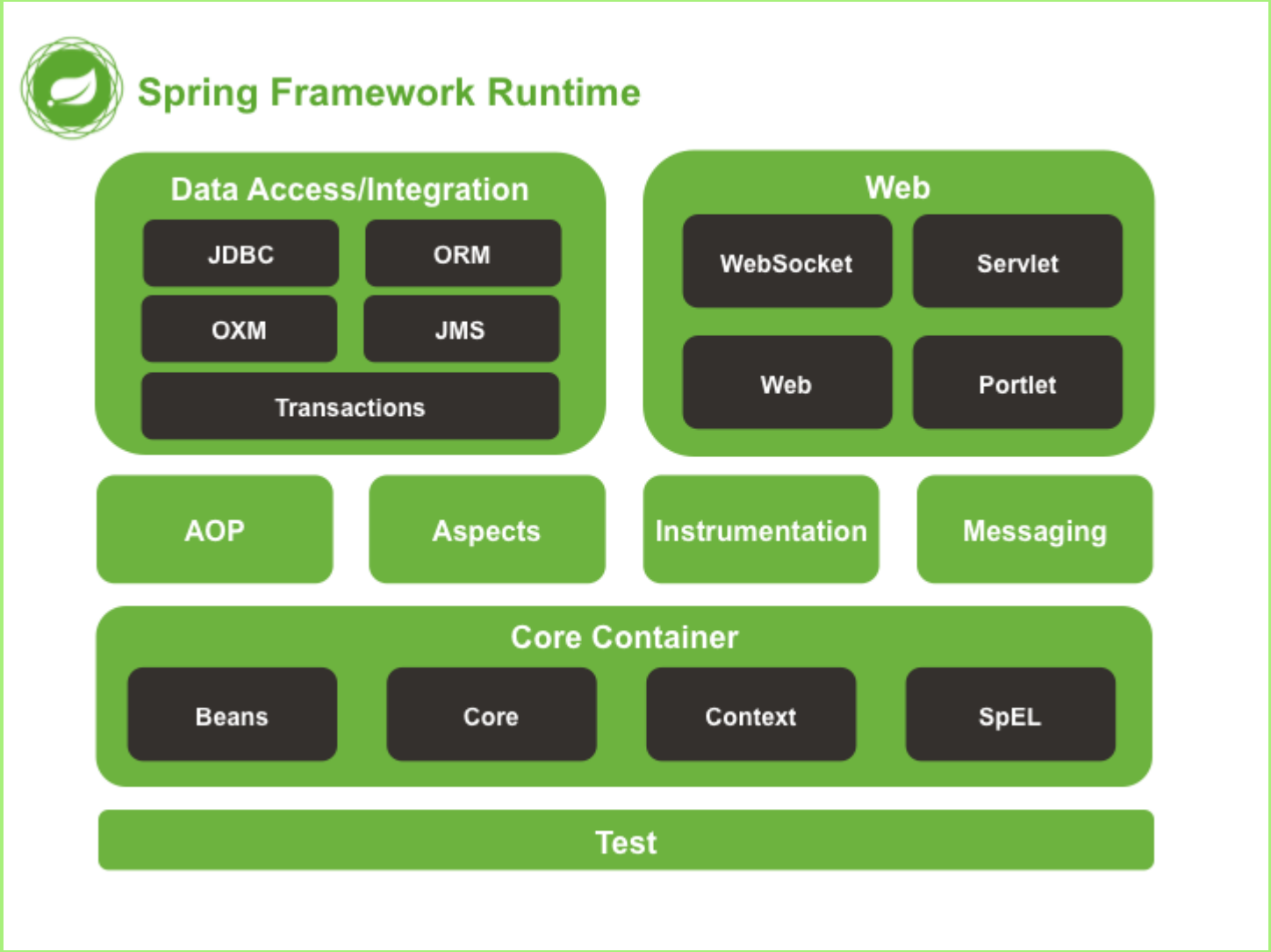
# Spring Framework

## Parte 01 – introdução e primeiros passos

# O que é Spring Framework?

- É um framework de apoio ao desenvolvimento de aplicações corporativas em Java que surgiu como uma alternativa à complexa plataforma J2EE, ganhando extrema popularidade.
- Provê uma série de funcionalidades, dentre as quais destacam-se **desenvolvimento de aplicações web e serviços REST**, **injeção de dependências**, **gerenciamento de transações, suporte a testes automatizados** e **suporte a programação orientada a aspectos**.
- Suporta diversos produtos Java populares tais como JPA, Hibernate, JSF, entre outros.
- Software livre, desenvolvido pela Pivotal.
- Pode ser utilizado em contêineres web, dispensando servidores de aplicações JEE como Glassfish e JBoss. Também pode ser utilizado em aplicações desktop.

