FIAP GRADUAÇÃO

FIAP

DIGITAL BUSINESS ENABLEMENT

Prof. Me. Thiago T. I. Yamamoto

#12 – INTRODUÇÃO - SPRING MVC

 thiagoyama  thiagoyama@gmail.com

#12 – INTRODUÇÃO AO SPRING MVC

- Spring Framework
- Primeiro projeto Spring MVC
- Controllers
- Envio de dados da View para o Controlller
- Envio de dados do Controller para a View

CONTEÚDO PROGRAMÁTICO

SPRING FRAMEWORK

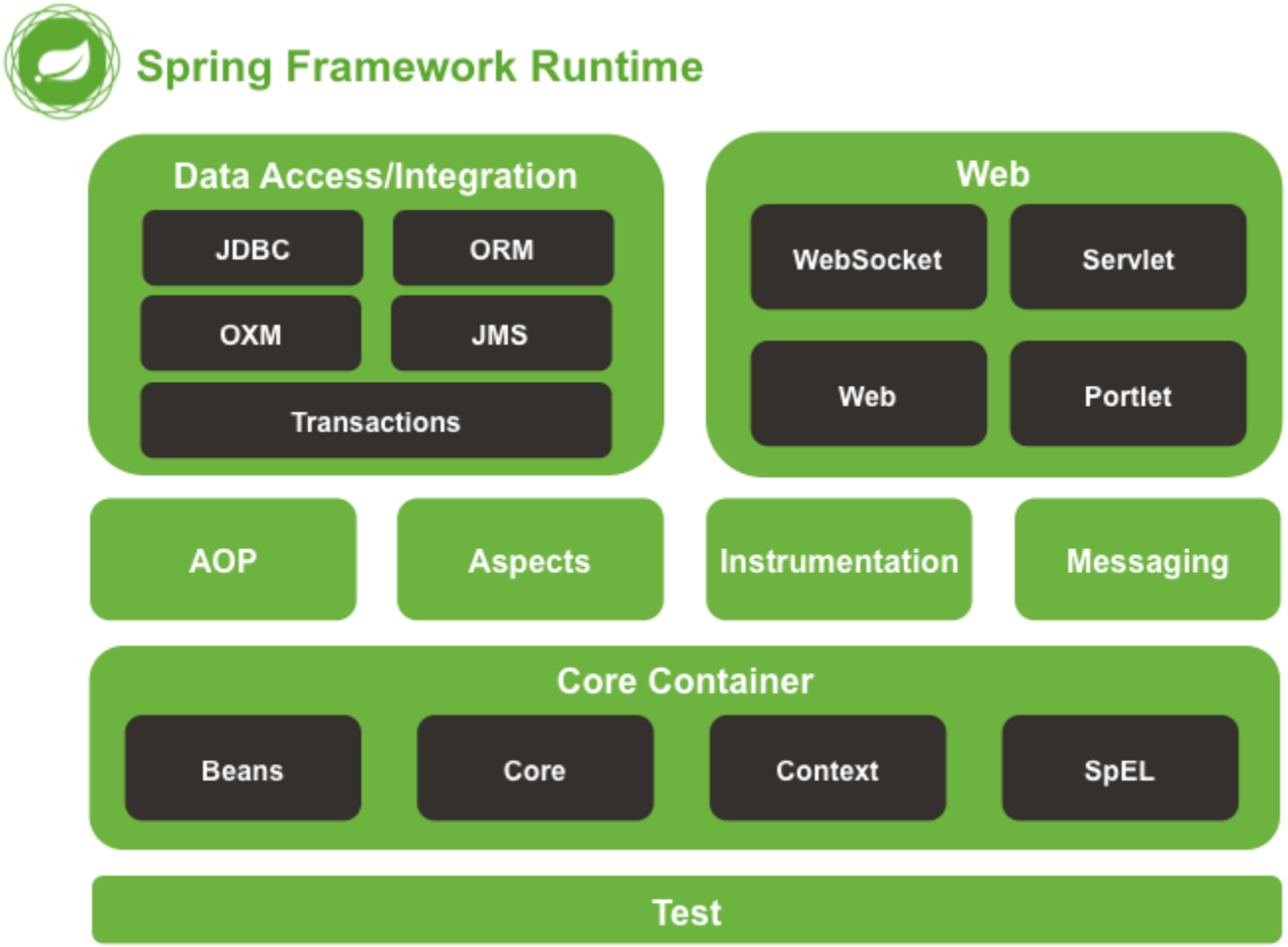
- O Spring é um framework de **código aberto** (opensource), criado em 2002 por Rod Johnson. Esse framework foi criado para ser uma **alternativa** ao **JavaEE** no desenvolvimento de aplicações corporativas.

- O Spring possui diversos módulos como Spring Data (persistencia), Spring Security, Spring Social..

