

PI – DESENVOLVIMENTO DE SISTEMA ORIENTADO A WEB

Prof. Me. Wilson Lourenço

Agenda



O ARQUIVO POM



TAGS



REVISÃO



IMPLEMENTANDO AS
CLASSES DE ENTIDADES

O ARQUIVO POM

- O POM é um dos arquivos mais importantes em um projeto Maven, ele descreve uma série de configurações que o projeto terá e quais repositórios e dependências seu projeto irá precisar.
- No cabeçalho de um POM temos algumas tags básicas que definem qual versão do modelo de POM utilizado.

- O seu GroupId que seria algo como o prefixo da estrutura de pacotes do projeto.
- O ArtifactId que define qual é o nome o artefato final .war ou .jar terá quando empacotado.
- Version define a versão do projeto que irá complementar o nome do artefato.

- A tag `Packaging` por sua vez define qual tipo de empacotamento o projeto terá após o processo de build, no nosso caso será um `.war`.
- E a tag `Name` define o nome do projeto.

TAGS

Exemplo

```
<modelVersion>4.0.0</modelVersion>  
<groupId>br.com.semeru</groupId>  
<artifactId>semeru_jsf_maven</artifactId>  
<version>1.0-SNAPSHOT</version>  
<packaging>war</packaging>  
<name>semeru_jsf_maven</name>
```


- A tag Properties possibilita definir por exemplo a versão do Spring, ou do JSF adotada para o projeto.
- Se a dependência do spring for definida como:
 - `<spring.version>3.0.5.RELEASE</spring.version>`

- Ao mudarmos a versão do Spring, por exemplo, para 3.1 todas as dependências serão baixadas para a versão 3.1.
- Podemos definir o contêiner web no qual será feito o deploy da aplicação (Tomcat) e o tipo de codificação utilizada pelo projeto, no nosso caso o UTF8 .

Exemplo

<properties>

 <spring.version>3.0.5.RELEASE</spring.version>

 <themes.version>1.0.8</themes.version>

 <jsf.version>2.1.7</jsf.version>

 <jstl.version>1.2</jstl.version>

 <netbeans.hint.deploy.server>Tomcat</netbeans.hint.deploy.server>

 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

- A tag `Repositories` define a lista de repositórios que serão acessados pelo Maven para baixar nossas dependências.
- Muitas vezes é necessário colocar um repositório prioritário no início do POM para que o Maven inicie a busca pelas dependências a partir dele.

Exemplo

```
<!-- PRIMEFACES REPOSITORY -->
<repository>
  <id>prime-repo</id>
  <name>PrimeFaces Maven Repository</name>
  <url>http://repository.primefaces.org</url>
  <layout>default</layout>
</repository>

<!-- FACELETS TAGLIBRARIES REPOSITORY -->

<repository>
  <id>org.springframework.security.taglibs.facelets</id>
  <url>http://spring-security-facelets-taglib.googlecode.com/svn/repo/</url>
</repository>

</repositories>
```

- A tag `dependencies` define quais serão as dependências utilizadas no projeto no trecho de código abaixo declaramos algumas das dependências necessárias para se trabalhar com JavaServer Faces.

Exemplo

```
<dependencies>
  <!-- || DEPENDÊNCIAS DO JAVA SERVER FACES || -->
  <!-- ##### JSF-API ##### -->
  <dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>${jsf.version}</version>
    <scope>compile</scope>
  </dependency>

  <!-- ##### JSF-IMPL ##### -->
  <dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-impl</artifactId>
    <version>${jsf.version}</version>
  </dependency>

  <!-- ##### JSTL ##### -->
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>${jstl.version}</version>
  </dependency>
</dependencies>
```

