Tema 3. Tecnologías del servidor. Introducción: Java en el lado servidor

Programación web

Boni García

Curso 2016/2017

Índice

Índice

Maven

Introducción

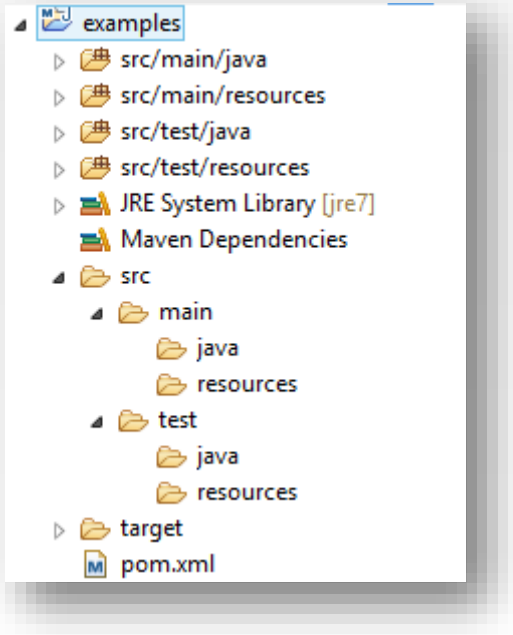
- Maven es una herramienta que permite la gestión del ciclo de vida de proyectos **Java**
- Automatiza tareas como la gestión de dependencias, compilación, ejecución, pruebas, despliegue, etc.
- Software libre (licencia Apache 2.0) desarrollado por la fundación Apache
- Versión estable (marzo 2018): 3.5.3
- Existen otras herramientas similares: Ant, Gradle

https://maven.apache.org/

Maven

Estructura de un proyecto Maven

- Maven sigue el principio ágil de "*convención sobre configuración*"
- Un proyecto Maven tiene una estructura de carpetas determinada



- `src/main/java`: clases Java
- `src/main/resources`: recursos (≠ clases Java)
- `src/test/java`: tests Java
- `src/tets/resources`: recursos para tests

Maven

Estructura de un proyecto Maven

- Además, un proyecto Maven debe incluir en su raíz un fichero **`pom.xml`** (*project object model*). Ejemplo:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.u-tad.web</groupId>
   <artifactId>examples</artifactId>
   <version>0.0.1-SNAPSHOT</version>