# Módulos


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test

# Spring Framework 4 e Java

- Exige JDK 6+.
- Suporta Java 8.
- Suporta uma série de especificações JEE 7 tais como JPA 2.1, JMS 2.0, JTA 1.2 e Bean Validation 1.1.

# Outros projetos Spring

- **Spring MVC** para desenvolvimento de aplicações web (módulo do Spring Framework).
- **Spring Security** para inserção de funcionalidades de autenticação e autorização.
- **Spring Data** para aplicações que usam novas tecnologias de armazenamento de dados como bancos NoSQL e serviços na nuvem.
- **Spring Social** para fácil integração com redes sociais.
- **Spring Web Flow** é uma extensão do Spring MVC para permitir a implementação de fluxos (wizards) de telas.
- **Spring Roo** para desenvolvimento RAD ao estilo Ruby on Rails.
- Além de outros. Visite http://spring.io/projects

# Ferramentas adotadas no curso

- Eclipse Mars for JEE Developers
- Java 8
- Tomcat 8
- H2 database

# Apache Maven

- Um projeto Spring típico possui muitas dependências, fazendo com que o gerenciamento manual seja inviável.
- É comum a utilização do Apache Maven para a criação e/ou gerenciamento do projeto.
- Maven é um automatizador de tarefas capaz de gerenciar as dependências de uma projeto Java.
- As configurações de um projeto são definidas em um arquivo POM (Project Object Model).
- Maven é embutido no Eclipse Mars.