- Porém, o módulo principal é o de **injeção de dependências;**

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/

SPRING FRAMEWORK

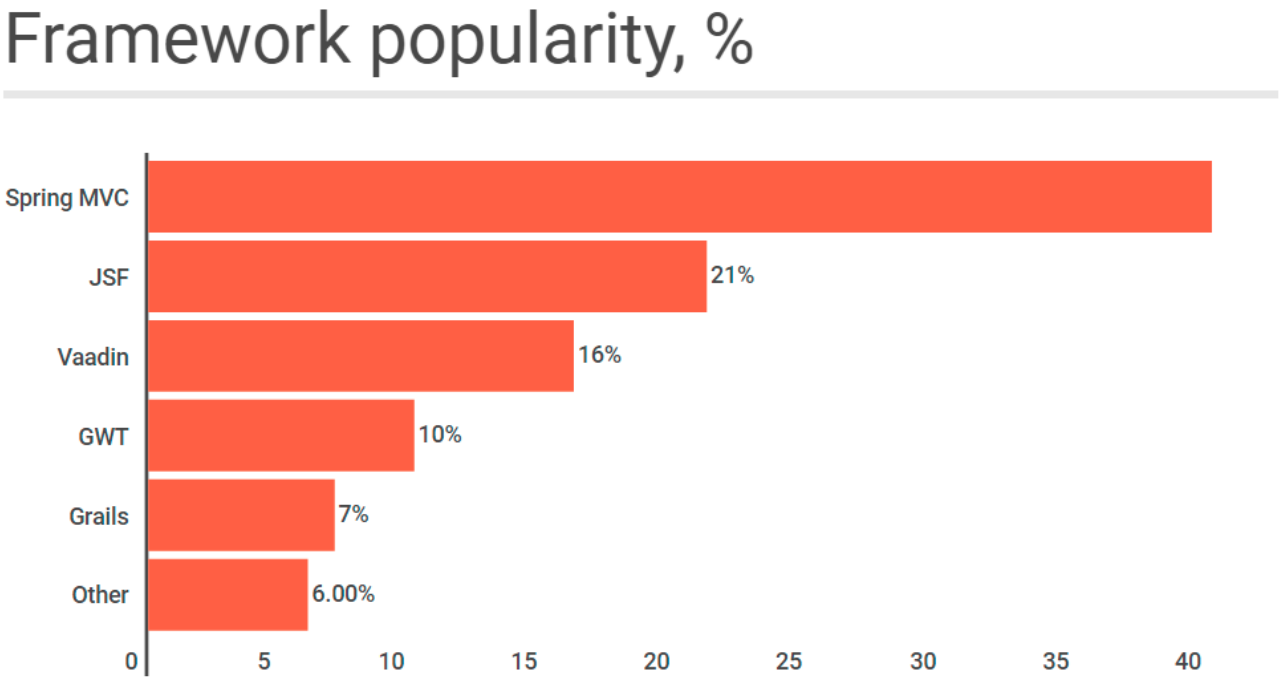


http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html

SPRING MVC

SPRING MVC