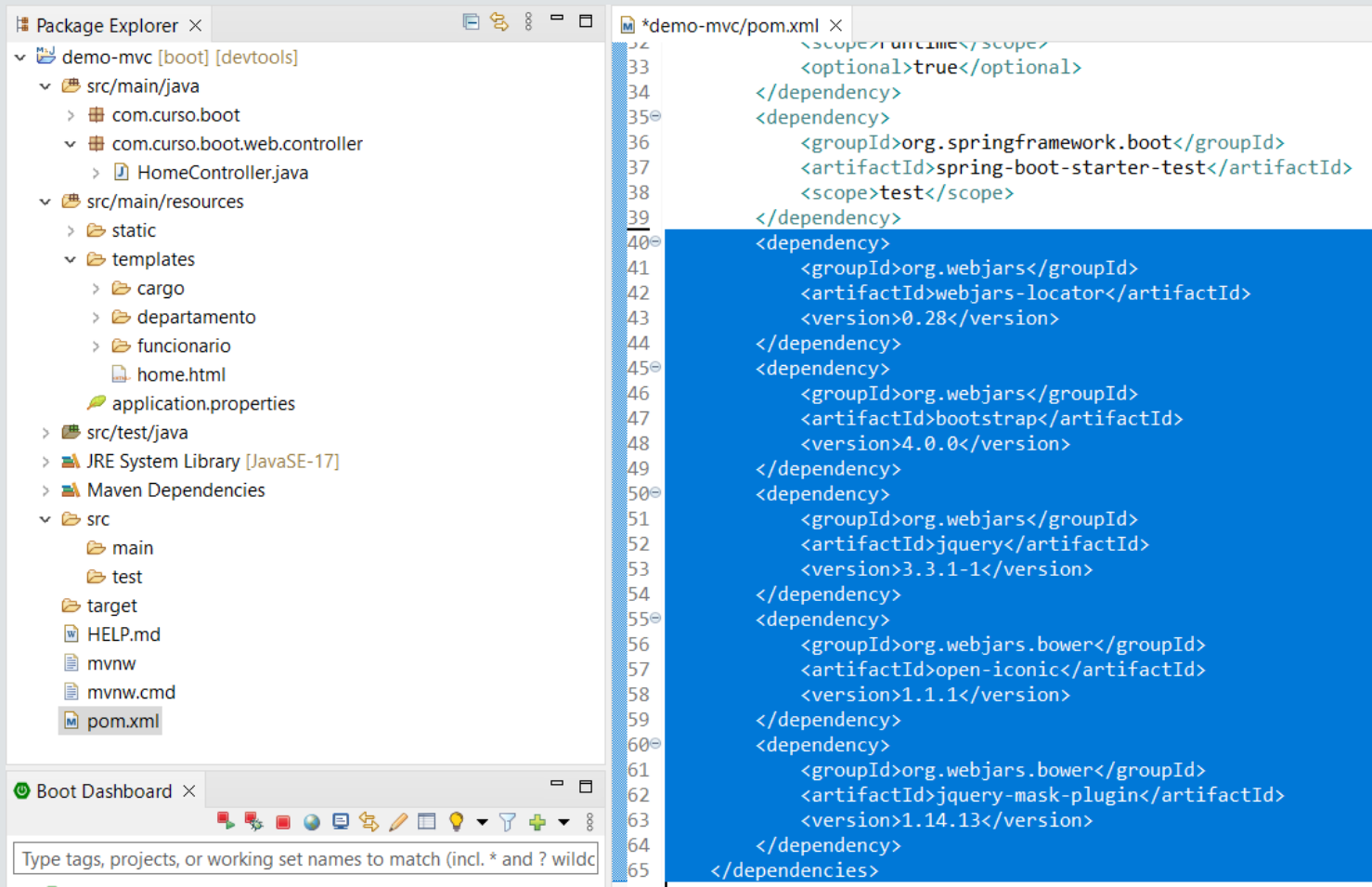
- Muitas vezes algumas dependências precisam ser excluídas.
- Isso acontece por vários motivos primeiro pode ser que o projeto já utilize uma versão diferente da mesma dependência.
- Um segundo motivo pode ser porque a dependência em questão gera conflitos com outras utilizadas no projeto.

- A sintaxe para remover uma dependência é como o exemplo a seguir:
- Nele estamos declarando a dependência do DOM4J mas estamos dizendo ao Maven para não baixar a dependência da XML-APIS.
- Se tirarmos essa exclusão o Maven vai analisar o POM da DOM4J e verificará que ela possui uma dependência da XML-APIS e assim entenderá que o projeto necessita dela e irá baixá-la.

```
<!-- ##### DOM4J ##### -->
<dependency>
  <artifactId>dom4j</artifactId>
  <groupId>dom4j</groupId>
  <type>jar</type>
  <version>1.6.1</version>
  <exclusions>
    <exclusion>
      <artifactId>xml-apis</artifactId>
      <groupId>xml-apis</groupId>
    </exclusion>
  </exclusions>
</dependency>
```

Exercício:

- Acrescente as dependências do nosso projeto, no arquivo pom.xml:



The image shows an IDE interface with two main panels. The left panel, titled 'Package Explorer', displays the project structure for 'demo-mvc'. It includes source code folders like 'src/main/java' (containing 'com.curso.boot' and 'com.curso.boot.web.controller'), 'src/main/resources' (containing 'static', 'templates', and 'application.properties'), 'src/test/java', and 'src' (containing 'main', 'test', 'target', 'HELP.md', 'mvnw', and 'mvnw.cmd'). The 'pom.xml' file is selected at the bottom. The right panel shows the content of 'pom.xml', which is a Maven Project Object Model file. It defines the project as 'demo-mvc' with version '1.0.0-SNAPSHOT' and uses the 'org.springframework.boot' parent. It includes several dependencies: 'spring-boot-starter-test' (scope: test), 'webjars-locator' (version: 0.28), 'bootstrap' (version: 4.0.0), 'jquery' (version: 3.3.1-1), 'open-iconic' (version: 1.1.1), and 'jquery-mask-plugin' (version: 1.14.13). The file is highlighted with a blue background.