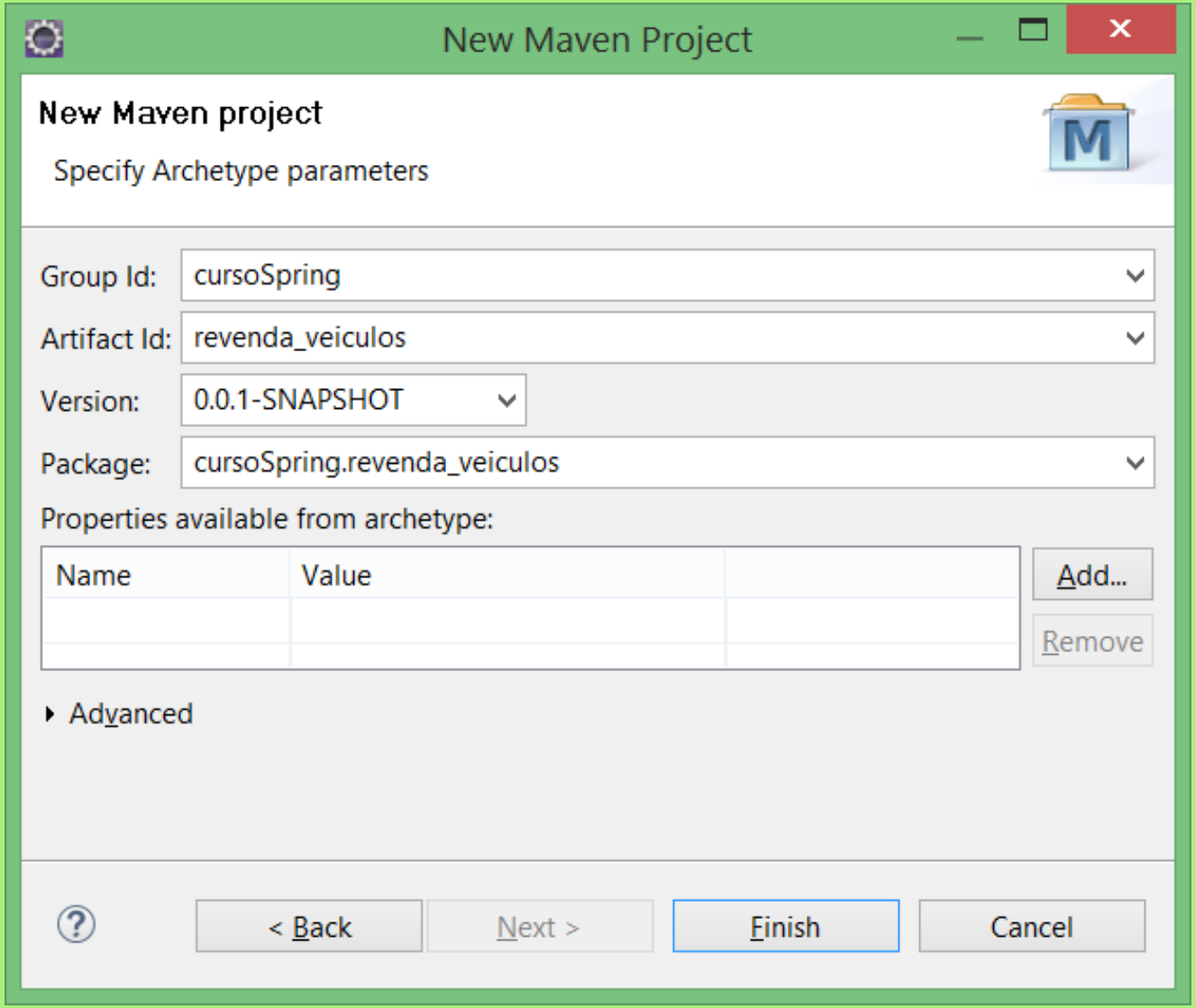
# Configurando o Tomcat

- Adicione o Tomcat ao Eclipse:
  - **Menu Window > Preferences > Server > Runtime Enviroments > Add**
  - Na janela **New Server Runtime Enviroment** Selecione a opção **Apache Tomcat v8.0**
  - Marque a opção **Create a new local server** e em seguida **Next**
  - Na janela **Tomcat Server** indique o diretório de instalação do Tomcat 8 e em seguida **Finish**
- Repare que é criado um projeto **Servers**
- O servidor pode ser controlado pela guia **Servers** (**Window > Show View > Other > Server > Servers**)

# Iniciando um projeto Maven no Eclipse Mars

- Menu **File > New > Other**
- Opção **Maven > Maven Project**
- Desmarque a opção **Create simple project**
- Selecione o archetype **maven-archetype-webapp**
  - *Archetype* é um template de projeto.
- Preencha os campos **Group Id**, **Artifact Id** e **Package**.
  - **Group Id**:identifica a organização/empresa.
  - **Artifact Id**: nome do artefato principal a ser gerado (arquivo JAR ou WAR). Também é o nome do diretório raiz do projeto.
  - **Package**: estrutura de pacotes Java a ser adotada.

# Iniciando um projeto Maven no Eclipse Mars



# Iniciando um projeto Maven no Eclipse Mars

- Vincular o projeto ao runtime do Tomcat (adiciona as bibliotecas de servlets e JSP ao projeto):
  - Em **Propriedades do projeto > Targeted Runtimes**, marcar a opção **Apache Tomcat 8.0** e em seguida **Apply**
- Criar os seguintes diretórios como source folders:
  - **src/main/java**
  - **src/test/java**

# Iniciando um projeto Maven no Eclipse Mars

- Em propriedades do projeto **> Project Facets**:
  - Alterar a versão Java para 1.8
  - Alterar a versão da Servlet API:
    - Desmarcar a opção **Dynamic Web Module** e confirmar em **Apply.**
    - Alterar a versão de **Dynamic Web Module** para 3.1
    - Marcar a opção **Dynamic Web Module**
    - Clicar na opção **Further Configuration Available**. Na tela seguinte, marcar a opção **Generate web.xml deployment descriptor**. e informar **src/main/webapp** para o campo **Content directory**.
  - Clicar em **Apply**.

# Iniciando um projeto Maven no Eclipse Mars

- Abrir o arquivo **src/main/webapp/WEB-INF/web.xml** e verificar a versão da API de servlets. Se estiver diferente de 3.1, excluir o arquivo e em seguida usar a opção **Botão direito no projeto > Java EE Tools > Generate Deployment Descriptor Stub** para gerar um novo web.xml.

