Curso: Análise e Desenvolvimento de Sistemas – ADS

Ano: 2023/1

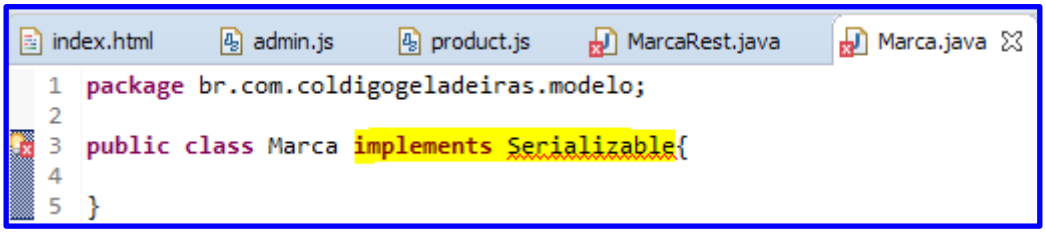
Orientação Técnica (OT) – 17

Cadastro REST - Exibir Marcas – Party #2

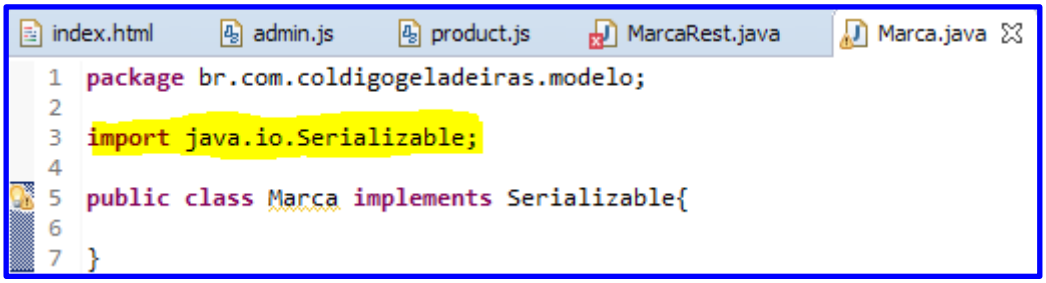
Criando a classe "modelo" para uma Marca

Como sabe desde a trilha POO, uma classe deve representar uma abstração de algo. Em projetos mais complexos, algumas classes tem "**abstrações mais abstratas**", como a "**MarcaRest**" (ou vai dizer que você consegue imaginar no mundo real o que é uma "MarcaRest"?), mas ainda há as "**abstrações mais reais**" em relação aos elementos que observamos na realidade, como é o caso de uma **marca**. Assim, criaremos essa classe que representará o **modelo de uma marca** em nosso projeto. Para isso, crie em **Java Resources - src** um novo pacote chamado **br.com.coldigogeladeiras.modelo** e, dentro dele, uma classe chamada **Marca**:

Assim que a criar, faça a alteração no novo arquivo adicionando o código destacado a seguir:



Agora importe a classe dele, mais uma vez copiando o código em destaque:



Por fim, adicione **dentro da classe Marca** o atributo em destaque:

```
public class Marca implements Serializable{

    private static final long serialVersionUID = 1L;

}
```

ATENÇÃO: A classe Serializable e o serialVersionUID são muito importantes no Java web, e podem ser melhor entendidos neste

https://www.alura.com.br/artigos/entendendo-o-serialversionuid.