   <dependencies>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>4.12</version>
         <scope>test</scope>
      </dependency>
   </dependencies>
</project>
```

El `groupId`, `artifactId` y `version` forman las **coordenadas** del proyecto. Deben identificar unívocamente dicho proyecto.

El prefijo `-SNAPSHOT` en la versión se suele emplear para identificar la versión de desarrollo

Maven

Gestión de dependencias

- Maven se encarga de descargar la dependencias de forma automática, simplemente declarando dichas dependencias en la sección `dependencies` del `pom.xml`
- Por defecto las dependencias se descargan del repositorio central de Maven: http://search.maven.org/
- Se descargan en el repositorio local en nuestro sistema ubicado en `~/.m2/repository`
- Las dependencias de las dependencias directas se conocen como dependencias transitivas, y son resueltas automáticamente por Maven

Maven

Gestión de dependencias

- Los principales tipos de ámbitos en Maven son los siguientes:
  - importantes
    - `<scope>compile</scope>`: Dependencia visible para todos las clases del proyecto (dentro de `main` y de `test`). Opción por defecto (no es necesario ponerla explícitamente)
    - `<scope>test</scope>`: Dependencia visible sólo para los tests (dentro de carpeta `test`)
  - `<scope>provided</scope>`: Dependencia necesaria en tiempo de compilación pero no en tiempo de ejecución
  - `<scope>runtime</scope>`: Dependencia no necesaria en tiempo de compilación pero sí en tiempo de ejecución

Maven

Gestión del ciclo de vida

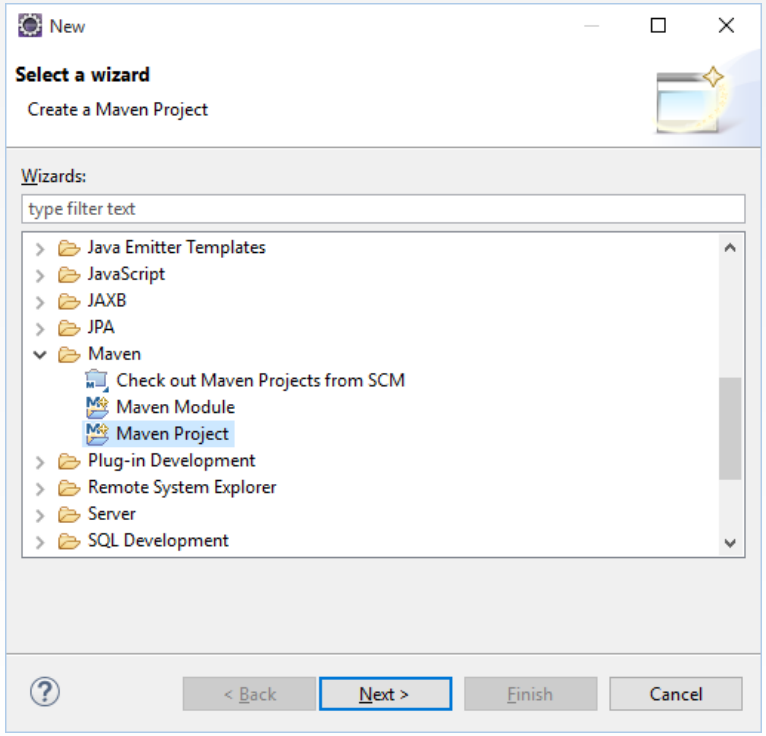
- Maven se puede usar desde la consola de comandos o bien integrado en un entorno de desarrollo (por ejemplo Eclipse)
- Las principales fases del ciclo de vida manejadas desde la consola son:
  - `mvn compile`: Compila el código fuente Java
  - `mvn test`: Ejecuta las pruebas unitarias (clases `*Test.java`)
  - `mvn package`: Empaqueta el proyecto (típicamente como JAR o WAR)