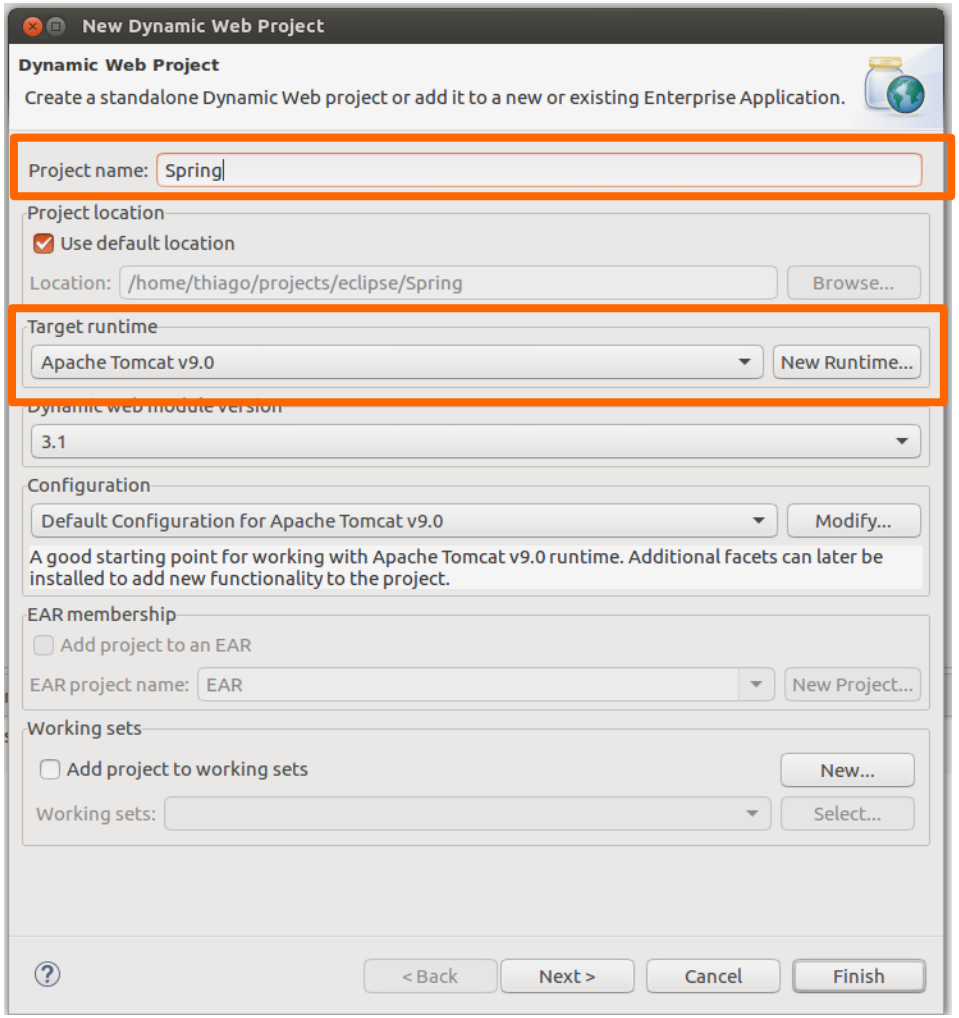
O **Spring MVC** é um dos módulos mais populares do Spring Framework. Isso se deve a facilidade de desenvolvimento de aplicações Java Web com o padrão MVC.



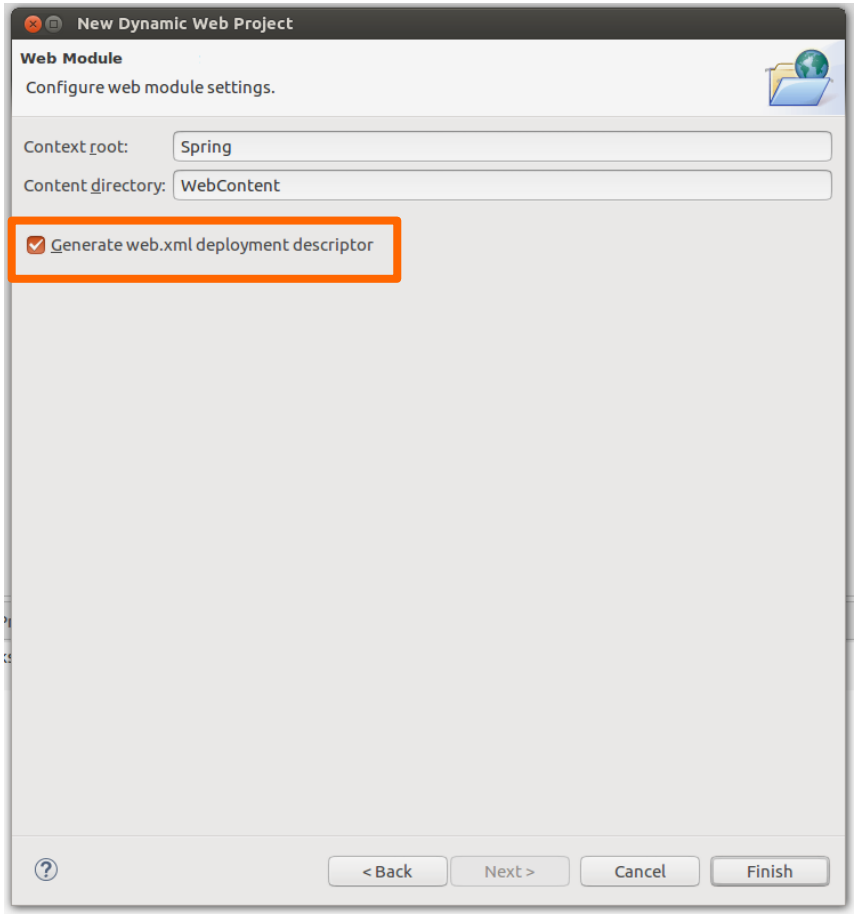
Source: RebelLabs for Zero TurnAround Inc.

