

## Curso: Análise e Desenvolvimento de Sistemas – ADS

Ano: 2023/1

Orientação Técnica (OT) – 16

### Cadastro REST - Exibir Marcas

---

#### Introdução

Como visto na OT anterior, para cadastrarmos um produto, devemos identificar sua marca em um dos campos do formulário. Mas como poderíamos fazer isso?

Poderíamos até fazer o usuário inserir manualmente o nome de uma marca, mas com isso não conseguiríamos garantir que a marca preenchida é realmente uma das registradas no banco, gerando problemas de [integridade referencial](#). Além disso, no nosso BD, a tabela “produtos” é relacionada com a tabela “marcas”, certo? Por quê? Justamente para que identifiquemos a marca que é referente a cada produto.

Assim, para fazermos o registro de produtos com sucesso, devemos fazer com que o usuário escolha uma **marca que já existe no BD**. Isso pode ser feito de várias maneiras. Aqui, buscamos fazer da maneira mais simples possível, porém como é a nossa primeira OT onde realmente programaremos em Java Web, vários conceitos novos entrarão em pauta. Porém, também nela faremos configurações iniciais e downloads (de plugins, bibliotecas e outros), entre outras atividades, que em nenhuma das outras OTs precisaremos repetir, assim a enriquecemos com muitas imagens (basicamente uns 75% do arquivo são imagens), então essa OT deve ser a **com mais páginas da trilha**. Assim, **mantenha o foco**. A complexidade dessa OT é, na realidade, bem menor do que parece.

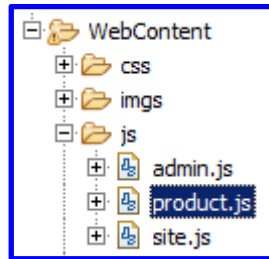
Uma dica muito importante que já demos anteriormente, mas voltamos a reforçar, é: **desabilite o cache no navegador!** Se não o fizer, pode ser que suas alterações não sejam carregadas devido ao cache e você fique com a impressão de que as coisas não estão funcionando por culpa do código, mesmo que ele esteja certo. Por sinal, faça isso em **TODAS as OTs!** Fica aqui o aviso, isso não será repetido!

Por fim, é relevante que saiba e note que **a sequência de trabalho utilizada na OT é a mais próxima possível da ordem lógica de execução das ações pelo computador**, então você terá em certos momentos que “parar” para fazer algum download, ou voltar a algum arquivo que já modificamos no início da OT para adicionar algum código. Tudo isso para **facilitar sua compreensão e mostrar aproximadamente como é o fluxo das ações**. Assim, mãos à obra!

---

#### Criando o arquivo JS do Registro de Produto

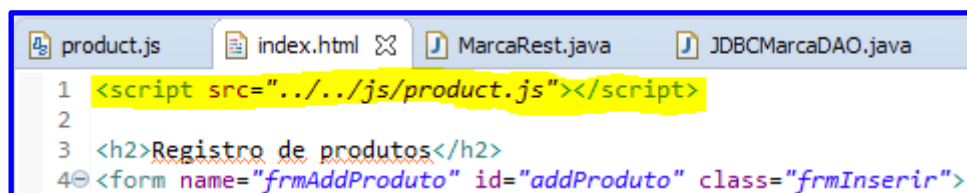
Essa parte é fácil. Crie o arquivo destacado na imagem no respectivo local e pronto...



Esse arquivo será responsável no frontend pelas funções que realmente **farão as coisas acontecerem**, desde a validação de dados dos formulários até inserção, exclusão, busca, alteração... Esses códigos serão inseridos conforme sua necessidade, assim, basicamente em cada OT adicionaremos algo a ele.