Package Explorer ×

- demo-mvc [boot] [devtools]
 - src/main/java
 - com.curso.boot
 - com.curso.boot.web.controller
 - HomeController.java
 - src/main/resources
 - static
 - templates
 - cargo
 - departamento
 - funcionario
 - home.html
 - application.properties
 - src/test/java
 - JRE System Library [JavaSE-17]
 - Maven Dependencies
 - src
 - main
 - test
 - target
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

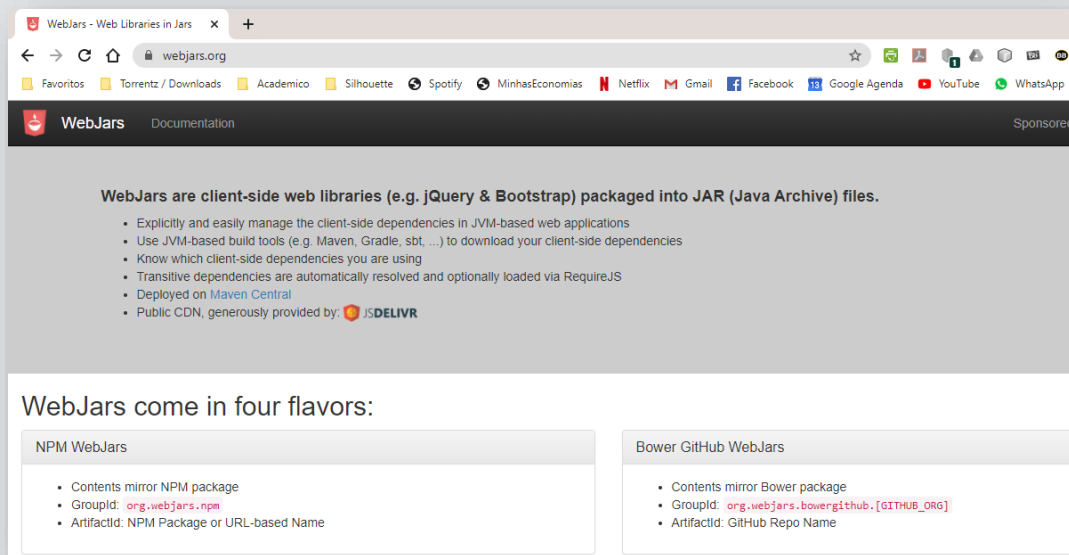
Boot Dashboard ×

Type tags, projects, or working set names to match (incl. * and ? wildc

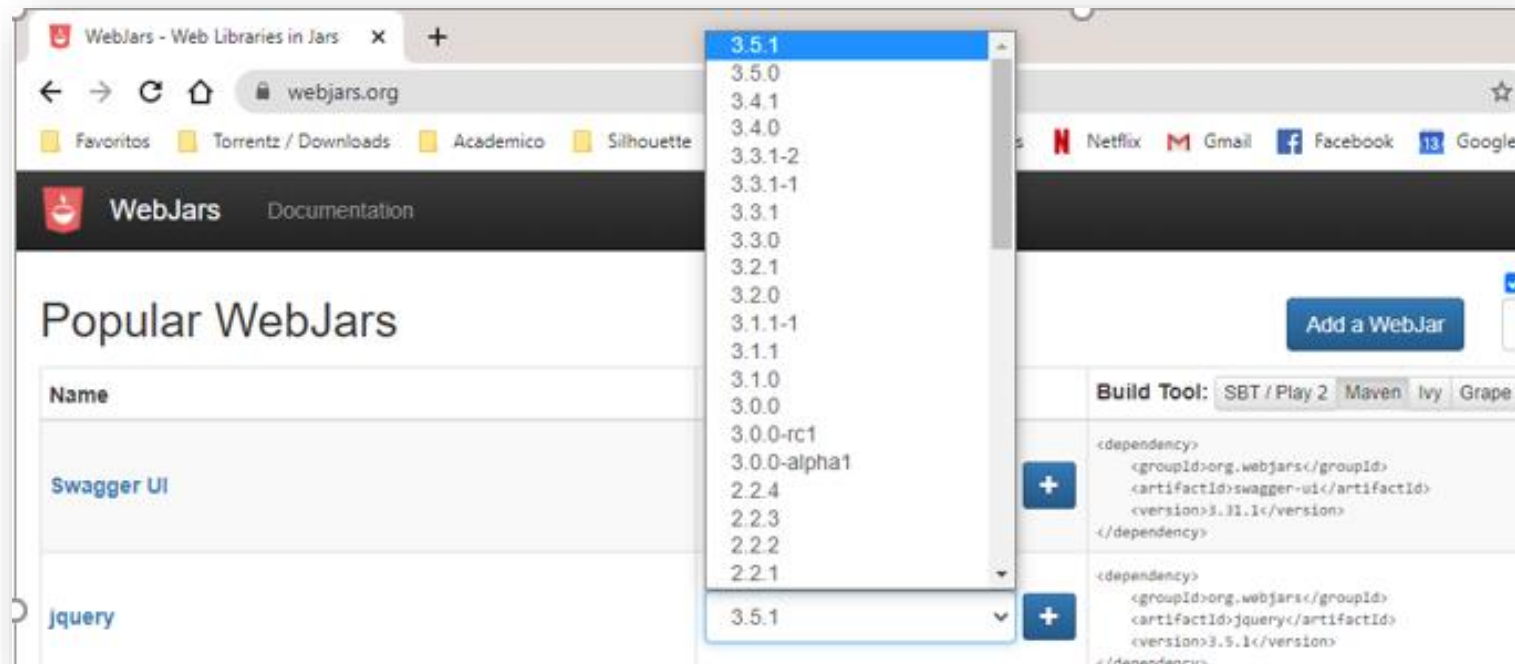
*demo-mvc/pom.xml ×

```
32 <scope>runtime</scope>
33 <optional>true</optional>
34 </dependency>
35 <dependency>
36 <groupId>org.springframework.boot</groupId>
37 <artifactId>spring-boot-starter-test</artifactId>
38 <scope>test</scope>
39 </dependency>
40 <dependency>
41 <groupId>org.webjars</groupId>
42 <artifactId>webjars-locator</artifactId>
43 <version>0.28</version>
44 </dependency>
45 <dependency>
46 <groupId>org.webjars</groupId>
47 <artifactId>bootstrap</artifactId>
48 <version>4.0.0</version>
49 </dependency>
50 <dependency>
51 <groupId>org.webjars</groupId>
52 <artifactId>jquery</artifactId>
53 <version>3.3.1-1</version>
54 </dependency>
55 <dependency>
56 <groupId>org.webjars.bower</groupId>
57 <artifactId>open-iconic</artifactId>
58 <version>1.1.1</version>
59 </dependency>
60 <dependency>
61 <groupId>org.webjars.bower</groupId>
62 <artifactId>jquery-mask-plugin</artifactId>
63 <version>1.14.13</version>
64 </dependency>
65 </dependencies>
```