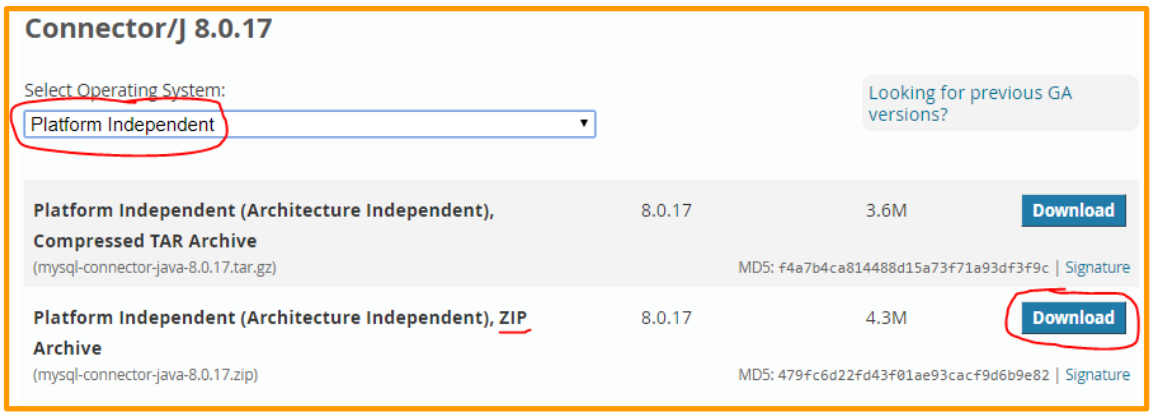
Agora que já preparamos a classe, podemos criar os atributos e métodos que nos interessam, copiando o código **entre as áreas destacadas**:

```
    private static final long serialVersionUID = 1L;

    private int id;
    private String nome;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }

}
```

Veja que são os **mesmos atributos da tabela marcas do BD**! E falando no BD...

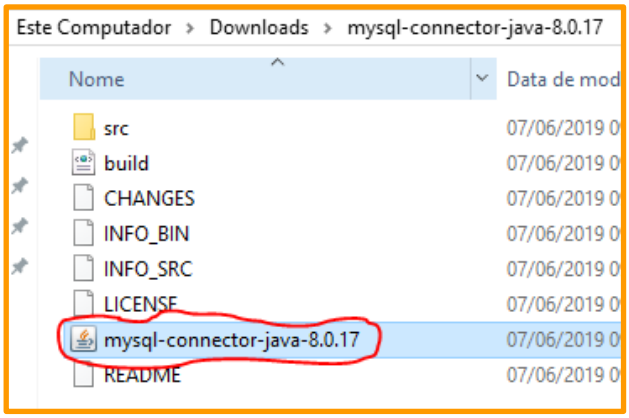
Criando a conexão com o BD

Para isso, precisaremos de um **driver de conexão ao BD MySQL**. Para baixá-lo, acesse https://dev.mysql.com/downloads/connector/j/ e selecione as opções marcadas na imagem abaixo:

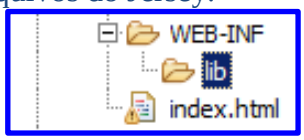


Mais uma vez, **use a versão mais recente**… Após clicar em **Download**, você será convidado a fazer uma conta na Oracle ou se autenticar, mas se não quiser, é só clicar logo abaixo em **No thanks, just start my download.**

Após o download, descompacte o arquivo e copie o arquivo em destaque na imagem abaixo:

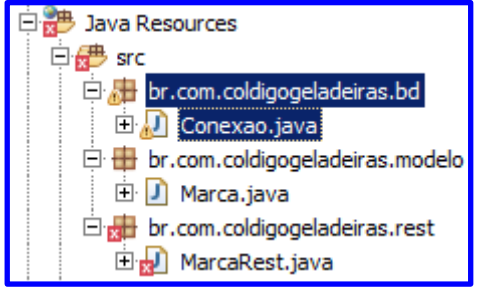


Cole-o na pasta **lib**, junto aos arquivos do Jersey:



Pronto! **Nosso projeto já tem o que é necessário para que possamos nos conectar ao banco de dados através dele.**

Nosso próximo passo é **criar uma classe Java para realizarmos essa conexão** através dela. Assim, crie o pacote e classe indicados na imagem abaixo:



Agora vamos editar essa classe com o que precisamos. Insira nela o código conforme exibido abaixo:

MarcaRest.java | Marca.java | Conexao.java

```
1  package br.com.coldigogeladeiras.bd;
2
3  import java.sql.Connection;
4
5  public class Conexao {
6
7      private Connection conexao;
8
9  }
```