PRIMEIRO PROJETO SPRING MVC

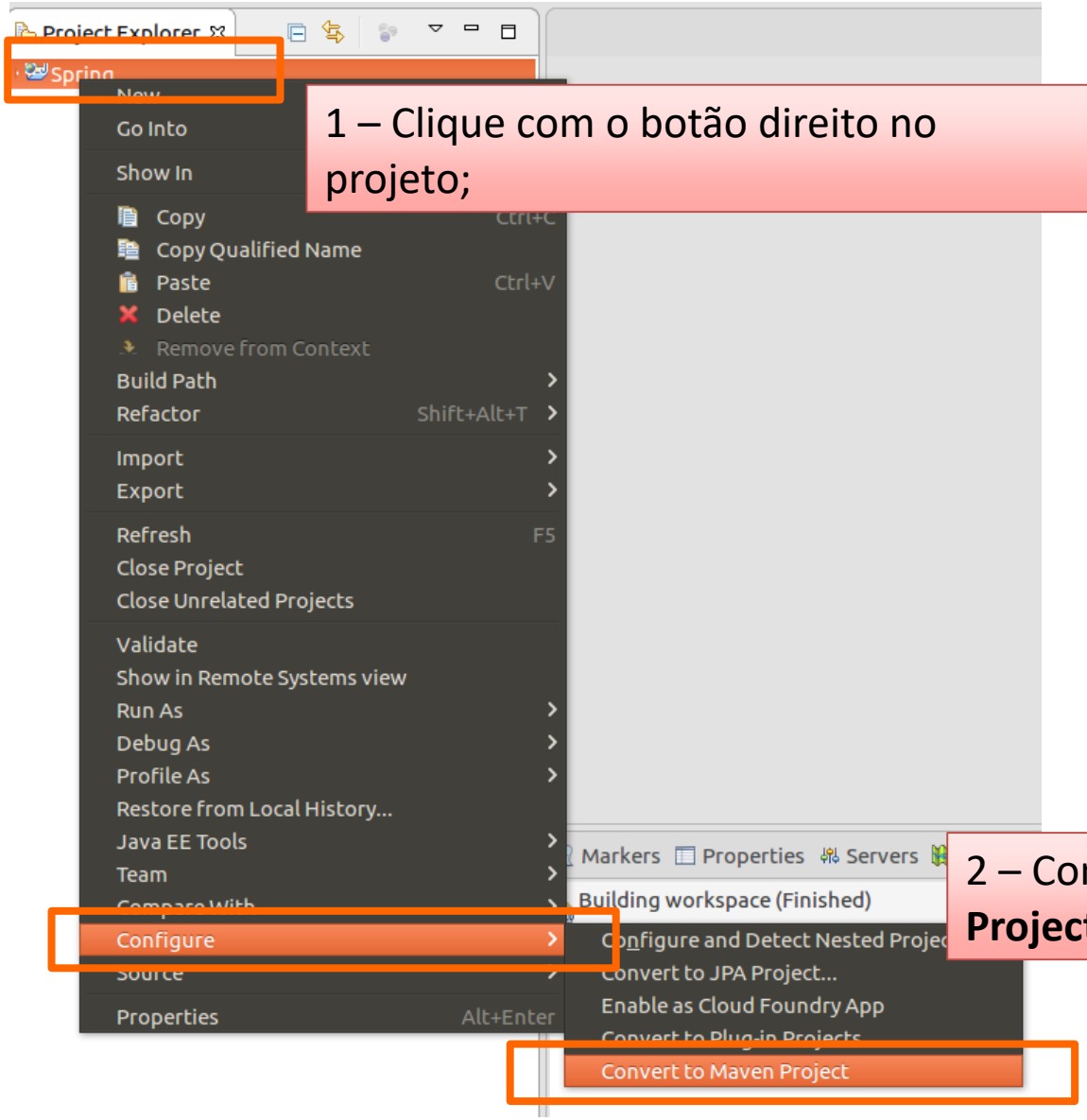
1 – Crie um **Dynamic Web Project** utilizando o tomcat.



2 – Marque para criar o **web.xml**



MAVEN NO PROJETO



1 – Clique com o botão direito no projeto;

