

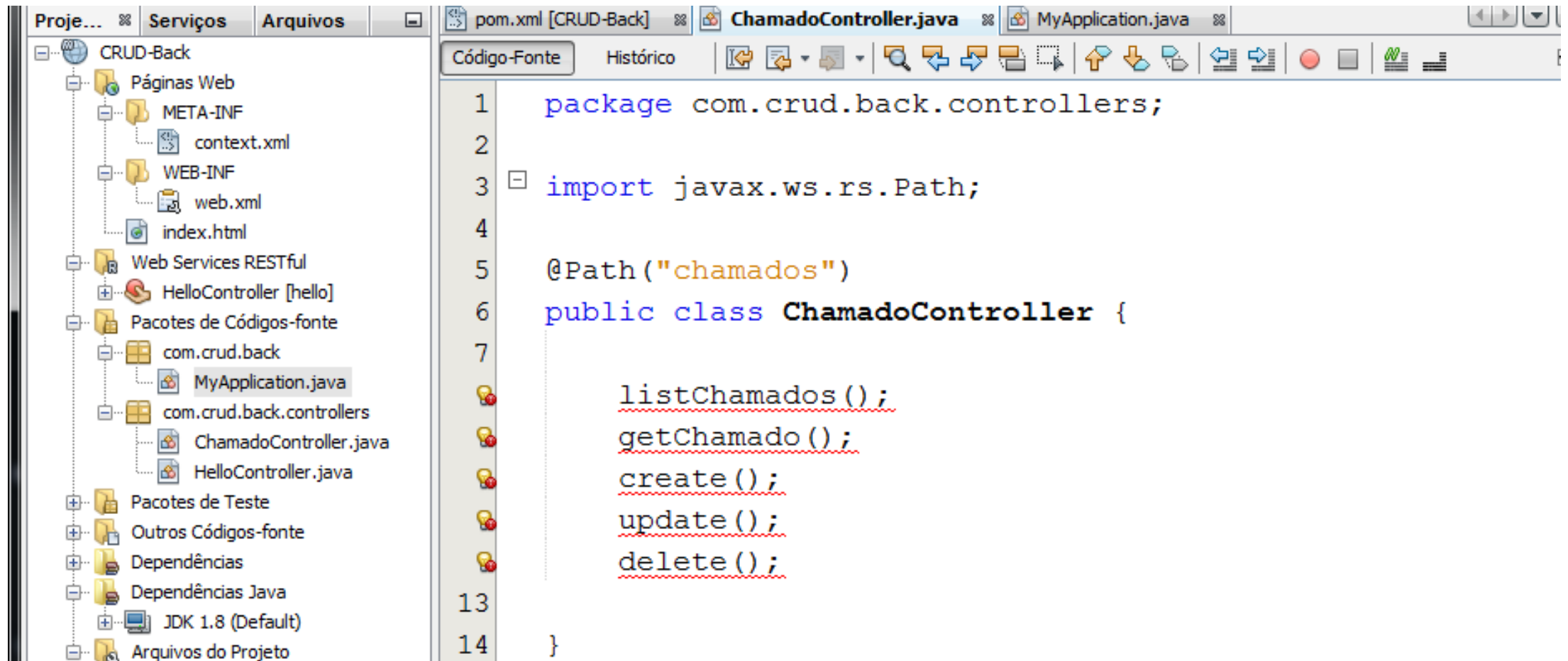
Introdução ao AngularJS

Professor Anderson Henrique



Criando nosso projeto CRUD

Vamos criar uma classe java no pacote **com.crud.back.controllers** chamada: **ChamadoController**



Nesta classe vamos inserir a anotação `@Path` que indica o caminho para o controller

Vamos implementar os métodos que serão utilizados no nosso controller para:

listChamados -> retornar todos os chamados

getChamado -> retornar um chamado específico

Create -> criar um novo chamado

Update -> atualizar um chamado existente

Delete -> excluir um chamado específico

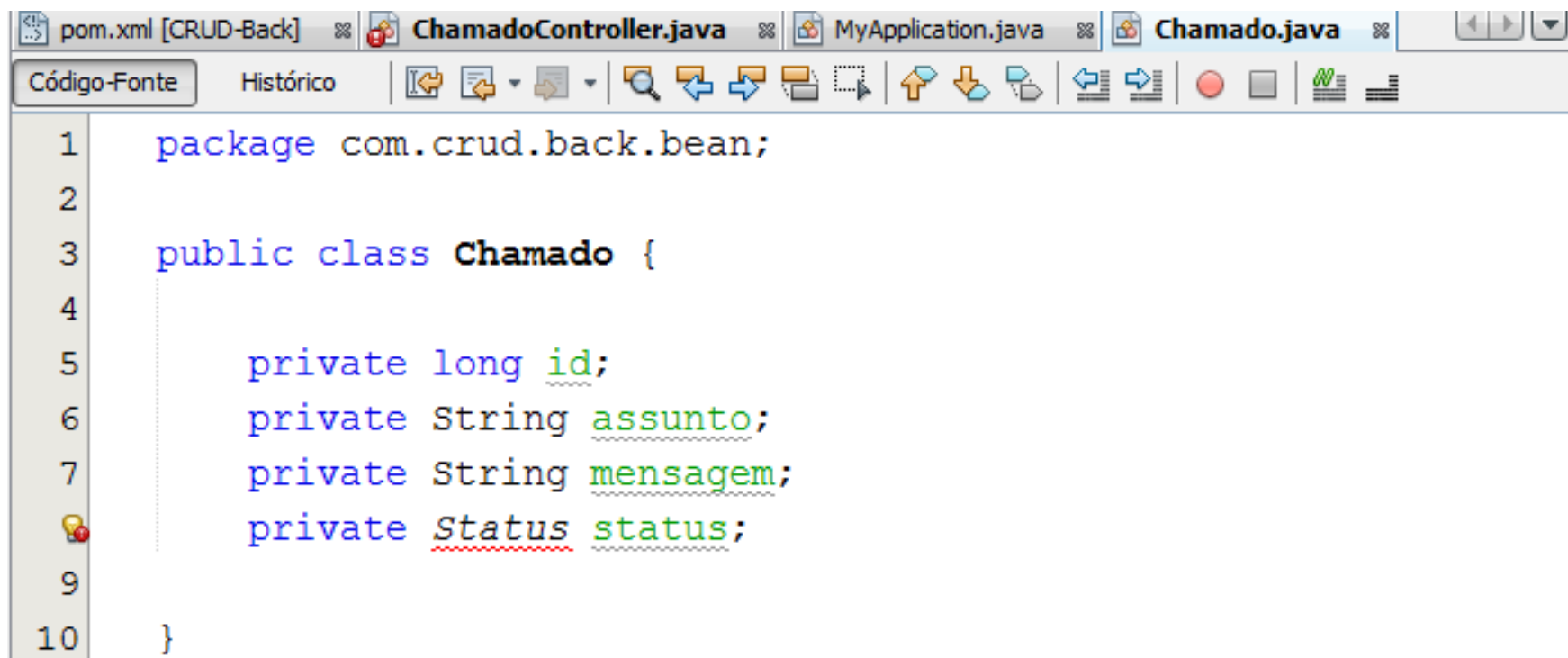
```
5  @Path("chamados")
6  public class ChamadoController {
7
8      listChamados();
9      getChamado();
10     create();
11     update();
12     delete();
13
14 }
```

Vamos implementar o método `listChamados`, precisamos fazer a anotação do método **@GET**, **@Produces** que produzirá dados do tipo json e **@Path** é o caminho do método na URL

```
12      @GET
13      @Produces(MediaType.APPLICATION_JSON)
14      @Path("/")
15      public List<Chamado> listChamados() {
16
17      }
```