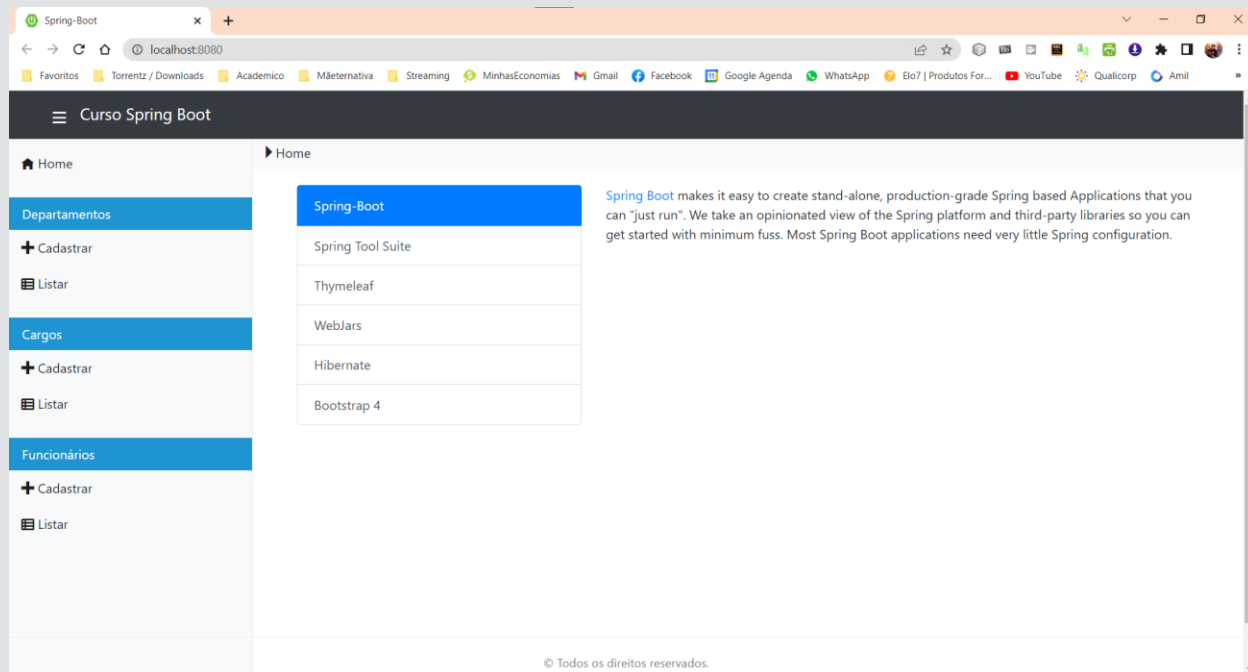

- Como descobrir qual dependência para webjars adicionar?
 - Quando precisar de uma dependência webjars, acesse a página webjars.org.