2 – Converta o projeto para **Maven Project**

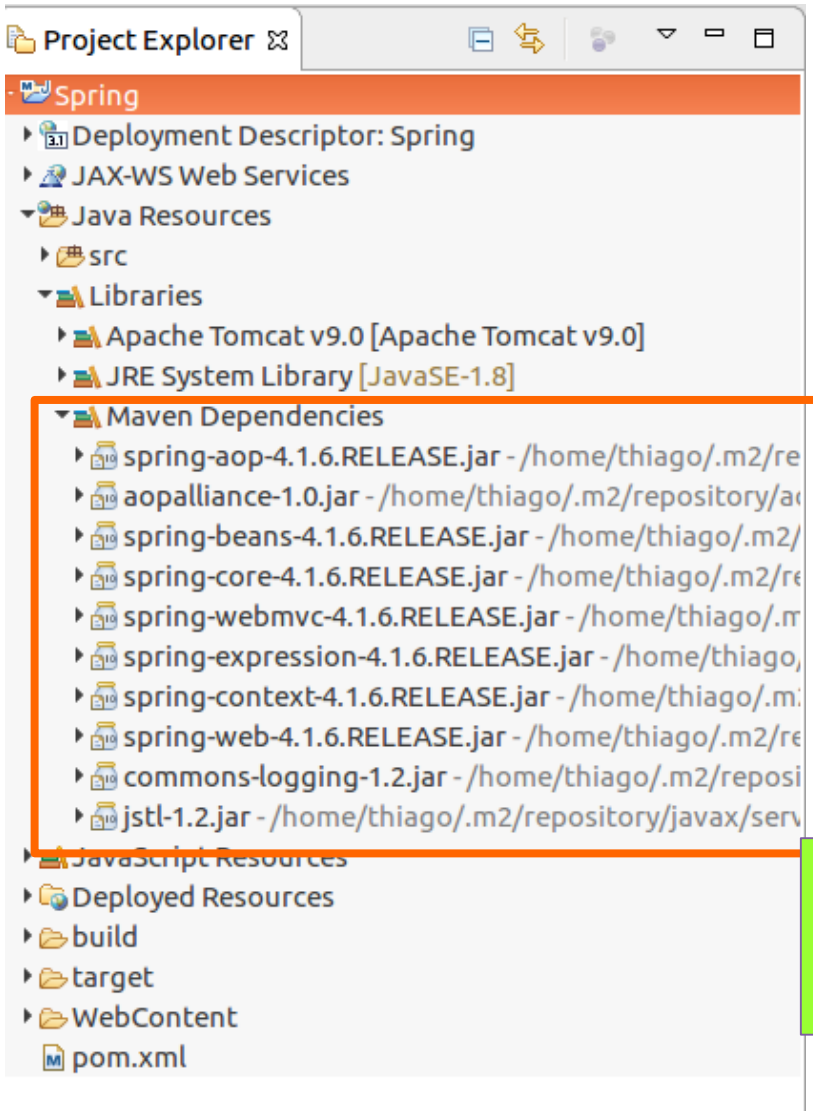
GERENCIANDO AS DEPENDÊNCIAS COM MAVEN

```
<dependencies>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-aop</artifactId>
			<version>4.3.5.RELEASE</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>4.3.5.RELEASE</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>4.3.5.RELEASE</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-web</artifactId>
			<version>4.3.5.RELEASE</version>
		</dependency>
		<dependency>
			<groupId>commons-logging</groupId>
			<artifactId>commons-logging</artifactId>
			<version>1.2</version>
		</dependency>
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>jstl</artifactId>
			<version>1.2</version>
		</dependency>
</dependencies>
```

Abra o arquivo **pom.xml** e adicione as dependências do Spring MVC.

Obs. Adicione logo abaixo da tag **</build>**

MAVEN DEPENDENCIES



Após a configuração é possível visualizar as dependências adicionadas pelo Maven.

SPRING MVC - CONFIGURAÇÕES

FIAP

WEB.XML

```
 <servlet>
  <servlet-name>spring mvc</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-context.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>spring mvc</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

No arquivo **web.xml** adicione o mapeamento da servlet do spring mvc.

Crie o arquivo de configuração do Spring, de acordo com o valor do parâmetro (diretório e nome).

SPRING MVC - CONFIGURAÇÕES

SPRING-CONTEXT.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/mvc
   http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd
   http://www.springframework.org/schema/tx
   http://www.springframework.org/schema/tx/spring-tx-4.1.xsd">

<context:component-scan base-package="br.com.exemplo" />

<mvc:annotation-driven />

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
</bean>

</beans>
```

Pacote base para os bean do spring

Habilita a utilização de annotations

Configuração das páginas, localização e extensão. Crie o diretório para as páginas jsp.

CONTROLLER

CONTROLLER

- As urls são mapeadas para métodos da classe controller;

Controller:

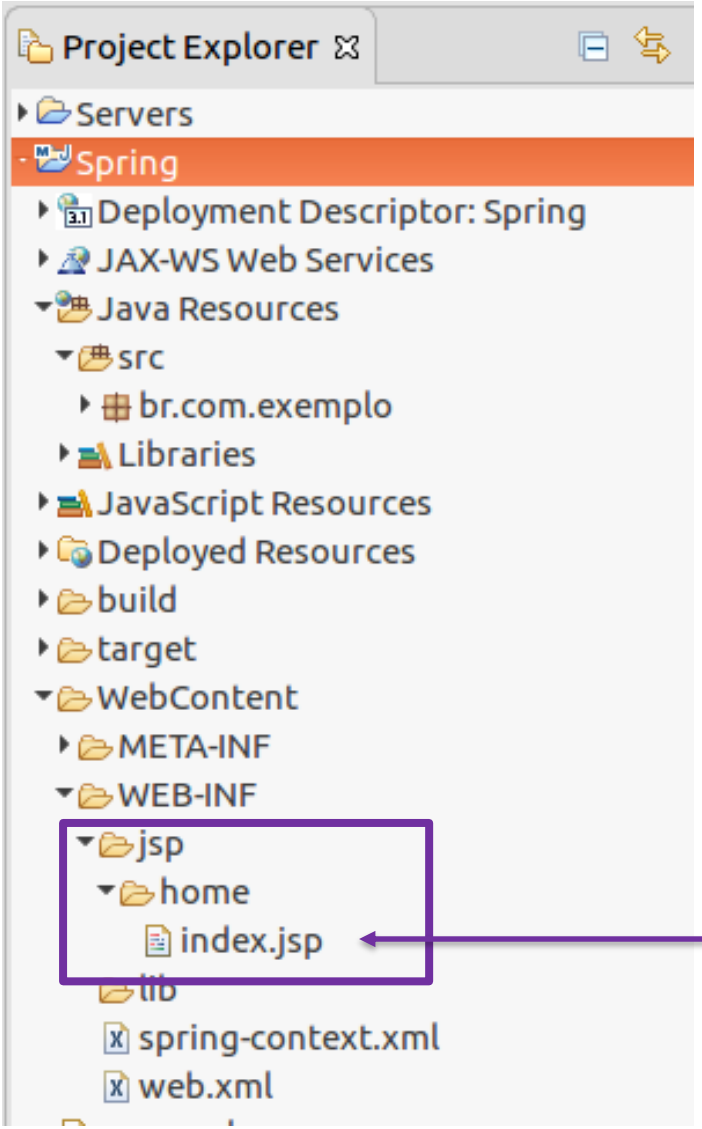
Responsabilidades básicas:

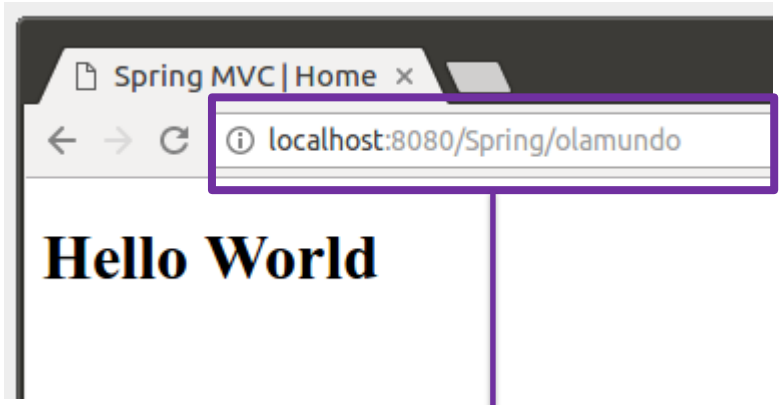
- Recuperar dados enviados pelo usuário;
- Interagir com a camada model;
- Acionar a camada de apresentação para enviar a resposta ao usuário;

Regras:

- A classe deve ser anotada com **@Controller**

CONTROLLER



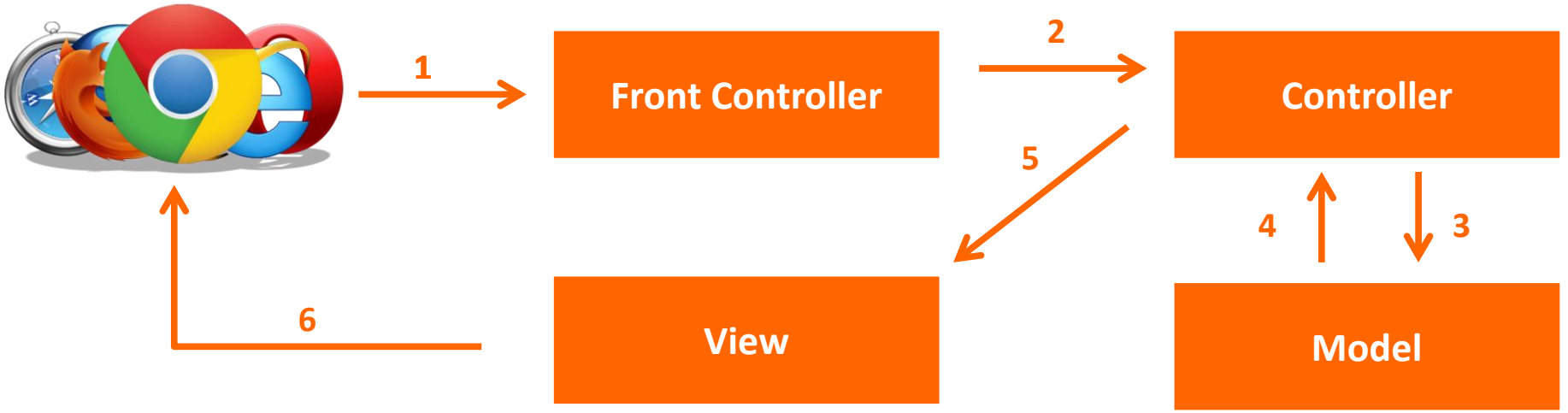



```
@Controller
public class HomeController {

        @RequestMapping("/olamundo")
        public String index(){
                return "home/index";
        }

}
```