# Iniciando um projeto Maven no Eclipse Mars

- Errado:

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
</web-app>
```

- Correto:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                      http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1">
  <display-name>curso</display-name>
```

# Configurando Spring 4

- No arquivo **pom.xml**:
  - Definir Spring IO Platform como gerenciador de dependências adicionando o código abaixo.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.spring.platform</groupId>
      <artifactId>platform-bom</artifactId>
      <version>1.1.3.RELEASE</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

# Configurando Spring 4

- Atualizando as dependências. No **pom.xml**:
  - Remova a dependência do JUnit.
  - Adicione Spring MVC e Commons Logging como dependências.

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
  </dependency>
  <dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
  </dependency>
</dependencies>
```

# Configurando Spring 4

- Criar o diretório **src/main/webapp/WEB-INF/views**
- Crie uma classe com as configurações do Spring MVC.
  - No exemplo: **AppWebConfig**
- Crie uma subclasse de **AbstractAnnotationConfigDispatcherServletInitializer** para integrar Spring MVC com a aplicação web.
  - No exemplo: **SpringMVCServlet**

# Configurando Spring 4

```
package cursoSpring.revenda_veiculos.config;

import org.springframework.context.annotation.Bean;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@EnableWebMvc
public class AppWebConfig {

  @Bean
  public InternalResourceViewResolver internalResourceViewResolver(){
    InternalResourceViewResolver resolver =
                new InternalResourceViewResolver();
    resolver.setPrefix("/WEB-INF/views/");
    resolver.setSuffix(".jsp");
    return resolver;
  }
}
```

# Configurando Spring 4

```
package cursoSpring.revenda_veiculos.config;
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherSer
vletInitializer;

public class SpringMVCServlet extends
AbstractAnnotationConfigDispatcherServletInitializer {

  @Override protected Class<?>[] getRootConfigClasses() {
    return null;
  }

  @Override protected Class<?>[] getServletConfigClasses() {
    return new Class[]{AppWebConfig.class};
  }

  @Override protected String[] getServletMappings() {
    return new String[]{"/"};
  }
}
```

# Olá mundo!

- Passos:
  - Criar a página **olaMundo.jsp** em **src/main/webapp/WEB-INF/views**
  - Criar o controlador web (classe **OlaMundoController**)
  - Atualizar a classe **AppWebConfig** com a anotação **@ComponentScan**

# Olá mundo!

- **olaMundo.jsp**

```
<%@ page language="java"
    contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
  <title>Olá Mundo</title>
</head>
<body>
  <h1>${msg}</h1>
</body>
</html>
```

# Olá mundo!

- **OlaMundoController.java**

```
package cursoSpring.revenda_veiculos.web;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class OlaMundoController {

  @RequestMapping("/olamundo")
  public String olaMundo(Model model){
    model.addAttribute("msg", "Olá Mundo!");
    return "olaMundo";
  }
}
```

# Olá mundo!

- **AppWebConfig.java**

```
...
import org.springframework.context.annotation.ComponentScan;
import cursoSpring.revenda_veiculos.web.OlaMundoController;

@EnableWebMvc
@ComponentScan(basePackageClasses={OlaMundoController.class})
public class AppWebConfig {
  ...
}
```

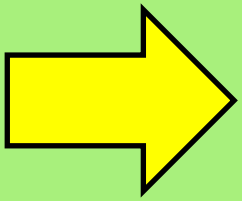
# IoC e injeção de dependências

- Na **inversão de controle** (IoC) o gerenciamento do ciclo de vida dos objetos fica sob responsabilidade de uma infraestrutura de software tal como o contêiner JEE.
- Tipicamente, um objeto possui dependências. Na IoC, esse objeto não cria as dependências mas deve obtê-las de alguma forma. A **injeção de dependências** é um mecanismo que fornece essas dependências.

# IoC e injeção de dependências

- Princípio de Hollywood: não nos ligue, nós te ligamos.