  - `mvn install`: Copia el proyecto empaquetado en el repositorio local

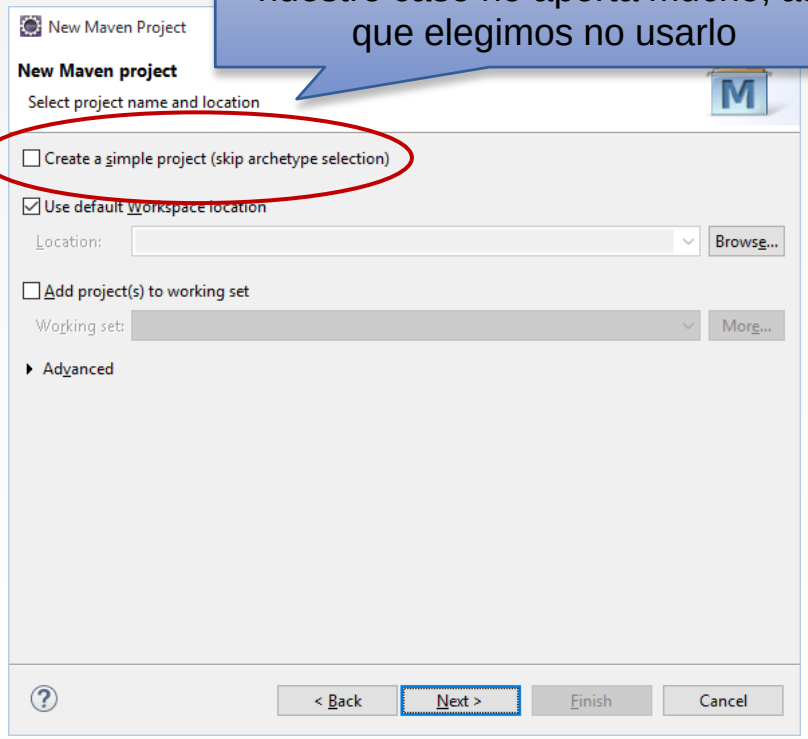
- Maven genera los artefactos en la carpeta `target` en la raíz del proyecto. Para borrar esos artefactos usamos el comando `mvn clean`

Maven

Creación de un proyecto Maven desde Eclipse

- File → New → Other → Maven Project

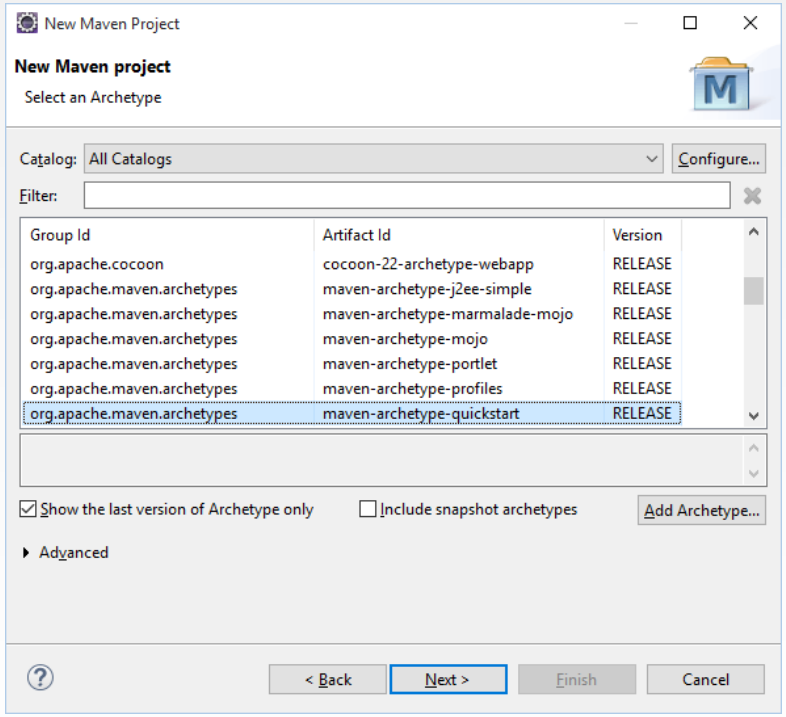




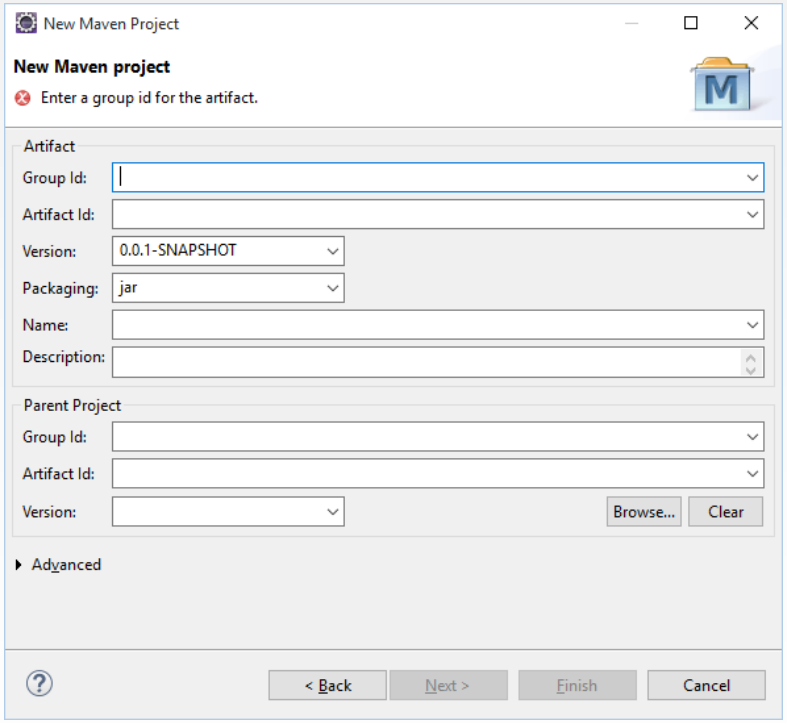
Un **arquetipo** es una plantilla para un determinado tipo de proyecto. En nuestro caso no aporta mucho, así que elegimos no usarlo

Maven

Creación de un proyecto Maven desde Eclipse



Con arquetipo

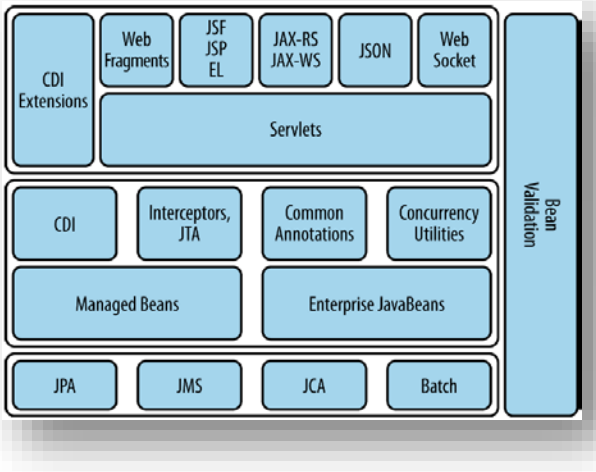


Sin arquetipo

Java EE

Introducción

- **Java Enterprise Edition** (EE) es la extensión de Java Standard Edition (SE) que permite el desarrollo de aplicaciones web "empresariales"
- La arquitectura de estas aplicaciones se divide en capas:
  - Capa de presentación a la información
  - Capa de lógica del negocio (requisitos funcionales)
  - Capa de acceso a base de datos
  - Capa de integración con sistemas externos
- Java EE es un conjunto de **especificaciones** JSR (*Java Specification Requests*)



Java EE

Servidor de aplicaciones

- Un **servidor de aplicaciones** consiste en un servidor web (HTTP) que además permite la ejecución de aplicaciones en el lado servidor
- En el mundo Java también se conoce a los servidores de aplicaciones como contenedores (*containers*)
- Algunos ejemplos de contenedores en Java son:
  - Contenedores Java EE: Glassfish, WildFly (anteriormente JBoss)
  - Contenedores web: Apache Tomcat, Jetty



Java EE

Empaquetado de aplicaciones Java

- Una aplicación Java SE se empaqueta en un fichero JAR (*Java archive*)
  - Un JAR es un fichero comprimido que contiene clases Java compilados (bytecodes, ficheros .class) y otros recursos
- Una aplicación web Java EE se empaqueta en un fichero WAR (*Web application archive*)
