o padrão e boas práticas de programação, o pacote se chamará “br.uninove.poo.view”.

A essa altura, o seu projeto deve estar com essa estrutura:

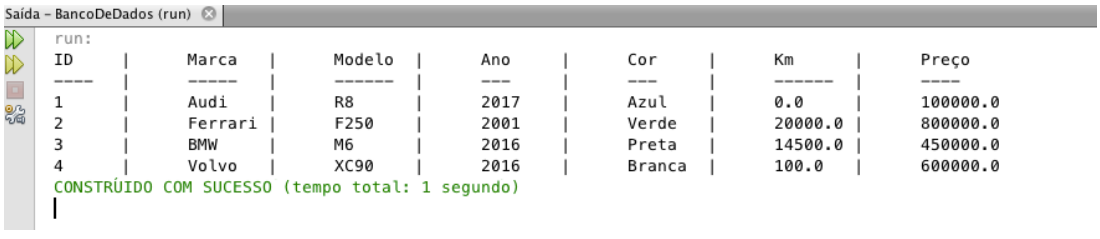


Estrutura do projeto de banco de dados com Java no NetBeans.

Dentro da classe principal, teremos um método para listagem e impressão dos carros. Veja uma possível implementação da classe principal com um método que chama a listagem de carros (com o select, na classe CarrosDAO) e imprime cada item da lista, iterando-a:

```
1.  package br.uninove.poo.view;
2.
3.  //Importa as classes necessárias
4.  import br.uninove.poo.dao.CarrosDAO;
5.  import br.uninove.poo.negocio.Carro;
6.  import java.util.ArrayList;
7.  import java.util.List;
8.
9.  public class Principal {
10.
11.     public static void main(String args[]) {
12.         listaTodosCarros();
13.     }
14.
15.     //Método de listagem de todos os carros
16.     public static void listaTodosCarros() {
17.
18.         //Cria uma instância de CarrosDAO na memória
19.         CarrosDAO carro = new CarrosDAO();
20.
21.         //Cria a lista de carros local, que será preenchida
22.         List<Carro> listaDeCarros = new ArrayList<Carro>();
23.
24.         //Obtém a lista de carros através do objeto
25.         listaDeCarros = carro.getListaDeCarros();
26.
27.         //Começa a imprimir os dados
28.           System.out.println("ID\t|\tMarca\t|\tModelo\t|\tAno\t|\tCor\t|\tKm\t|\tPreç
    o");
29.           System.out.println("----\t|\t-----\t|\t------\t|\t---\t|\t---\t|\t------\t|
    \t----");
30.         for (Carro carroLocal : listaDeCarros) { //Iterator: Para cada carro na list
    de carros...
31.             System.out.print(carroLocal.getIdCarro() + "\t|\t");
32.             System.out.print(carroLocal.getFabricante() + "\t|\t");
33.             System.out.print(carroLocal.getModelo() + "\t|\t");
34.             System.out.print(carroLocal.getAno() + "\t|\t");
35.             System.out.print(carroLocal.getCor() + "\t|\t");
36.             System.out.print(carroLocal.getKm() + "\t|\t");
37.             System.out.print(carroLocal.getValor());
38.             System.out.println();
39.         }
40.     }
41. }
```

E o resultado desta primeira versão de nosso programa pode ser visto abaixo:



Resultado da primeira execução do projeto de listagem de carros com Java.

Bem, já conseguimos listar os carros. Hora de começar a acrescentar um carro ao banco de dados. Para isso, temos que modificar nossa classe “CarrosDAO”, acrescentando um novo método, para inserção de um carro no banco.

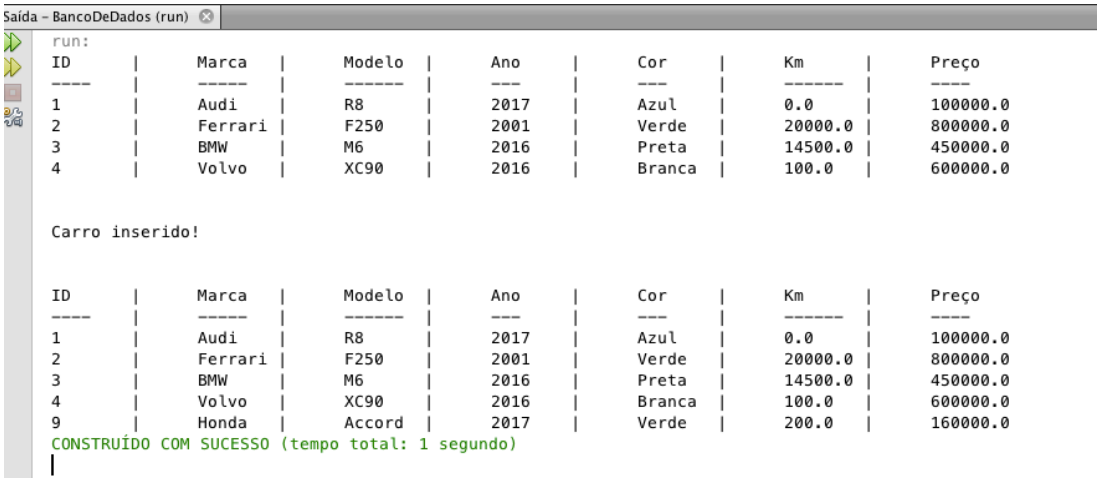
O método abaixo descrever um possível mecanismo de inserção de um carro no banco:

```
//...
    public boolean insereCarro(Carro c) {
        try {
            //Seta e abre a conexão
            Conexao conexao = new Conexao();
            conexao.getConexao();

            //Query de inserção (as aspas são muit importantes):
            String sql = "";
            sql += "insert into tb_Carros (`fabricante`, `modelo`, "
                    + "`anoFabricacao`, `cor`, `km`, `valor`)";
            sql += " VALUES ";
            sql += " ('" + c.getFabricante()+ "', '" + c.getModelo() + "', "
                    + c.getAno() + ",'" + c.getCor() + "', " + c.getKm() + ", "
                    + c.getValor() + ")";

            //Executa
            PreparedStatement comando = conexao.getConexao().prepareStatement(sql);
            comando.executeUpdate(sql);
            return true; //se inseriu, retorna verdadeiro.

        } catch (Exception e) { //Se deu algum erro...
            System.out.println(e.getMessage());
            return false;
        }
    }
//...
```