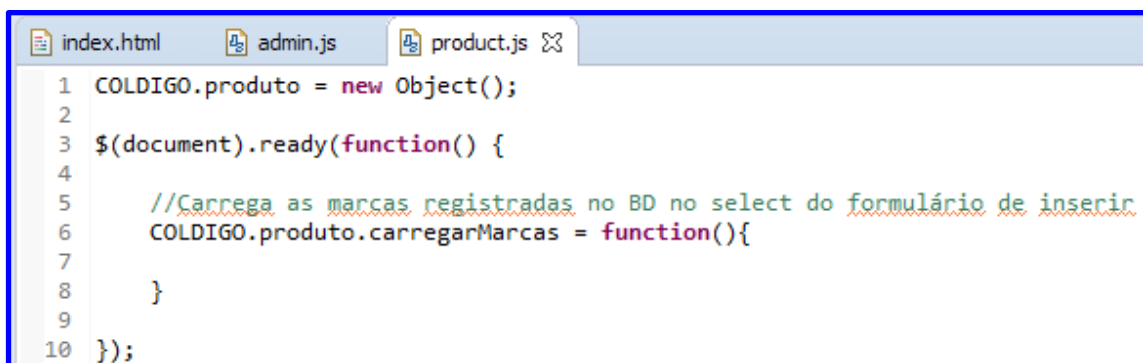
## Criando a função JS de carregamento de marcas

Primeiro, no arquivo **product/index.html**, adicione a linha em destaque:



Assim, quando clicarmos na opção do menu “**Registros - Produtos**” e essa **página for carregada**, o **arquivo js também será**.

Agora abra o arquivo recém criado (**product.js**) e nele insira o código da imagem abaixo:



Vamos explicar parte a parte o que foi feito:

- Primeiramente, criamos um objeto chamado **COLDIGO.produto**. Lembra-se do objeto COLDIGO que criamos anteriormente? Pois é, esse objeto é “filho” dele. Nele, teremos todas as funções e valores relativos aos processos do registro de produto.
- Depois, indicamos que assim que o documento for carregado, deve ser criada uma função chamada **carregarMarcas**, dentro do objeto **COLDIGO.produto**.

Essa função será onde, no frontend, **chamaremos o servidor**, em busca do **REST** responsável por nos devolver as marcas registradas no BD. Para acessarmos o lado servidor de nossa aplicação, utilizaremos a função **Ajax do jQuery** (lembre-se: a index.html da área administrativa faz o carregamento do arquivo **jquery.js**, assim podemos utilizá-lo em todas as páginas carregadas dentro dela, como é o caso do arquivo **product/index.html**, que carrega por sua vez o **product.js**).

Continuando, insira o código Ajax destacado dentro da função recém criada:

```
//Carrega as marcas registradas no BD no select do formulário de inserir
COLDIGO.produto.carregarMarcas = function(){

    $.ajax({
        type: "GET",
        url: "/ProjetoTrilhaWeb/rest/marca/buscar",
        success: function () {
        },
        error: function () {
        }
    });
}
```

Nele, fazemos 4 configurações básicas:

- Método de acesso ao servidor: GET
- URL de destino: “/ProjetoTrilhaWeb/rest/marca/buscar”
- Duas funções (por enquanto vazias), sendo uma em caso de sucesso na comunicação com o servidor e outra em caso de erro.