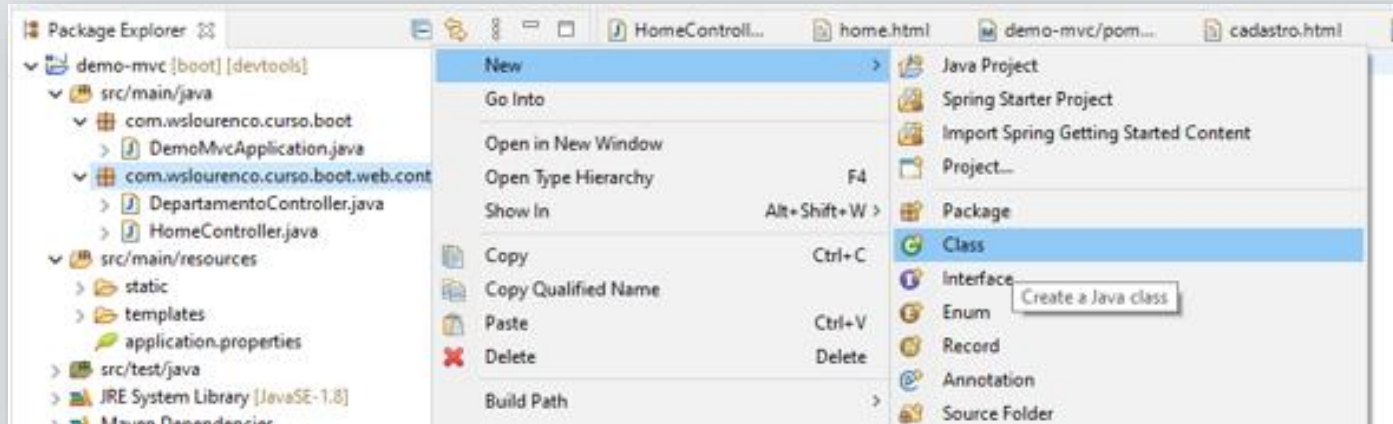
- Selecione o que você procura, por exemplo:
 - Como acrescentar uma dependência para o JQuery, selecione JQuery, a versão e a aba Maven, copie a dependência e cole no seu arquivo POM.xml.



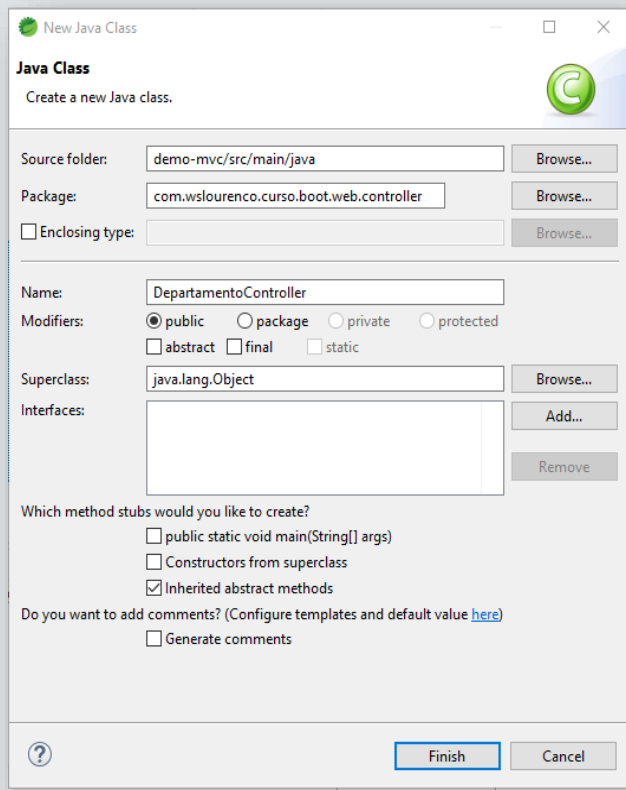
- Execute o projeto, e repare que agora, com as dependências baixadas, estamos usando a formatação do bootstrap.



- Vamos agora criar uma classe, clique com o botão direito do mouse no pacote `com.curso.boot.web.controller`, vá em `New/Class`



- Digite DepartamentoController em name e clique em Finish:



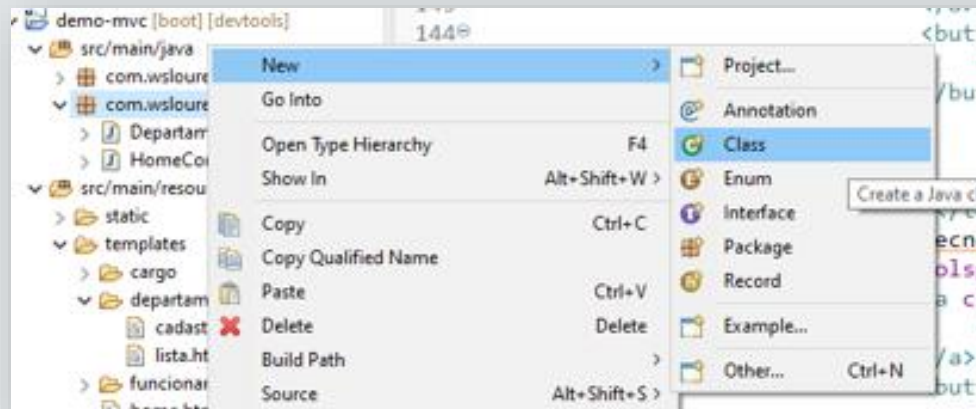
- A classe DepartamentoController será criada, digite o código abaixo:

```
DepartamentoController.java x
1 package com.curso.boot.web.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestMapping;
6
7 @Controller
8 @RequestMapping("/departamentos")
9 public class DepartamentoController {
10
11     @GetMapping("/cadastrar")
12     public String Cadastrar() {
13         return "/departamento/cadastro";
14     }
15
16     @GetMapping("/listar")
17     public String Listar() {
18         return "/departamento/lista";
19     }
20 }
```