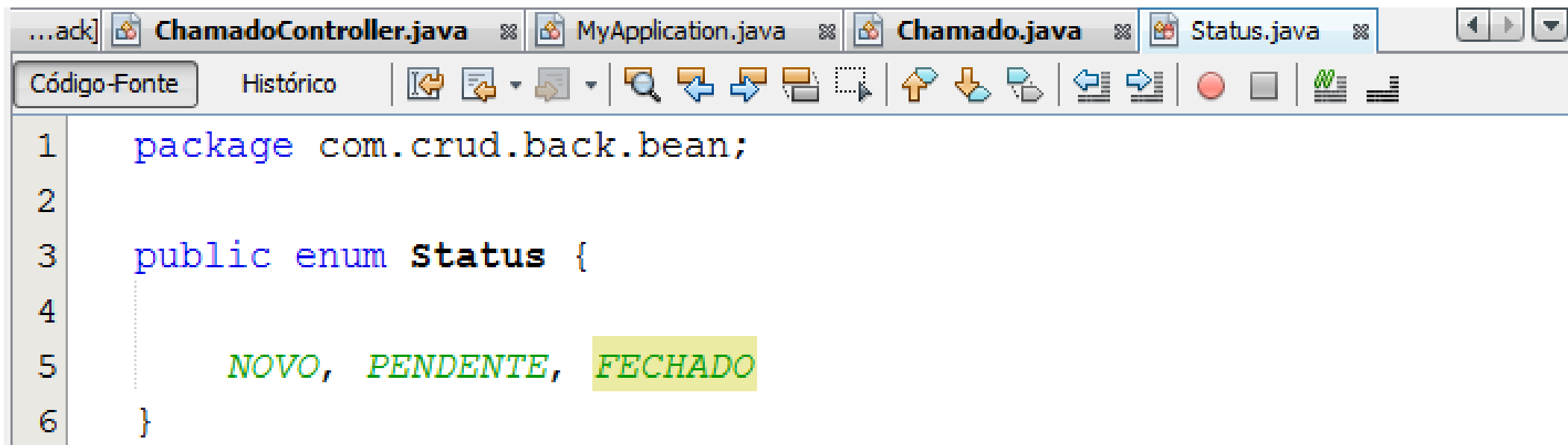
Como vamos retornar objetos do tipo **Chamado**, precisamos criar esta classe que representa os **dados** na tabela do nosso **Banco de Dados**, em java esses dados são chamados de **Bean**

Vamos criar um novo pacote java chamado **com.crud.back.bean** e neste pacote vamos criar a classe java Chamado, que terá a princípio esses 4 atributos

A screenshot of an IDE window showing the code for Chamado.java. The window has a title bar with tabs for pom.xml [CRUD-Back], ChamadoController.java, MyApplication.java, and Chamado.java. Below the title bar is a toolbar with icons for file operations and editing. The code is displayed in a text area with line numbers on the left. The code defines a package and a class with four private attributes.

```
1 package com.crud.back.bean;
2
3 public class Chamado {
4
5     private long id;
6     private String assunto;
7     private String mensagem;
8     private Status status;
9
10 }
```

O atributo **status** ainda não existe, ele será um **enum** java chamado **Status**, também vamos criá-lo



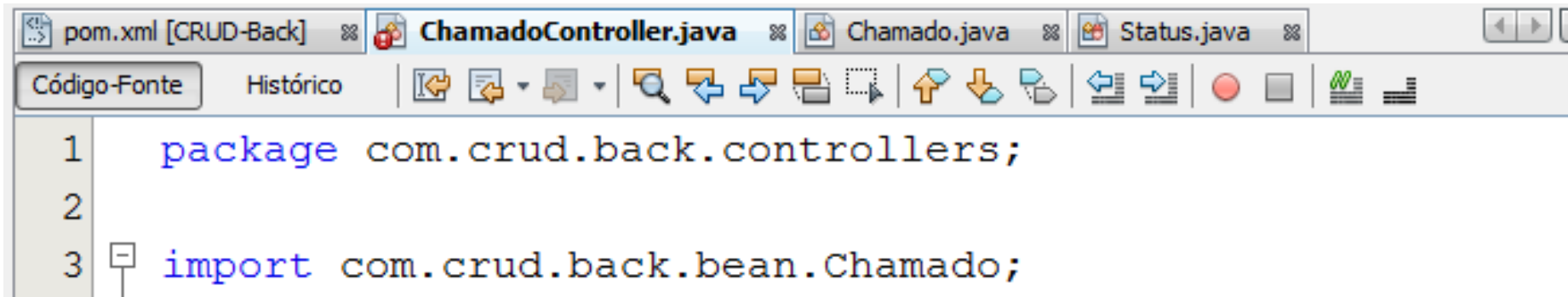
The screenshot shows an IDE window with four tabs: ChamadoController.java, MyApplication.java, Chamado.java, and Status.java. The Status.java tab is active, displaying the following code:

```
1 package com.crud.back.bean;
2
3 public enum Status {
4
5     NOVO, PENDENTE, FECHADO
6 }
```

The code is written in a monospaced font with syntax highlighting. The package name is in blue, the enum keyword is in blue, and the enum name 'Status' is in bold black. The enum values 'NOVO', 'PENDENTE', and 'FECHADO' are in green. The IDE interface includes a toolbar with various icons for file operations, search, and code navigation.