**ATENÇÃO:** sabe dizer o motivo de termos usado o método GET? Dica: tem a ver com RESTful...

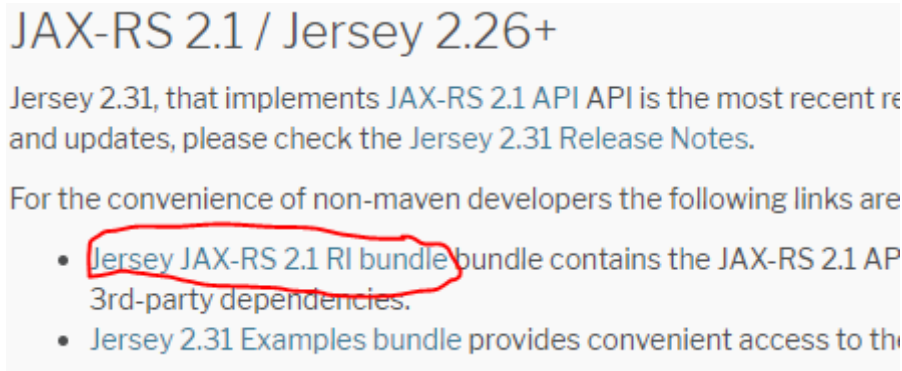
Com essa configuração básica, já teríamos uma tentativa de conexão com o servidor, mas ele ainda não está pronto para recebê-la... Precisamos do Jersey!

---

## Aplicando a API REST Jersey

Acesse <https://eclipse-ee4j.github.io/jersey/download.html> para fazer o download da API Jersey.

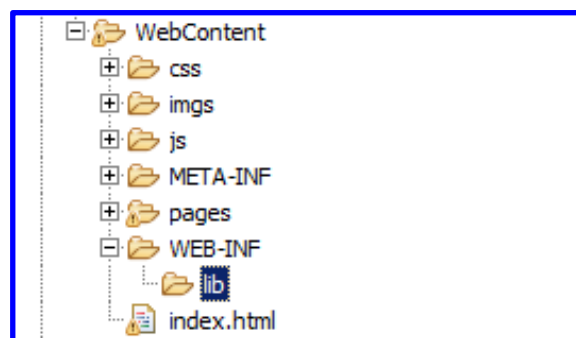
Observe a imagem:



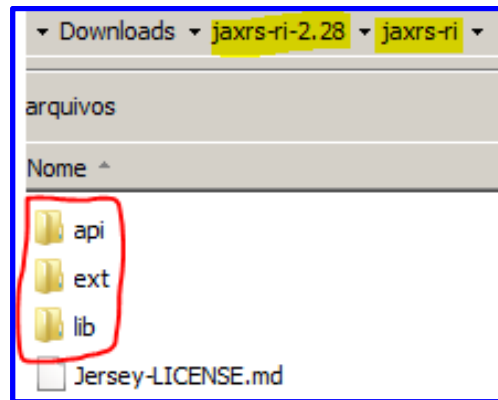
Clique no link **equivalente ao da imagem, independente da versão**. Como sempre, orientamos o download da versão mais atual, assim, se a versão que você ver for diferente da imagem, não busque a versão igual à nossa, faça o download da mais atual, mas do pacote equivalente ao que baixamos.

**Observação (05/2021): para seu funcionamento na versão 9 do Tomcat, é necessário o uso da versão 2.x.x (mais atual possível). A versão 3.x.x. é para o Tomcat 10 em diante, e alterações nos códigos seriam necessárias para seu correto funcionamento.**

Com o download realizado, descompacte o arquivo em algum lugar. Agora volte ao seu projeto e encontre a pasta “**WebContent/WEB-INF/lib**”, local esse destacado na imagem a seguir:



Essa pasta é onde colocaremos **todas as bibliotecas Java** externas. Agora abra as pastas descompactadas do Jersey e encontre as destacadas na imagem:



Todos os arquivos contidos nessas pastas devem ser copiados para a pasta “lib” que chamamos atenção acima (apenas os arquivos, não a pasta inteira, então abra cada uma dessas pastas, copie os arquivos e cole no local indicado!).

**ATENÇÃO:** seu download não tem as mesmas pastas? Verifique se realmente fez o download equivalente ao que pedimos, nos mesmo link. Se tiver certeza que sim e mesmo assim estiver diferente, comunique imediatamente um orientador!

Agora, para indicarmos como o Jersey deve trabalhar, abra o arquivo **web.xml** dentro da pasta **WEB-INF** (caso ele não exista, é só criá-lo).