Vamos chama-lo na classe principal, para isso, teremos de modifica-la. Uma possível reimplementacão da casse principal pode ser vista abaixo:

```
package br.uninove.poo.view;

//Importa as classes necessárias
import br.uninove.poo.dao.CarrosDAO;
import br.uninove.poo.negocio.Carro;
import java.util.ArrayList;
import java.util.List;

public class Principal {

    public static void main(String args[]) {
        listaTodosCarros();

        //Vamos inserir um carro novo no banco:
        //Cria a instância e seta os valores (você pode ler do teclado se quiser...)
        Carro carro = new Carro();
        carro.setAno(2017);
        carro.setCor("Verde");
        carro.setFabricante("Honda");
        carro.setKm(200);
        carro.setModelo("Accord");
        carro.setValor(160000);

        //Note que não setamos o ID, pois ele é auto incrementado pelo banco!
        //Insere...
        CarrosDAO cD = new CarrosDAO();
        cD.insereCarro(carro);
        System.out.println("\n\nCarro inserido!\n\n");

        //Vamos listar todos os carros agora, para ver o novo caro na lista...
        listaTodosCarros();
    }

    //Método de listagem de todos os carros
    public static void listaTodosCarros() {

        //Cria uma instância de CarrosDAO na memória
        CarrosDAO carro = new CarrosDAO();

        //Cria a lista de carros local, que será preenchida
        List<Carro> listaDeCarros = new ArrayList<Carro>();

        //Obtém a lista de carros através do objeto
        listaDeCarros = carro.getListaDeCarros();

        //Começa a imprimir os dados
          System.out.println("ID\t|\tMarca\t|\tModelo\t|\tAno\t|\tCor\t|\tKm\t|\tPreço");
          System.out.println("----\t|\t-----\t|\t------\t|\t---\t|\t---\t|\t------\t|\t----");
        for (Carro carroLocal : listaDeCarros) { //Iterator: Para cada carro na list de carros...
            System.out.print(carroLocal.getIdCarro() + "\t|\t");
            System.out.print(carroLocal.getFabricante() + "\t|\t");
            System.out.print(carroLocal.getModelo() + "\t|\t");
            System.out.print(carroLocal.getAno() + "\t|\t");
            System.out.print(carroLocal.getCor() + "\t|\t");
            System.out.print(carroLocal.getKm() + "\t|\t");
            System.out.print(carroLocal.getValor());
            System.out.println();
        }
    }
}
```

E o resultado da execução deste código pode ser visto abaixo. Note a nova listagem dos carros, com a inserção do carro criado manualmente. Você pode cria-lo pedindo para que o usuário digite os dados do veículo (dica para estudar: altere o código para que ele funcione assim).



Resultado da execução do novo código-fonte .

Agora tente, por conta própria, criar um método para atualização de um carro no banco de dados. Será uma ótima forma de estudar. Lembre-se de criar este método dentro da classe “CarrosDAO”, pois é a classe responsável pela manipulação dos dados o banco.

Resumo dessa aula

Nesta aula, você programou bastante e aprendeu que:

- Para conexões com bancos de dados, precisamos de um driver de conexão instanciado em nosso projeto
- Precisamos criar uma classe apenas para conexão
- Criamos uma classe para cada objeto plausível do banco de dados
- Criamos uma classe para manipulação desse objeto. Uma para cada objeto existente no banco.
- É possível listar, inserir e manipular objetos diretamente na programação

Chegamos ao fim de mais uma aula. Estude bastante estes conceitos e não deixe de tirar dúvidas com o professor, caso necessário. O conceito de conexão a bancos de dados em Java é muito cobrado no mercado de trabalho. Bons estudos e boa programação!

Quiz

Exercício Final

INICIAR

Referências

Deitei P. e Deitel H., 2010, Java : Como programar, 8ª Edição, Pearson Pretice Hall

Teruel, E. C., 2015, Programação Orientada a Objetos com Java - sem mistérios - 1ª Ed., Editora Uninove

Schildt, H., 2015, Schildt, Java para iniciantes : crie, compile e execute programas Java rapidamente, Bookman

MySQL Documentation, disponível em http://dev.mysql.com/doc/, acessado dia 19/04/2016

Avalie este tópico

Ajuda? (https://ava.un idCurso=)