Vamos gerar os getters e setters, gerar o equals() e hashCode() apenas do atributo id e gerar o método toString de todos os atributos na classe Chamado, salve o arquivo

Na classe **ChamadoController**, importe o **bean** que foi criado: a classe java Chamado

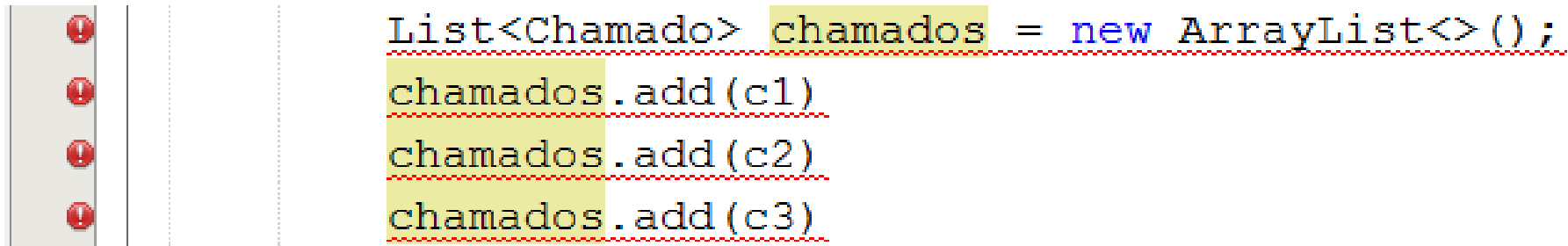


```
1 package com.crud.back.controllers;
2
3 import com.crud.back.bean.Chamado;
```


Como ainda não temos o nosso banco de dados, vamos gerar alguns chamados manualmente dentro do método listChamados

```
19 Chamado c1 = new Chamado();
20 c1.setAssunto("Assunto1");
21 c1.setMensagem("Mensagem1");
22 c1.setStatus(Status.NOVO);
23
24 Chamado c2 = new Chamado();
25 c2.setAssunto("Assunto2");
26 c2.setMensagem("Mensagem2");
27 c2.setStatus(Status.NOVO);
28
29 Chamado c3 = new Chamado();
30 c3.setAssunto("Assunto3");
31 c3.setMensagem("Mensagem3");
32 c3.setStatus(Status.FECHADO);
```

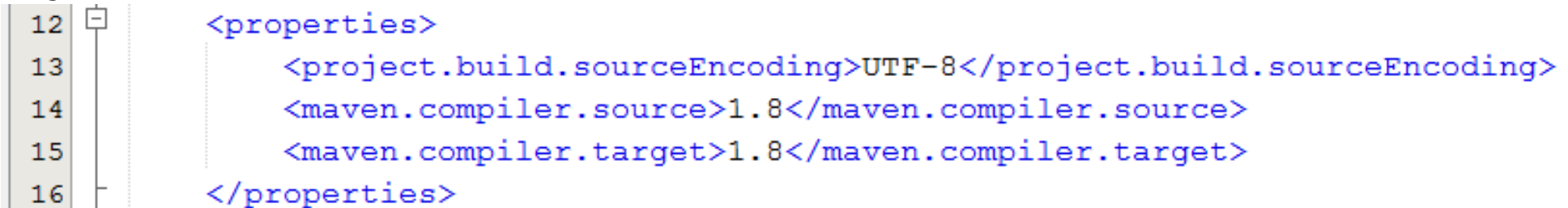
A nossa aplicação não está reconhecendo a versão mais nova do **javaEE**, ele não está aceitando o tipo de operação diamante, para isto vamos alterar o arquivo **pom.xml**

A screenshot of a code editor showing four lines of Java code. To the left of the code, there are four red circular icons, each containing a white exclamation mark, indicating compilation errors. The code is:

```
List<Chamado> chamados = new ArrayList<>();  
chamados.add(c1);  
chamados.add(c2);  
chamados.add(c3);
```

```
List<Chamado> chamados = new ArrayList<>();  
chamados.add(c1);  
chamados.add(c2);  
chamados.add(c3);
```

Dentro da tag <properties> vamos informar para o Maven qual a versão do javaEE que vamos utilizar no projeto, em tempo de compilação como a aplicação final