```
class A{
  private B b;
  public A(){
    b = new B();
  }
}
```



```
class A{
  private B b;
  public A(B b){
    this.b = b;
  }
}
```

# IoC e injeção de dependências

- O **contexto do Spring** é o ambiente que realiza a instanciação de objetos (chamados de beans Spring) e a injeção de dependências.
- Anotações são utilizadas para indicar as classes dos beans Springs:
  - **@Component**: bean genérico.
  - **@Service**: bean da camada de serviço.
  - **@Repository**: bean da camada de persistência.
  - **@Controller**: bean que atua como controlador web.
- A anotação **@Autowired** é utilizada para realizar a injeção de dependências.

# IoC: exemplo

- Classe do objeto a ser injetado:

```
package cursoSpring.revenda_veiculos.web;

import org.springframework.stereotype.Component;

@Component
public class Calculador {

  public int calcular(int valor) {
    return valor * 2;
  }

}
```

# IoC: exemplo

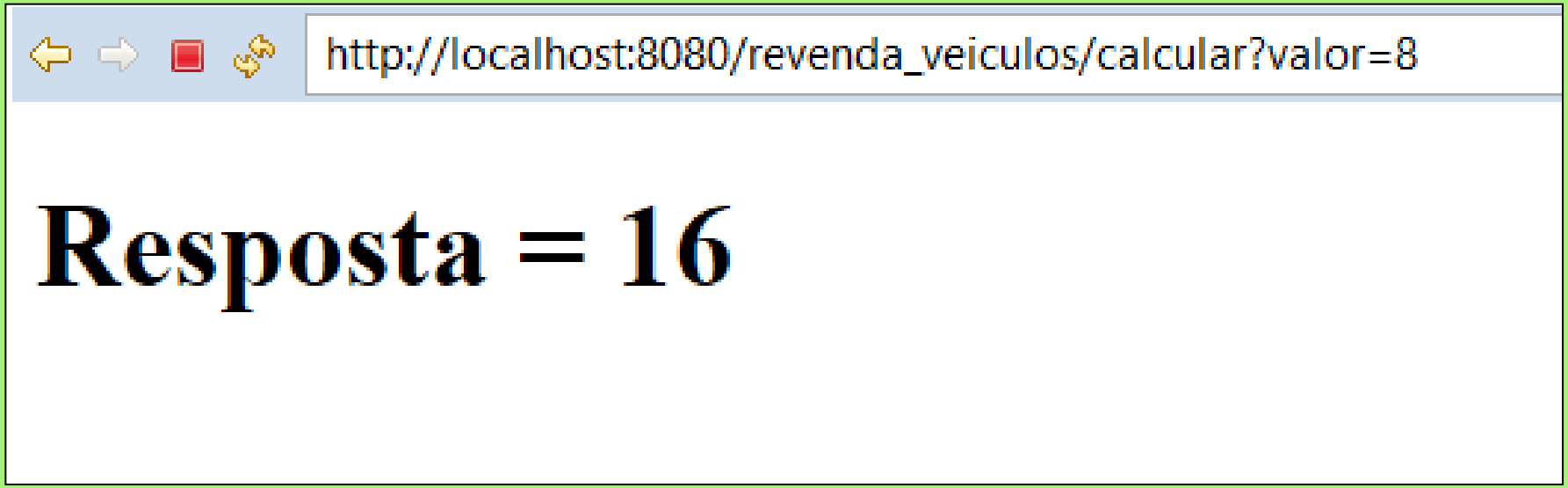
```
package cursoSpring.revenda_veiculos.web;

import org.springframework.beans.factory.annotation.Autowired;
...
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class CalculadorController {

  @Autowired private Calculador calculador;

  @RequestMapping("/calcular")
  public String olaMundo(@RequestParam(required=true) Integer
valor, Model model){
    model.addAttribute("resposta", calculador.calcular(valor));
    return "respostaCalculador";
  }
}
```

# IoC: exemplo

- **respostaCalculador.jsp**