- No arquivo home.html, note os href's para cadastrar e listar, departamentos e o Controller da classe, cadastrar e listar, são os Controllers dos métodos.

```
home.html x
39         class="oi oi-home"></i> <span>Home</span>
40     </a></li>
41 </ul>
42
43 <ul class="nav nav-pills">
44 <li class="nav-item">
45     <span class="nav-link active" >Departamentos</span>
46 </li>
47 <li class="nav-item">
48     <a class="nav-link" href="/departamentos/cadastrar">
49         <i class="oi oi-plus"></i>
50         <span>Cadastrar</span>
51     </a>
52 </li>
53 <li class="nav-item">
54     <a class="nav-link" href="/departamentos/listar">
55         <i class="oi oi-spreadsheet"></i>
56         <span>Listar</span>
57     </a>
58 </li>
59 </ul>
60
61 <ul class="nav nav-pills">
62     <li class="nav-item"><span class="nav-link active" >Cargos
```

- Inclua mais duas classes, CargoController e FuncionárioController:



- Digite CargoController em name, e repita a operação para criar também a classe FuncionarioController.

New Java Class

Java Class

Create a new Java class.

Source folder: demo-mvc/src/main/java Browse...

Package: com.curso.boot.web.controller Browse...

Enclosing type: Browse...

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

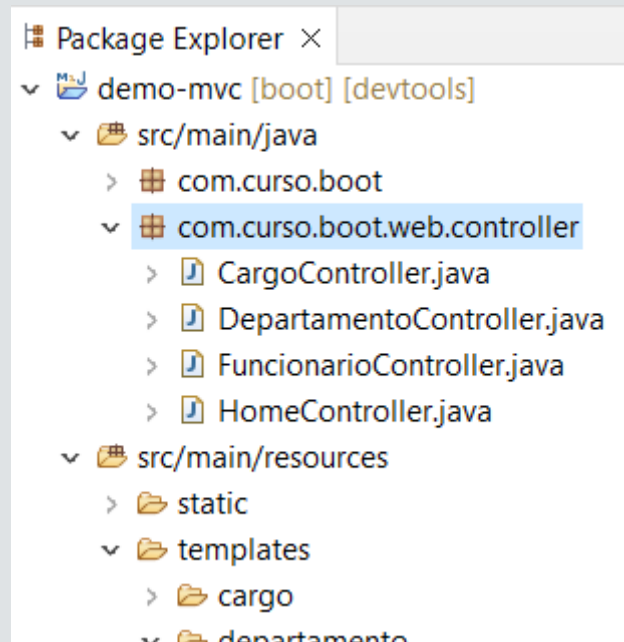
Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods
☐ Generate comments

Do you want to add comments? (Configure templates and default value [here](#))

Finish Cancel

- O pacote `com.curso.boot.web.controller`, deve ficar assim:



- Digite o código da classe FuncionarioController:

```
*FuncionarioController.java ×
1 package com.curso.boot.web.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestMapping;
6
7 @Controller
8 @RequestMapping("/funcionarios")
9 public class FuncionarioController {
10
11     @GetMapping("/cadastrar")
12     public String Cadastrar() {
13         return "/funcionario/cadastro";
14     }
15
16     @GetMapping("/listar")
17     public String Listar() {
18         return "/funcionario/lista";
19     }
20
21 }
```


- Digite o código da classe CargoController:

```
package com.wslourenco.curso.boot.web.controller;

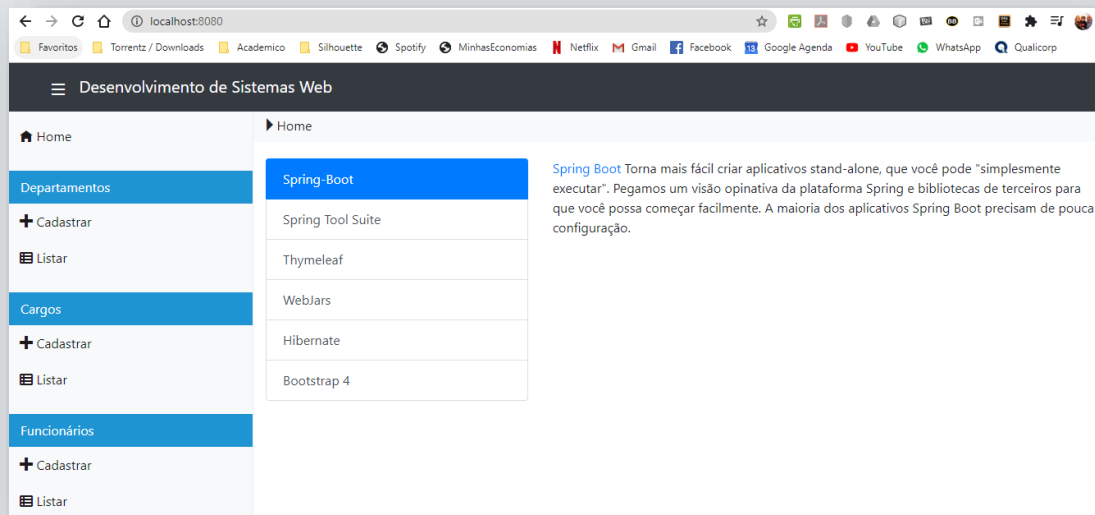
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/cargos")
public class CargoController {

    @GetMapping("/cadastrar")
    public String Cadastrar() {
        return "/cargo/cadastro";
    }

    @GetMapping("/listar")
    public String Listar() {
        return "/cargo/lista";
    }
}
```

- Execute e teste o projeto, clicando nos menus:



REVISÃO

Spring Boot Starter

- O Spring Boot tem como base fornecer os recursos dos módulos do Spring Framework e demais projetos vinculados a ele, por meio de dependências nomeadas como starters.