Perceba que criamos um atributo **conexao** do tipo **Connection**. Ele *guardará a conexão que criaremos com o BD*, para que caso precisemos executar mais de um comando no BD possamos reaproveitá-la, ao invés de criá-la várias vezes.
Agora, vamos programar o método **abrirConexao,** que criará a conexão com o BD. Para isso, copie o código da imagem abaixo entre as duas partes destacadas:

```
public class Conexao {

    private Connection conexao;

    public Connection abrirConexao(){
        try{
            Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
            conexao = java.sql.DriverManager.
                    getConnection("jdbc:mysql://localhost/bdcoldigo?"
                            + "user=root&password=root&useTimezone=true&serverTimezone=UTC");
        }catch(Exception e){
            e.printStackTrace();
        }
        return conexao;
    }

}
```

Dentro dele, destacamos que:

- Na primeira linha do *try*, com o **Class.forName,** indicamos o Driver JDBC que estamos usando (ele está no arquivo que acabamos de adicionar ao projeto). *Cuidado* para não errar esse nome!
- Na linha seguinte, **abrimos a conexão** passando as configurações dela como parâmetro **e a armazenamos no atributo conexao** criado anteriormente. Destaque para as partes sublinhadas em vermelho que, respectivamente, representam o nome do banco, e o usuário e senha usados para a conexão.
- Depois do *catch*, **retornamos o atributo conexao** para que possa ser usada por quem chamar o **abrirConexao**.

Bem, já que estamos aqui no arquivo que gerencia a conexão com o BD, **tão importante quanto abrir uma conexão é fechá-la**, para evitarmos problemas como *uso indesejado da conexão* e *travamento do servidor por múltiplas conexões ao banco abertas simultaneamente*. Vamos criar para isso o método **fecharConexao**, logo abaixo do abrirConexao entre as partes destacadas, conforme imagem abaixo:

```
        return conexao;
    }

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

}
```

Acreditamos que o código seja auto explicativo, mas se houver dúvida, conversamos na validação.

Codificando o método de busca das marcas

Com a conexão pronta para ser usada, voltemos agora à **MarcaRest**, mais especificamente ao método **buscar()**, para que possamos codificá-lo. Nele, adicione a linha destacada na imagem abaixo:

```
public Response buscar(){

    List<Marca> listaMarcas = new ArrayList<Marca>();

}
```

Devido a sua experiência anterior, dispensamos comentários sobre ela. Para corrigir os erros, importe as classes **List** e **ArrayList** :

```
package br.com.coldigogeladeiras.rest;

import java.util.ArrayList;
import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
```

Não se esqueça também da nossa classe modelo **Marca**, que apesar de ser nossa, não está no mesmo pacote, então precisa ser importada:

```
import javax.ws.rs.core.Response;

import br.com.coldigogeladeiras.modelo.Marca;

@Path("marca")
public class MarcaRest {
```

Agora vamos criar parte do código referente à conexão com o banco, adicionando as linhas destacadas a nosso código:

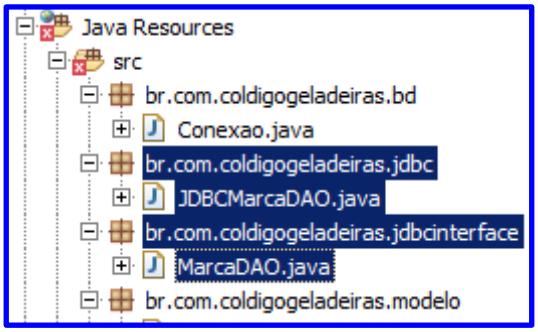
```
public Response buscar(){

    List<Marca> listaMarcas = new ArrayList<Marca>();

    Conexao conec = new Conexao();
    Connection conexao = conec.abrirConexao();

}
```

Com elas, criamos uma nova instância da classe **Conexao** e a armazenamos no objeto **conec,** para em seguida utilizarmos o método **abrirConexao()** e **guardarmos a conexão retornada por ele** em um objeto chamado **conexao**.
Para resolver os erros, importe os arquivos **Connection** e **Conexao** conforme destacado abaixo:

```
import java.sql.Connection;
import java.util.ArrayList;
import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

import br.com.coldigogeladeiras.bd.Conexao;
import br.com.coldigogeladeiras.modelo.Marca;
```

Mais uma vez, **Conexão** precisa ser **importado** por **não estar no mesmo pacote da classe que estamos codificando**...
Agora, vamos fazer uma pequena pausa para criarmos **a classe que usará essa conexão** para buscar as marcas no BD.