```
<%@ page language="java"
        contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
  <title>Resposta Calculador</title>
</head>
<body>
  <h1>Resposta = ${resposta}</h1>
</body>
</html>
```

# IoC: exemplo

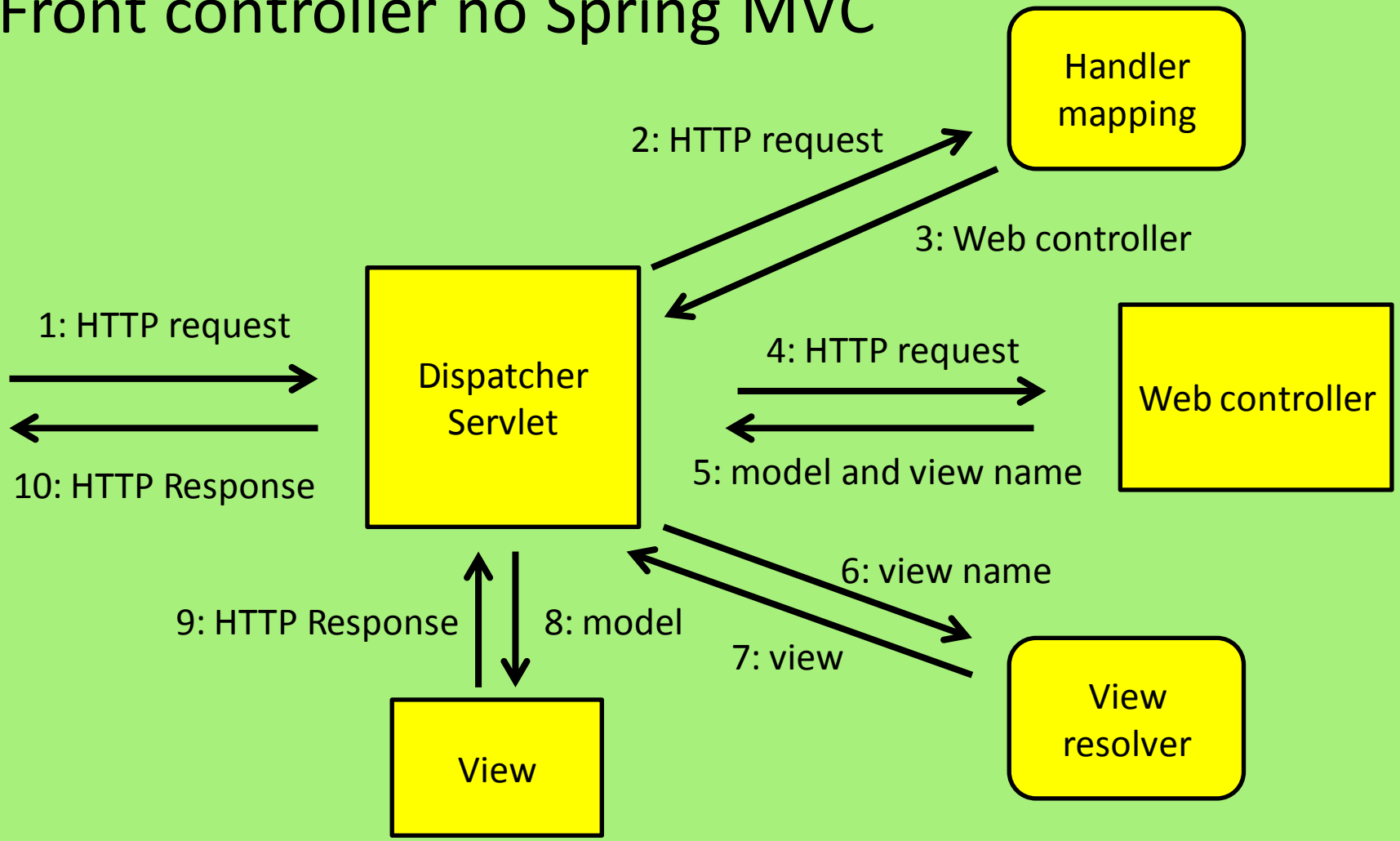

http://localhost:8080/revenda_veiculos/calcular?valor=8
Resposta = 16

# Entendendo as configurações

- Spring MVC adota o padrão front controller. **DispatcherServlet** é o objeto que captura as requisições HTTP e, com base nas URLs, executa os métodos definidos nos controladores web (classes marcadas com **@Controller**).
- Devido à herança de **AbstractAnnotationConfigDispatcherServletInitializer**, **DispatcherServlet** utilizará as configurações definidas em nossa classe **SpringMVCServlet**.

# Entendendo as configurações

- Front controller no Spring MVC



# Entendendo as configurações

```
...
public class SpringMVCServlet extends
AbstractAnnotationConfigDispatcherServletInitializer {

  @Override protected Class<?>[] getRootConfigClasses() {
    return null;
  }

  @Override protected Class<?>[] getServletConfigClasses() {
    return new Class[]{AppWebConfig.class};
  }

  @Override protected String[] getServletMappings() {
    return new String[]{"/"};
  }
}
```

# Entendendo as configurações

```
...
public class SpringMVCServlet extends
AbstractAnnotationConfigDispatcherSer

  @Override protected Class<?>[] getR
    return null;
  }

  @Override protected Class<?>[] getServletConfigClasses() {
    return new Class[]{AppWebConfig.class};
  }