A screenshot of a code editor showing the <properties> section of a pom.xml file. Line numbers 12 through 16 are visible on the left. The code is:

```
<properties>  
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
  <maven.compiler.source>1.8</maven.compiler.source>  
  <maven.compiler.target>1.8</maven.compiler.target>  
</properties>
```

```
<properties>  
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
  <maven.compiler.source>1.8</maven.compiler.source>  
  <maven.compiler.target>1.8</maven.compiler.target>  
</properties>
```

Salve o arquivo e não teremos mais o erro no ArrayList<Chamado>

```
34 List<Chamado> chamados = new ArrayList<>();  
35 chamados.add(c1);  
36 chamados.add(c2);  
37 chamados.add(c3);  
38  
39 return chamados;  
40 }
```

Por fim, retornamos a lista de chamados

Vamos agora implementar o método `getChamado`, para isto vamos fazer as anotações: **@GET** para definir o tipo de método utilizando, **@Produces** que produzirá um documento tipo json e **@Path** passamos o caminho do parâmetro `id` que será informado.

No parâmetro do método fazemos a anotação **@PathParam** para vincular ao **@Path** do `id` do método

```
43 @GET
44 @Produces(MediaType.APPLICATION_JSON)
45 @Path("/{id}/")
46 public Chamado getChamado(@PathParam("id") long id) {
47 }

```

Como ainda não temos um banco de dados, vamos criar um objeto do tipo chamado dentro do método e retorná-lo

```
47 Chamado c1 = new Chamado();  
48 c1.setId(id);  
49 c1.setAssunto("Assunto" + id);  
50 c1.setMensagem("Mensagem" + id);  
51 c1.setStatus(Status.NOVO);  
52  
53 return c1;
```

Vamos criar o método create, ao invés de produzir informação ele consumirá a informação

Vamos inserir as anotações nesse método: **@POST** é o método utilizada para criar um novo chamado, **@Consumes**, ele consumirá dados do tipo json, **@Path** é o caminho da URL, esse método retorna objeto do tipo **Response** e como ainda não temos o banco de dados vamos dar print nas informações que chegam através do json no console

```
59      @POST
60      @Consumes(MediaType.APPLICATION_JSON)
61      @Path("/")
62      public Response create(Chamado chamado) {
63          System.out.println(chamado.toString());
64          return Response.status(Response.Status.OK).build();
65      }
```

O status **OK** no protocolo HTTP é **200**

Vamos implementar o método `update` com as seguintes anotações: **@PUT** é o método utilizado nessa requisição, **@Consumes**, **@Path** serão idênticos ao do método **create**, a assinatura do método será parecida apenas mudando o **nome** do método

```
68 @PUT
69 @Consumes (MediaType.APPLICATION_JSON)
70 @Path ("/")
71 public Response update (Chamado chamado) {
72     System.out.println(chamado.toString());
73     return Response.status(Response.Status.OK).build();
74 }
```

Por último, vamos implementar o método delete, vamos fazer as anotações: **@DELETE** será utilizando no método e o **@Path** que define o caminho do método com a passagem do parâmetro id, também temos que vincular com o parâmetro do método através da anotação **@PathParam**

```
77      @DELETE
78      @Path("/{id}/")
79      public Response delete(@PathParam("id") long id) {
80          System.out.println("Deletando ID: " + id);
81          return Response.status(Response.Status.OK).build();
82      }
```


Estes são os **import(s)** utilizados no controller **ChamadoController**

```
3 import com.crud.back.bean.Chamado;  
4 import com.crud.back.bean.Status;  
5 import java.util.ArrayList;  
6 import java.util.List;  
7 import javax.ws.rs.Consumes;  
8 import javax.ws.rs.DELETE;  
9 import javax.ws.rs.GET;  
10 import javax.ws.rs.POST;  
11 import javax.ws.rs.PUT;  
12 import javax.ws.rs.Path;  
13 import javax.ws.rs.PathParam;  
14 import javax.ws.rs.Produces;  
15 import javax.ws.rs.core.MediaType;  
16 import javax.ws.rs.core.Response;
```

Vamos Limpar e Construir o nosso projeto para saber se vai compilar com sucesso

**Na próxima aula vamos testar esse controller utilizando a ferramenta
POSTMAN**

Dúvidas?

Professor Anderson Henrique



Para a próxima aula

01 – Testando nosso controller ChamadoController com POSTMAN

02 – Melhorando o nosso projeto

Professor Anderson Henrique