Realizando a busca das marcas no BD
Primeiramente, vamos criar duas novas classes, em pacotes novos e diferentes, conforme a imagem a seguir:



MarcaDAO será uma ***interface DAO*** para a classe modelo **Marca** que já criamos antes, e **JDBCMarcaDAO** é a classe que **realizará a interação com o BD** (agora para encontrar as marcas, e futuramente para muito mais).
Vamos primeiro editar o arquivo **MarcaDAO**. Por ser uma **interface**, seu papel é identificar o que pode ser feito pelas classes que a implementarem, **obrigando-as a fazê-lo**, assim seu código será bem simples agora:

MarcaRest.java | Marca.java | JDBCMarcaDAO.java | MarcaDAO.java

```
1  package br.com.coldigogeladeiras.jdbcinterface;
2
3  import java.util.List;
4
5  import br.com.coldigogeladeiras.modelo.Marca;
6
7  public interface MarcaDAO {
8
9      public List<Marca> buscar();
10
11 }
```

Indicamos através da assinatura do **método buscar()** que ele deverá ser **público** e retornar uma **List** com objetos do tipo **Marca.** Vale lembrar que, por enquanto, estamos *apenas buscando os dados das marcas* no banco de dados, portanto só devemos *forçar na interface a implementação do método para isso*.
Agora, vamos para a classe **JDBCMarcaDAO**. Nela, insira os códigos em destaque, nos locais em que se encontram na imagem a seguir:

MarcaRest.java | Marca.java | JDBCMarcaDAO.java | Marca

```
1  package br.com.coldigogeladeiras.jdbc;
2
3  import java.sql.Connection;
4
5  import br.com.coldigogeladeiras.jdbcinterface.MarcaDAO;
6
7  public class JDBCMarcaDAO implements MarcaDAO {
8
9      private Connection conexao;
10
11     public JDBCMarcaDAO(Connection conexao) {
12         this.conexao = conexao;
13     }
14
15 }
```

Explicando, por destaque:

- Importação das classes a serem utilizadas;
- Indicação de que essa classe implementa a interface MarcaDAO;
- Criação de um atributo **conexao**, que guardará a conexão aberta, e um **método construtor**, que deve receber uma conexão como parâmetro para internamente colocá-la no atributo **conexao**.
-

Nesse momento, você consegue imaginar onde devemos chamar esse método construtor?

Se não, tudo bem, mas se pensou em **MarcaRest**, **acertou**!

Vá então até essa classe, e entre os códigos destacados na imagem, insira as 2 novas linhas relacionadas à nossa mais nova classe:

```
        Conexao conec = new Conexao();
        Connection conexao = conec.abrirConexao();
        JDBCMarcaDAO jdbcMarca = new JDBCMarcaDAO(conexao);
        listaMarcas = jdbcMarca.buscar();

    }
```

Com elas, criaremos um **objeto jdbcMarca**, que receberá uma **instância de JDBCMarcaDAO** *criada com uso do construtor* que fizemos, para que já *possua a conexão ativa*, e abaixo chamaremos através desse objeto o método **buscar()**, que nos *retornará uma lista com as marcas do BD*, que guardaremos em **listaMarcas**.

Como vimos pela imagem, o Eclipse aponta **erros** em JDBCMarcaDAO. Esperamos a essa altura que **já sabe como resolvê-los,** certo? Então, **resolva-os e siga em frente**...
Nosso próximo passo é implementar o método **buscar()** da classe **JDBCMarcaDAO**, que como já dito anteriormente, é a classe *responsável pelas ações que necessitam de conexão com o banco para a classe Marca*. Então vá até ela e, para começarmos essa implementação, *abaixo do método construtor*, crie o código da imagem abaixo:

```
public List<Marca> buscar() {

    //Criação da instrução SQL para busca de todas as marcas
    String comando = "SELECT * FROM marcas";

    //Criação de uma lista para armazenar cada marca encontrada
    List<Marca> listMarcas = new ArrayList<Marca>();

    //Criação do objeto marca com valor null (ou seja, sem instanciá-lo)
    Marca marca = null;

}
```