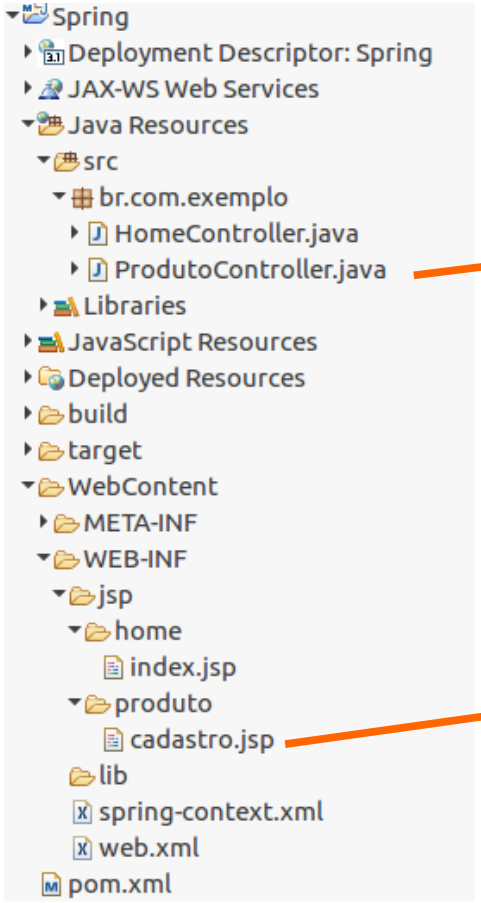
SPRING MVC



1. O usuário realiza uma requisição que será recebida pelo controller do framework Spring MVC;
2. O controller do Spring vai redirecionar para o Controller responsável por tratar a requisição;
3. O controller acessa o model para realizar as regras de negócio, acesso ao banco e etc..
4. O model retorna as informações para o controller;
5. O controller retorna as informações e o nome da *view* (página) para o framework processa-la e gerar o HTML final;
6. A página final é devolvida para o usuário.

RECEBENDO PARÂMETROS DA VIEW

Vamos criar um formulário de cadastro de um produto, para isso é preciso criar um novo controller com uma action e uma view:



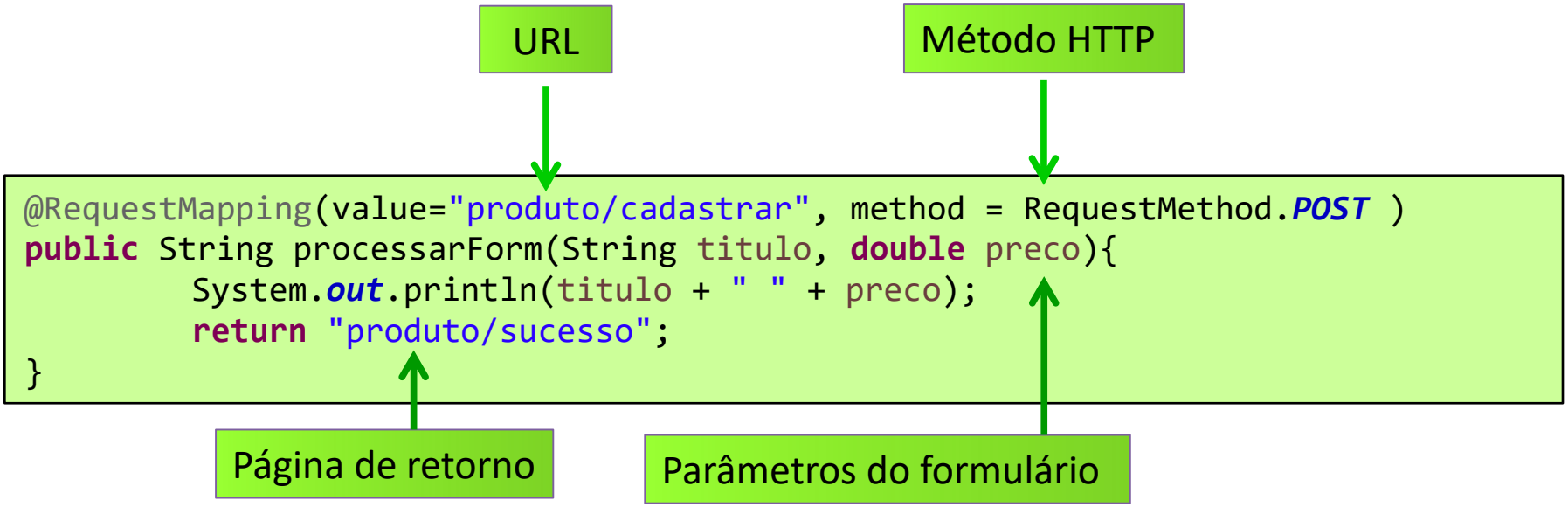

```java
@Controller
public class ProdutoController {

        @RequestMapping("produto/cadastrar")
        public String abrirForm(){
                return "produto/cadastro";
        }
}
```

```html
<form action="cadastrar" method="post">
        <input type="text" name="titulo"
                        placeholder="Digite o título">
        <input type="text" name="preco"
                        placeholder="Digite o preço">
        <input type="submit" value="Salvar">
</form>
```