Esse arquivo é responsável, como viu em Servlet, por **identificar os recursos existentes no servidor** (o usamos para fazer o servlet-mapping, lembra?). Ele deverá ter um código conforme a imagem abaixo (se foi você quem o criou, ele estará vazio, então replique nele o código abaixo):

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://java.sun.com/xml/ns/javaee"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5   id="WebApp_ID" version="2.5">
6   <display-name>Coldigo Geladeiras</display-name>
7   <welcome-file-list>
8     <welcome-file>index.html</welcome-file>
9     <welcome-file>index.htm</welcome-file>
10    <welcome-file>index.jsp</welcome-file>
11    <welcome-file>default.html</welcome-file>
12    <welcome-file>default.htm</welcome-file>
13    <welcome-file>default.jsp</welcome-file>
14  </welcome-file-list>
15 </web-app>
```

Agora, entre os destaques em amarelo, crie o código da imagem a seguir:

```
</welcome-file-list>

<servlet>
  <servlet-name>Jersey REST Service</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
  <init-param>
    <param-name>jersey.config.server.provider.packages</param-name>
    <param-value>br.com.coldigogeladeiras.rest</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Jersey REST Service</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>

</web-app>
```

Reparou algo familiar? Pois é! **Identificamos uma Servlet** do Jersey, e a **mapeamos** para tratar **tudo que chegar no servidor** com o **endereço** “/ProjetoTrilhaWeb/rest/algumacoisa” com os arquivos contidos em um pacote chamado “**br.com.coldigogeladeiras.rest**” (que criaremos em breve para armazenar todos os nossos arquivos REST).

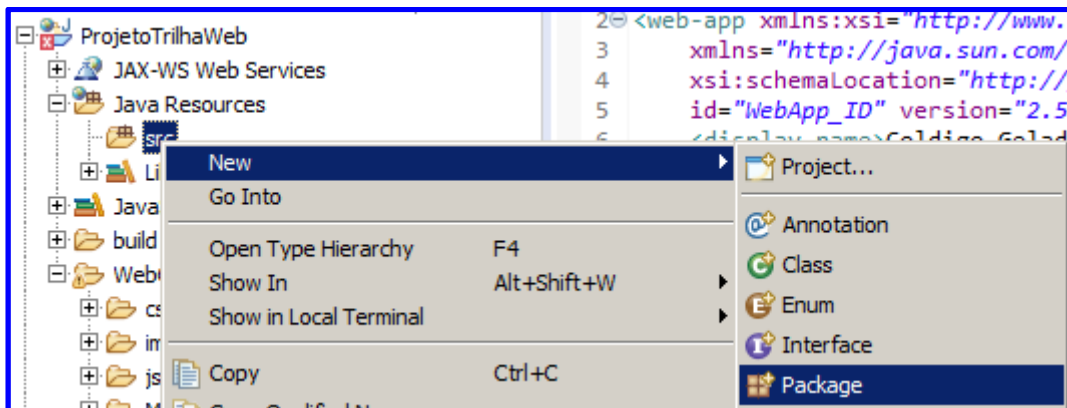