- Entre os diversos starters, que podem ser visualizados no capítulo 13.5 do manual de referência do Spring Boot 1.5.10.RELEASE, temos o principal que é o spring-boot-starter-parent.
- Esse starter tem como objetivo informar a versão do Spring Boot que será usada no projeto.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.2.9.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

- Todos os demais starters declarados no arquivo de gerenciamento de dependências, como o Maven, serão baseados nas versões referentes a do spring-boot-starter-parent.
- Assim, não é necessário informar a tag de versão da dependência declarada como starter do módulo a ser incluído no projeto.

- Como exemplo, veja o módulo web, o qual fornece os recursos para uso do Spring MVC.

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-web</artifactId>  
</dependency>
```


- Outro item importante que deve ser incluído no arquivo pom.xml do Maven é o plugin de build spring-boot-maven-plugin. Ele tem como responsabilidade gerar um arquivo .jar executável como artefato do projeto.
- O Spring Boot é executado por meio de uma classe principal, por isso, o artefato é um .jar executável.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

- A classe principal, responsável por executar, ou seja, inicializar o Spring Boot é similar a apresentada a seguir:

```
@SpringBootApplication
public class DemoMvcApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoMvcApplication.class, args);
    }
}
```

- No código da classe `DemoMvcApplication` podemos ver o uso da anotação `@SpringBootApplication`, responsável por incluir no projeto alguns recursos de inicialização do Spring Framework.
- Já no método `main()` a instrução `SpringApplication.run()` recebe como parâmetro a classe que contém a anotação `@SpringBootApplication`.

Páginas e Arquivos Estáticos

- Quando trabalhamos com Thymeleaf junto ao Spring Boot, os arquivos de páginas (.html) e arquivos estáticos como CSS, JS ou imagens, devem ser adicionados no classpath do projeto, ou seja, diretório src/main/resources.

- Nesse diretório teremos dois sub-diretórios principais, o static, onde serão armazenados os arquivos estáticos e o template, onde serão adicionadas os htmls:

```
src/main/resources/static/css/style.css  
src/main/resources/static/js/jquery.js  
src/main/resources/static/templates/index.html
```

WebJars

- O projeto WebJars tem como objetivo fornecer arquivos .jar contendo bibliotecas de CSS, JS ou mesmo de imagens.
- Esses arquivos podem então ser adicionados no projeto como uma dependência no arquivo pom.xml

- Deste modo, ao invés de realizar o download dos códigos fonte de recursos como o Bootstrap ou jQuery e incluí-los como recursos estáticos, incluimos no lugar a biblioteca do WebJar referente.

```
<dependency>  
  <groupId>org.webjars</groupId>  
  <artifactId>bootstrap</artifactId>  
  <version>4.0.0</version>  
</dependency>
```


- Uma das vantagens desse recurso é que se for necessário alterar a versão da biblioteca, basta alterá-la no arquivo pom.xml.
- Outra vantagem é que esta dependência estará disponível no repositório local do Maven, facilitando o uso dela em qualquer outro projeto que esteja sendo desenvolvido.

- Para usar este recurso junto ao Spring Boot, é necessário incluir a dependência do webjars-locator.
- Essa dependência já faz parte dos recursos fornecidos pelo Spring Boot e assim, não é necessário adicionar uma versão ao declará-la.
- A inclusão dela no arquivo pom.xml é para dizer ao Spring Boot que você vai fazer uso de seus recursos.

```
<dependency>  
  <groupId>org.webjars</groupId>  
  <artifactId>webjars-locator</artifactId>  
</dependency>
```

- E a finalidade principal da webjars-locator é relacionar as dependências de webjars (Bootstrap, jQuery, ...) com as urls incluídas nas páginas html.

```
<link href="/webjars/bootstrap/css/bootstrap.min.css" rel="stylesheet"/>
```

- Desse modo, a página consegue localizar as bibliotecas estáticas entre as dependências do projeto e assim, quando a página abrir no navegador vai conseguir acessar tais recursos.

Resolvendo Solicitações

- Após a inclusão das páginas no projeto, será necessário preparar os controladores para que as solicitações sejam processadas e assim, as páginas possam ser acessadas.

- Por exemplo, a página home.html tem no menu lateral o link Cadastrar para abrir a página cadastro.html referente a Departamentos. Este link é declarado da seguinte forma:

```
<a                class="nav-link"  
href="/departamentos/cadastrar">Cadastrar</a>
```

- Observe que a url possui dois caminhos que são:
- /departamentos - tem como função acessar a classe DepartamentoController;
- /cadastrar - tem como responsabilidade acessar o método mapeado com este caminho dentro do controller de departamentos.

- Em DepartamentoController a anotação @RequestMapping, declarada sobre a assinatura da classe, tem como valor o caminho /departamentos.
- Dessa forma, a url do link Cadastrar vai chegar até esse controller.