  @Override protected String[] getServletMappings() {
    return new String[]{"/"};
  }
}
```

O método **getServletMappings** retorna os padrões de URL que serão tratados pelo DispatcherServlet.

# Entendendo as configurações

```
...
public class SpringMVCServlet extend
AbstractAnnotationConfigDispatcherSe

  @Override protected Class<?>[] get
    return null;
  }

  @Override protected Class<?>[] getServletConfigClasses() {
    return new Class[]{AppWebConfig.class};
  }

  @Override protected String[] getServletMappings() {
    return new String[]{"/"};
  }
}
```

O método **getServletConfigClasses** retorna a relação de classes que contém as configurações do Spring MVC.

# Entendendo as configurações

```
...

@EnableWebMvc
@ComponentScan(basePackageClasses={OlaMundoController.class})
public class AppWebConfig {

  @Bean
  public InternalResourceViewResolver internalResourceViewResolver(){
    InternalResourceViewResolver resolver =
                new InternalResourceViewResolver();
    resolver.setPrefix("/WEB-INF/views");
    resolver.setSuffix(".jsp");
    return resolver;
  }
}
```

# Entendendo as configurações

```
...

@EnableWebMvc
@ComponentScan(basePackageClasses={OlaMundoController.class})
public class AppWebConfig {

  @Bean
  public InternalResourceViewResolver internalResourceViewResolver(){
    InternalResourceViewReso
                new Interna
    resolver.setPrefix("/WEE
    resolver.setSuffix(".jsp
    return resolver;
  }
}
```

Esta classe contém as configurações para o Spring MVC. **@EnableWebMvc** habilita uma série de funcionalidades tais como serialização JSON e geração de RSS. **@ComponentScan** indica os pacotes que contém classes anotadas como beans Spring.

# Entendendo as configurações

Além de JSP, Spring MVC suporta outras tecnologias de páginas dinâmicas tais como FreeMarker e GroovyMarkup. Um view resolver é um objeto capaz de renderizar páginas em uma determinada tecnologia. Este método define um view resolver apropriado para páginas JSP. As chamadas **resolver.setPrefix** e **resolver.setSuffix** definem o que será respectivamente utilizado como prefixo e sufixo da string de visão retornada por um método de um controlador web. A anotação **@Bean** indica que o objeto retornado pelo método deve ser tratado gerenciado como um bean Spring.

```
@
@ComponentScan(basePackageClasses={OlaMundoController.class})
public class AppWebConfig {

  @Bean
  public InternalResourceViewResolver internalResourceViewResolver(){
    InternalResourceViewResolver resolver =
                 new InternalResourceViewResolver();
    resolver.setPrefix("/WEB-INF/views");
    resolver.setSuffix(".jsp");
    return resolver;
  }
}
```

# Referências


- Apache Foundation. **Apache Maven Project**. Disponível em <https://maven.apache.org/index.html>.
- Çaliskan, Mert e Sevindik, Kenan. **Beginning Spring**. Wrox, Indianapolis: 2015.
- JavaHash. **Spring 4 MVC Hello World Tutorial – Full Example**. Disponível em <http://javahash.com/spring-4-mvc-hello-world-tutorial-full-example/>
- Johnson, Rod et al. **Spring Framework Reference Documentation**, 4.2.1 release. Disponível em <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/>

# Referências


- Kainulainen, Petri. **Spring Data JPA Tutorial:** Getting the Required Dependencies. Disponível em <http://www.petrikainulainen.net/programming/spring-framework/spring-data-jpa-tutorial-getting-the-required-dependencies/>
- Souza, Alberto. **Spring MVC:** domine o principal framework web Java. São Paulo: Casa do Código, 2015.
- Wikipédia. **Inversão de controle**. Disponível em <https://pt.wikipedia.org/wiki/Invers%C3%A3o_de_controle>
- Wilkison, Andy. **Spring IO Platform Reference Guide**, 1.1.3 release. Disponível em <http://docs.spring.io/platform/docs/current/reference/htmlsingle/>

**Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte**

**Campus Natal Central**

**Diretoria Acadêmica de Gestão e Tecnologia da Informação**

**Curso de formação em Spring Framework 4**

Parte 1: introdução e primeiros passos.

Autor: Alexandre Gomes de Lima

Natal, outubro de 2015.