RECEBENDO PARÂMETROS DA VIEW

Agora é preciso criar a action que irá tratar as informações do formulário:



Os parâmetros dos métodos devem ter os mesmos nomes dos campos do formulário;

A conversão de dados é feita automaticamente.

RECEBENDO PARÂMETROS DA VIEW

É possível também receber um objeto que encapsula os valores recebidos:

O nome dos atributos devem ser iguais aos nomes dos campos do formulário.

```
<form action="cadastrar" method="post">
            <input type="text" name="titulo"
                                placeholder="Digite o título">
            <input type="text" name="preco"
                                placeholder="Digite o preço">
            <input type="submit" value="Salvar">
</form>
```

```
public class Produto {

        private String titulo;

        private double preco;

        // gets e sets..

}
```

```
@RequestMapping(value="produto/cadastrar", method = RequestMethod.POST )
public String processarForm(Produto produto){
        System.out.println(produto.getTitulo() + " " + produto.getPreco());
        return "produto/sucesso";
}
```

@REQUESTMAPPING

Podemos utilizar a anotação **@RequestMapping** para o controller;

Dessa forma, não será preciso repetir em todos os métodos a parte da url *"/produto"*:

```
@Controller
@RequestMapping("/produto")
public class ProdutoController {

		@RequestMapping("cadastrar")
		public String abrirForm(){
			return "produto/cadastro";
		}

		@RequestMapping(value="cadastrar", method = RequestMethod.POST )
		public String processarForm(Produto produto){
			System.out.println(produto.getTitulo() + " " + produto.getPreco());
			return "produto/sucesso";
		}

}
```

@REQUESTMAPPING

Na anotação **@RequestMapping** podemos configurar o tipo de método HTTP que a action irá atender: **GET, POST, PUT e DELETE;**

```
@RequestMapping(value="cadastrar", method = RequestMethod.POST )
public String processarForm(Produto produto){
        System.out.println(produto.getTitulo() + " " + produto.getPreco());
        return "produto/sucesso";
}
```

No Spring Framework 4.3 foram adicionados outras anotações para facilitar o mapeamento das ações e métodos HTTP:

- **@GetMapping**
- **@PostMapping**
- **@DeleteMapping**
- **@PutMapping**

```
@Controller
@RequestMapping("/produto")
public class ProdutoController {

        @GetMapping("cadastrar")
        public String abrirForm(){
                return "produto/cadastro";
        }

        @PostMapping(value="cadastrar")
        public String processarForm(Produto produto){
                System.out.println(produto.getTitulo() + " " +
                return "produto/sucesso";
        }

}
```

ENVIANDO VALORES PARA A VIEW

Podemos enviar informações para a view através do objeto **ModelAndView**;

Esse objeto pode receber o nome da view que será exibida para o usuário e os valores para a view processar.

```
@PostMapping(value="cadastrar")
public ModelAndView processarForm(Produto produto){
        ModelAndView retorno = new ModelAndView("produto/sucesso");
        retorno.addObject("prod", produto);
        return retorno;
}
```

Valores para a view

View que será exibida

ENVIANDO VALORES PARA A VIEW

- Na página JSP podemos recuperar a informação utilizando **Expression Language;**
- A EL é delimitada pelos caracteres **${ };**
- Para acessar o objeto enviado pelo controller é preciso referenciar a chave que foi adicionada no controller ***(prod)***, como o tipo de objeto enviado é um Produto, precisamos recuperar os seus atributos:

```
<h1>${prod.titulo } foi cadastrado!</h1>
<h2>O preço é: ${prod.preco }</h2>
```

FIAP

O insucesso é apenas uma oportunidade para recomeçar de novo com mais inteligência. Henry Ford