Explicando um pouco mais, o **Jersey** nada mais é do que um **Servlet Container**, recebendo requisições e redirecionando para os arquivos REST que criaremos seguindo as especificações do Jersey. O **<servlet-mapping>** em si é igual ao que fizemos em Servlet, mas na declaração do **<servlet>** temos algumas novidades:

- **<init-param>**: define configurações de um parâmetro inicial;
  - **<param-name>**: indica a configuração a ser modificada, nesse caso o *pacote* (*package*) que conterá nossas classes REST;
  - **<param-value>**: indica o valor a ser atribuído a essa configuração, ou seja, o *local do pacote* que conterá as REST;
  - **<load-on-startup>**: o valor 1 indica que *essa servlet deve ser carregada assim que o servidor for iniciado*. Isso porque a ServletContainer do Jersey deve estar pronta para receber as requisições às nossas REST desde o momento em que o servidor for ligado.
-

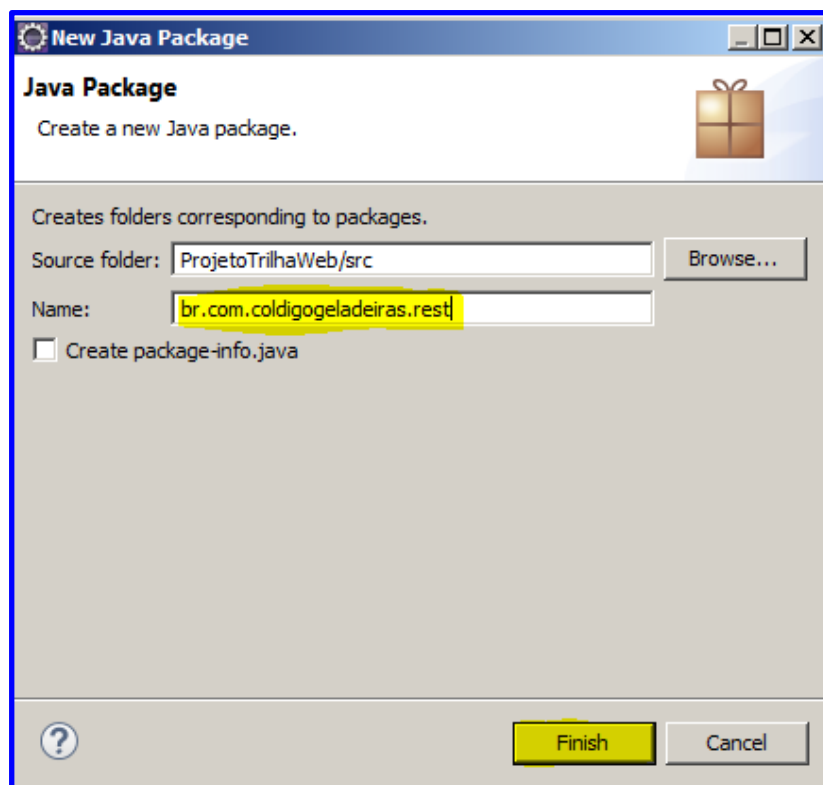


## Criando o primeiro arquivo REST

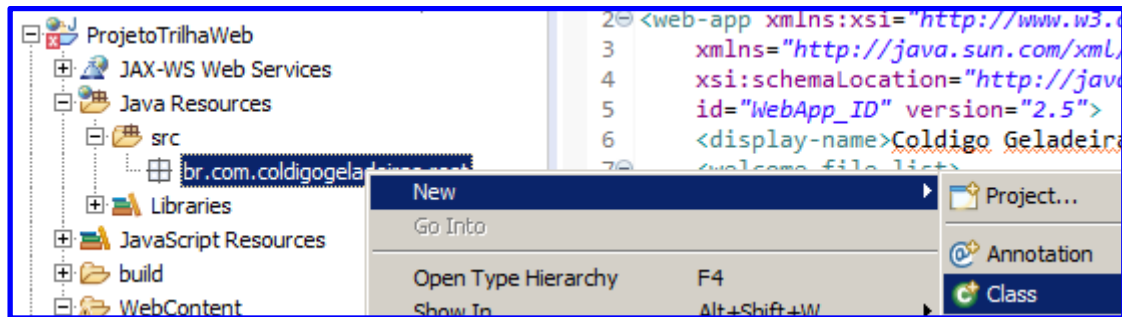
Primeiramente, vamos criar o pacote que indicamos na configuração do Jersey para conter nossas REST. Expanda a opção **Java Resources** e clique em **src** com o botão direito, escolhendo depois **New - Package**:



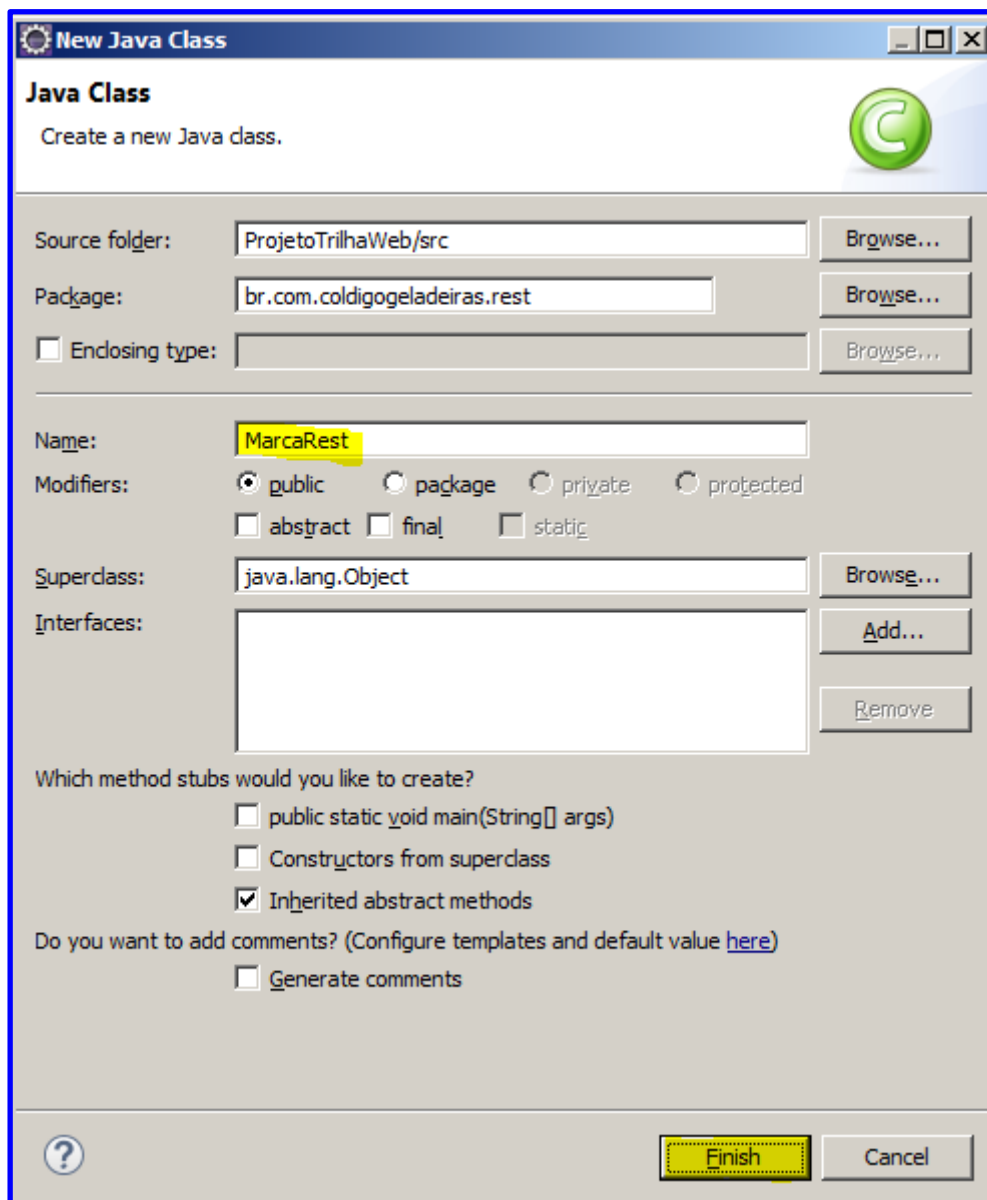
Agora, nomeie-o com o mesmo nome colocado no web.xml:



Assim que finalizar, o pacote será criado, vazio. Agora podemos criar nossa primeira classe REST, clicando no pacote novo com o botão direito, depois em **New - Class**:

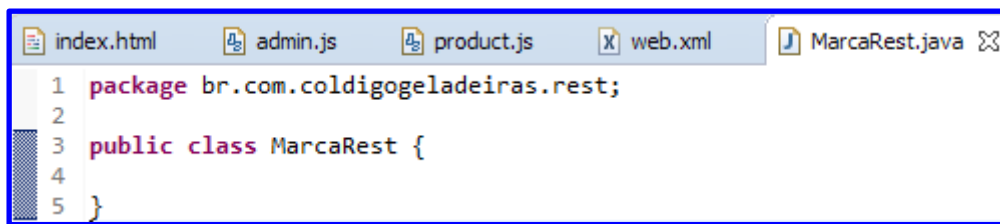


Nomeie a classe nova como **MarcaRest** e finalize:



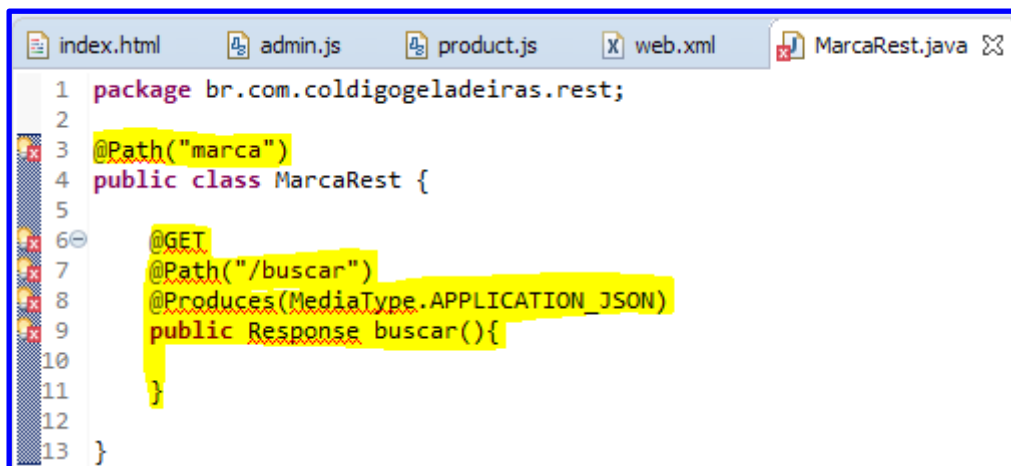


A classe recém criada ficará assim:



```
1 package br.com.coldigogeladeiras.rest;
2
3 public class MarcaRest {
4
5 }
```

Feito isso, podemos nele agora programar **nossa primeira ação REST**: a de **busca de marcas**! Insira o código destacado em amarelo nos locais mostrados na imagem:



```
1 package br.com.coldigogeladeiras.rest;
2
3 @Path("marca")
4 public class MarcaRest {
5
6     @GET
7     @Path("/buscar")
8     @Produces(MediaType.APPLICATION_JSON)
9     public Response buscar(){
10
11     }
12
13 }
```

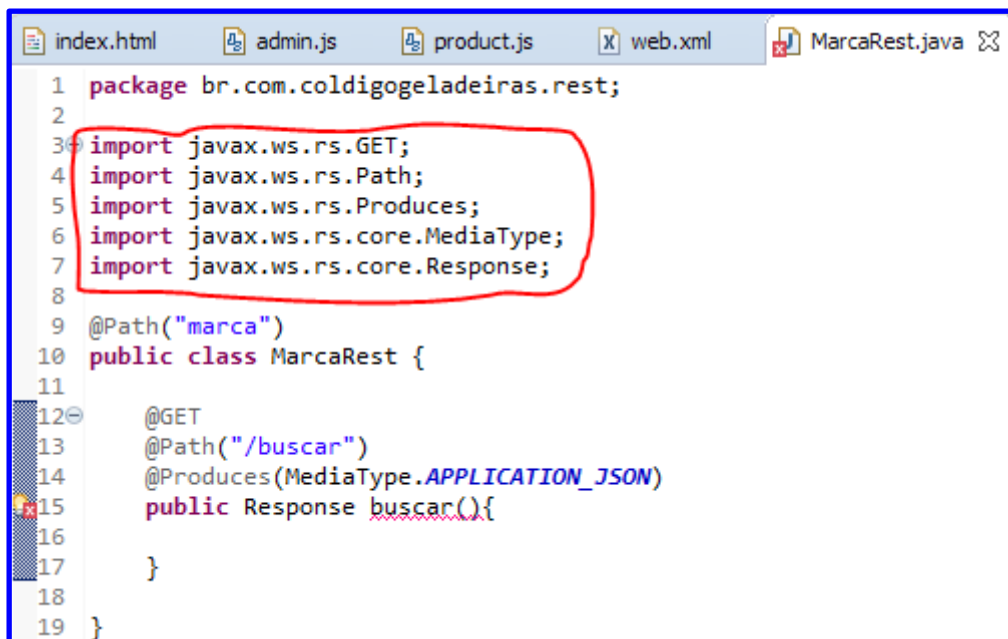
Primeiro, uma explicação sobre o uso do @: com ele, podemos criar em nossas classes o que chamamos de **anotações**, inserindo assim **metadados** (ou seja, dados sobre os dados contidos nela) do nosso código. Com o **Jersey**, usaremos várias delas para informar **como cada método de nossas classes REST devem se comportar**.

Agora vamos explicar o código com calma...

- Primeiro, repare na linha 3: fazemos uma **anotação** (@) referente ao **caminho (Path)** web dessa classe. Ou seja, toda vez que alguém tentar acessar o servidor pelo endereço “/ProjetoTrilhaWeb/rest/marca”, é essa classe que irá responder!
- Na linha 6, fazemos uma **anotação** pela qual indicamos que o método a seguir será acessível apenas pelo método **GET**.
- Na 7, fazemos uma **anotação** pela qual indicamos o **caminho (Path)** web desse método. Ou seja, toda vez que alguém tentar acessar “/ProjetoTrilhaWeb/rest/marca/buscar”, é esse método que irá responder!
- Já a linha 8 indica com uma **anotação** que esse método deve produzir (**Produces**) um resultado, ou seja, deve devolver algum valor ao lado cliente, no formato “JSON”.
- Na linha 9, criamos um método **público** chamado **buscar()** que terá a implementação necessária para **retornar as marcas cadastradas no formato JSON** ao lado cliente através de um objeto **Response** (você já o conhece de Servlet...).

Mas agora deixamos o código com **vários erros**, não é? Isso porque essas anotações todas não são do Java em si, mas do Jersey! Lembra dos arquivos que você colocou na pasta **WEB-INF/lib**? Então, eles contêm o que precisamos para essas anotações (e várias outras coisas) funcionarem. Mas precisamos importar os conteúdos necessários nessa classe para

esse reconhecimento acontecer. Vamos arrumar esses erros com o código da imagem a seguir, então digite-o no local indicado:



```
1 package br.com.coldigogeladeiras.rest;
2
3 import javax.ws.rs.GET;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.Produces;
6 import javax.ws.rs.core.MediaType;
7 import javax.ws.rs.core.Response;
8
9 @Path("marca")
10 public class MarcaRest {
11
12     @GET
13     @Path("/buscar")
14     @Produces(MediaType.APPLICATION_JSON)
15     public Response buscar(){
16
17     }
18
19 }
```

Veja que apenas o erro do método buscar() persiste. Não se preocupe com isso agora.

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