```
@Controller
@RequestMapping("/departamentos")
public class DepartamentoController {

    @GetMapping("/cadastrar")
    public String cadastrar() {
        return "/departamento/cadastro";
    }
}
```

- Em seguida, o acesso chega ao método `cadastrar()`, por conta da anotação `@GetMapping`, que possui o caminho de acesso para `/cadastrar`.
- Como resposta, o método `cadastrar()` retorna um objeto `String` com o valor `/departamento/cadastro`.

- Esse retorno vai abrir a página cadastro.html que está armazenada no diretório /templates/departamento.
- Entretanto, não se declara no retorno o /templates nem o .html, essas instruções estão implícitas para o Spring MVC.

IMPLEMENTANDO AS CLASSES DE ENTIDADES

JPA

- JPA é um framework leve, baseado em POJOS (Plain Old Java Objects) para persistir objetos Java.
- A Java Persistence API, diferente do que muitos imaginam, não é apenas um framework para Mapeamento Objeto-Relacional (ORM - Object-Relational Mapping), ela também oferece diversas funcionalidades essenciais em qualquer aplicação corporativa.

- Atualmente, praticamente todas as aplicações de grande porte utilizam JPA para persistir objetos Java. JPA provê diversas funcionalidades para os programadores.

HIBERNATE

- Um grande problema enfrentado pelos desenvolvedores que trabalham com linguagem orientada a objetos é o mapeamento desses objetos em banco de dados.
- Isso se deve a um grande número de bancos de dados, inclusive os maiores do mercado, trabalharem ainda no modelo relacional.

- Dessa forma, desenvolvedores necessitam criar mecanismos para converter dados em objetos e vice-versa, o que acaba desvirtuando-o do seu foco principal e da grande expectativa do cliente, que é o software final.
- Algumas soluções de mercado facilitam essa conversão, dentre estas o Hibernate.

- O Hibernate hoje é o framework Java para mapeamento objeto relacional mais conhecido no mercado.
- Sua principal função é abstrair o mapeamento, economizando esforço e preocupações concernentes a tal tarefa.
- Com uma arquitetura simples, de fácil configuração, e com funções de fácil entendimento, o Hibernate simplifica bastante a tarefa do desenvolvedor.

- Na versão 3.x o Hibernate implementa a especificação JPA (Java Persistence API) através do conceito de anotações (implementada a partir do JDK5), o que facilita ainda mais o mapeamento objeto-relacional, que pode agora ser feito diretamente na classe.

- Para demonstrar todo o poder de utilização do framework, apresentamos um exemplo de aplicação que expressa de forma prática todos os conceitos abordados.
- Muitos outros conceitos referentes ao Hibernate não foram abordados, pois o objetivo principal, é apresentar uma visão para desenvolvedores iniciantes.

- Trabalhar com softwares orientados a objetos e banco de dados relacionais pode ser enfadonho e consumir muito tempo de desenvolvimento.
- Quando a linguagem de programação Java começou a tomar força no mercado, várias soluções começaram a surgir para resolver este problema.

- A vencedora dessas soluções, que desbancou até mesmo o padrão EJB 2.0, foi o Hibernate, uma ferramenta para mapeamento objeto/relacional para ambientes Java.

- O termo mapeamento objeto/relacional (ORM) refere-se à técnica de mapeamento de uma representação de dados em um modelo de objetos para um modelo de dados relacional.
- O Hibernate não cuida somente do mapeamento das classes Java para tabelas do banco de dados (e dos tipos de dados Java para os tipos de dados SQL).

- Ele também provê facilidades para consultar e retornar os dados da consulta, e pode reduzir significativamente o tempo de desenvolvimento em contrapartida ao alto tempo gasto pelas operações manuais dos dados feitas com SQL e JDBC.
- O Hibernate é uma ferramenta de alta performance.

- Uma das soluções ORM mais flexíveis e poderosas no mercado, ele faz o mapeamento de classes Java para tabelas de banco de dados e de tipos de dados Java para tipos de dados SQL.
- Ele fornece consultas e facilidades para retorno dos dados que reduzem significativamente o tempo de desenvolvimento.

- A meta do projeto do Hibernate é aliviar os desenvolvedores de 95% das tarefas comuns de programação relacionadas à persistência, como a codificação manual com SQL e a API JDBC.
- O Hibernate gera o SQL para a aplicação, não necessitando o tratamento dos “resultsets” (comuns nas conexões manuais com JDBC), faz a conversão entre registros e objetos e permite que sua aplicação seja portátil para qualquer banco de dados SQL.

Dicas para Estudo



Seja “CURIOSO”:

Procure revisar o que foi estudado.

Pesquise as referências bibliográficas.



Seja “ANTENADO”:

Leia a próxima aula.



Seja
“COLABORATIVO”:

Traga assuntos relevantes para a sala de aula.

Participe da aula.

Proponha discussões relevantes sobre o conteúdo.



Prof. Me. Wilson Lourenço



**Dúvidas?
Não mais..**