As 3 instruções que fizemos acima estão comentadas, então vamos em frente. Para resolvermos o erro indicado no nome do método, vamos efetuar **o retorno da lista de marcas**:

```
public List<Marca> buscar() {

    //Criação da instrução SQL para busca de todas as marcas
    String comando = "SELECT * FROM marcas";

    //Criação de uma lista para armazenar cada marca encontrada
    List<Marca> listMarcas = new ArrayList<Marca>();

    //Criação do objeto marca com valor null (ou seja, sem instanciá-lo)
    Marca marca = null;

    //Retorna para quem chamou o método a lista criada
    return listMarcas;

}
```

Agora, vamos fazer as coisas de uma maneira mais passo a passo, então, para facilitar a explicação e termos menos erros durante a codificação, vamos criar primeiro um bloco **try-catch**, entre os códigos destacados na imagem:

```
//Criação do objeto marca com valor null (ou seja, sem instanciá-lo)
Marca marca = null;

//Abertura do try-catch
try {

//Caso alguma Exception seja gerada no try, recebe-a no objeto "ex"
} catch (Exception ex) {
    //Exibe a exceção na console
    ex.printStackTrace();
}

//Retorna para quem chamou o método a lista criada
return listMarcas;
```

O nosso grande foco em um try-catch é, obviamente, o **try**, já que é nele que colocaremos todo o código que deve ser executado se não forem gerados erros, assim vamos começar a codificá-lo:

```
//Abertura do try-catch
try {

    //Uso da conexão do banco para prepará-lo para uma instrução SQL
    Statement stmt = conexao.createStatement();

    //Execução da instrução criada previamente
    //e armazenamento do resultado no objeto rs
    ResultSet rs = stmt.executeQuery(comando);

//Caso alguma Exception seja gerada no try, recebe-a no objeto "ex"
} catch (Exception ex) {
```

Você pesquisou sobre as classes Statement e ResultSet, mas vamos especificar como as coisas acontecem por aqui:

- Na primeira linha, usamos o atributo **conexao** para criar um **Statement stmt**, pelo qual podemos executar comandos no BD;
- Na segunda linha, executamos o comando SELECT criado anteriormente, através do **executeQuery** e guardamos o resultado em um objeto **rs** do tipo **ResultSet**.

Continuando, acrescente o novo código entre as partes em destaque:

```
ResultSet rs = stmt.executeQuery(comando);

//Enquanto houver uma próxima linha no resultado
while (rs.next()) {

    //Criação de instância da classe Marca
    marca = new Marca();

    //Recebimento dos 2 dados retornados do BD para cada linha encontrada
    int id = rs.getInt("id");
    String nome = rs.getString("nome");

    //Setando no objeto marca os valores encontrados
    marca.setId(id);
    marca.setNome(nome);

    //Adição da instância contida no objeto Marca na lista de marcas
    listMarcas.add(marca);
}

//Caso alguma Exception seja gerada no try, recebe-a no objeto "ex"
} catch (Exception ex) {
```

Sobre o código em si, pelas linhas estarem comentadas, acreditamos não haver muitas dúvidas, então não traremos mais explicações.

ATENÇÃO: veja que antes do try criamos o objeto marca como null e apenas dentro do while atribuímos a ele uma instância da classe Marca. Reflita sobre o porquê e nos explique sua conclusão durante a validação.

Isso é tudo que precisávamos do método **buscar() de JDBCMarcaDAO**. Assim, pelo que já fizemos, o método **buscar() da classe MarcaRest** já criou uma conexão ao banco e fez a pesquisa das marcas cadastradas. Falta apenas **encerrar a conexão**.
Assim, volte ao método **buscar() da classe MarcaRest** e adicione *ao final dele*, depois da chamada de jdbcMarca.buscar(), a linha destacada na imagem abaixo:

```
    listaMarcas = jdbcMarca.buscar();
    conec.fecharConexao();

}
```

Com isso, **encerramos os processos relativos ao BD**. Precisamos então devolver a informação encontrada ao cliente...

As etapas ainda não acabaram, valide com o Orientador ate esse ponto e vamos seguir para os proximos passos