  - Un WAR es también un fichero comprimido que contiene los componentes Java activos en el servidor y la aplicación web (ficheros HTML, CSS, JavaScript, imágenes, etc.)

Spring

Introducción

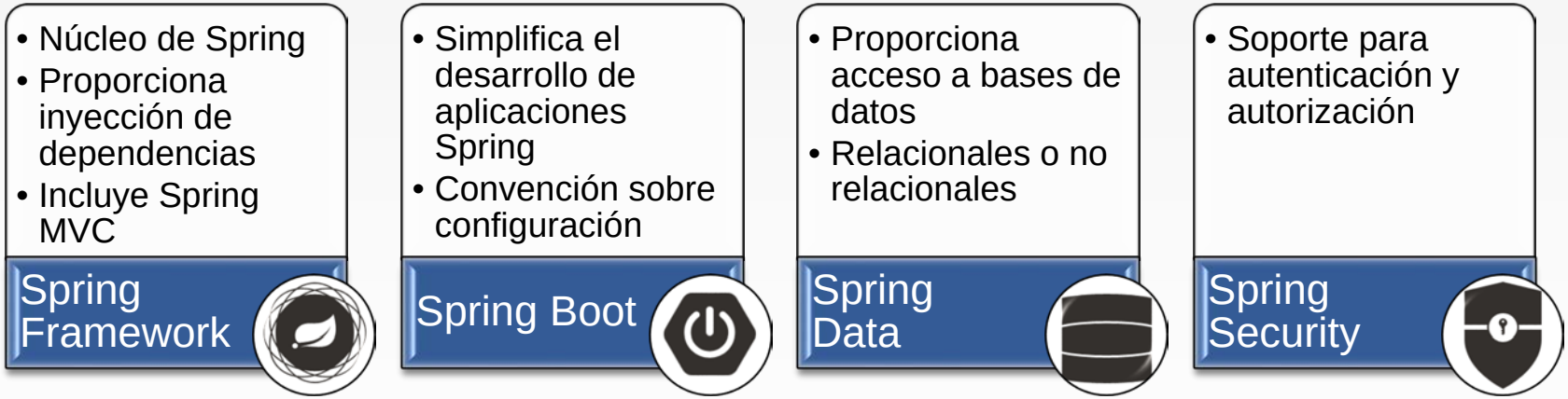
- Spring es un **framework** *open source* (licencia Apache 2.0) de desarrollo de aplicaciones empresariales basado en tecnologías Java
- El objetivo fundamental de Spring es **simplificar** el desarrollo de aplicaciones Java
- La primera versión fue escrita por Rod Johnson y descrita en su libro *Expert One-on-One J2EE Design and Development* (octubre 2002) y surgió como alternativa a Java EE (conocido como J2EE en esa época)
- La versión estable (marzo 2018) es la 5.0.5

http://spring.io/

Spring

Introducción

- Spring tiene una estructura modular, dividida en proyectos
- Algunos de los módulos Spring más significativos son:



https://spring.io/projects

Spring

Inyección de dependencias

- El patrón arquitectónico fundamental en el que se basan las aplicaciones Spring es la **inyección de dependencias**
- La idea consiste en crear aplicaciones en base a **componentes** (llamados *beans* en Spring) que son reutilizados en otros componentes
- De esta forma se promueve la modularidad, el bajo acoplamiento y las pruebas (*testability*)
- La gestión de estos componentes la realiza Spring en lo que se conoce como **contexto de aplicación**:
  - Creación de componentes
  - Inyección de componentes

Spring

Inyección de dependencias

- En las versiones iniciales de Spring, los beans se definían en XML
- A partir de Spring 2.5 se introdujo el uso de anotaciones Java para la definición de componentes, siendo la opción recomendada hoy día
- Las anotaciones básicas para realizar la inyección de dependencias son:
  - `@Component`: Permite definir un componentes (clase a Java)
  - `@Autowired`: Permite definir el uso (inyección de dependencias) de un componente en otro

Spring

Inyección de dependencias

- Ejemplo:


```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MessageComponent {

    @Autowired
    private MessageService messageService;

    public String getMessage() {
        return messageService.getMessage();
    }
}
```

```java
import org.springframework.stereotype.Component;

@Component
public class MessageService {

    public String getMessage() {
        return "Hello world!";
    }

}
```

Inyección por propiedades de clase (*field injection*)

Spring

Inyección de dependencias

- Ejemplo:


```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MessageComponent {

    private MessageService messageService;

    @Autowired
    public MessageComponent(MessageService messageService) {
        this.messageService = messageService;
    }

    public String getMessage() {
        return messageService.getMessage();
    }

}
```

Inyección por constructor (*constructor injection*). El uso de `@Autowired` es opcional a partir de Spring 4.3+

```java
import org.springframework.stereotype.Component;

@Component
public class MessageService {

    public String getMessage() {
        return "Hello world!";
    }

}
```

Spring

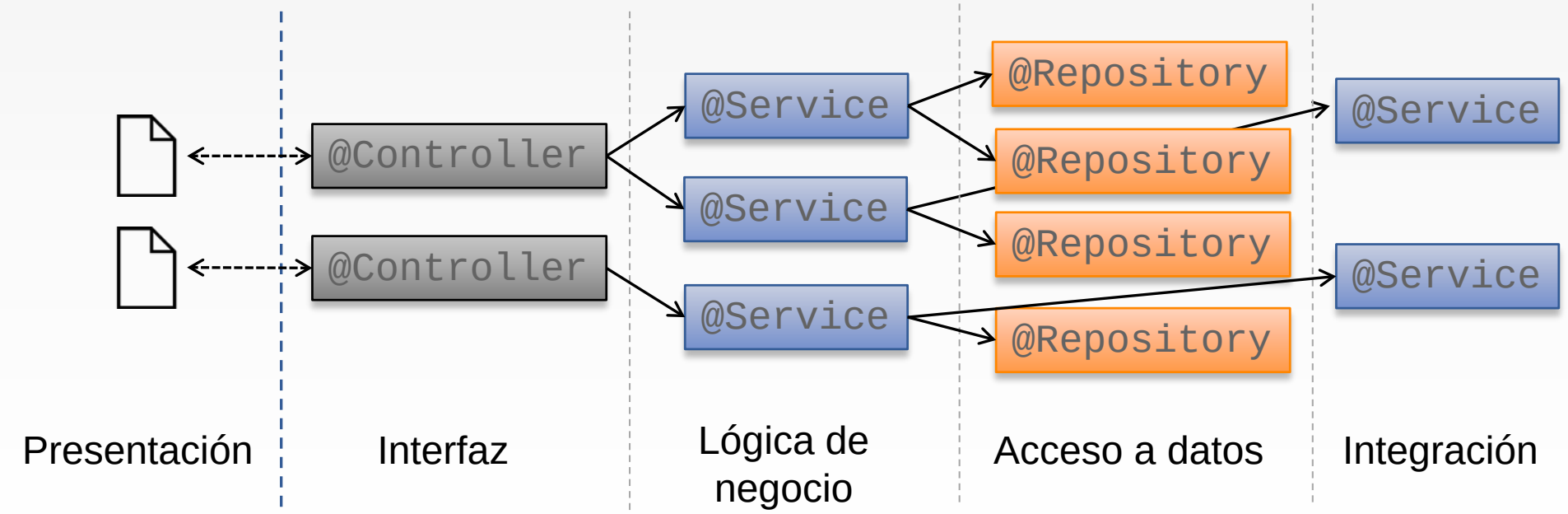
Inyección de dependencias

- Existen otras anotaciones en Spring para crear componentes:
  - `@Controller`: Permite definir un componentes que actuarán de interfaz con la capa de presentación de aplicaciones web
  - `@Repository`: Permite definir un componentes persistentes en base de datos
  - `@Service`: Se comportan igual que los componentes definidos con `@Component` pero se suelen usar en la capa de servicio
  - `@Bean`: Permite definir un componentes pasándole parámetros al constructor

Spring

Inyección de dependencias

- Al igual que en las aplicaciones Java EE, la arquitectura de las aplicaciones web basadas en Spring estará divida en **capas**



Spring Boot

Introducción

- Con el paso de los años, Spring se ha vuelto cada vez más complejo
- Spring Boot es un proyecto de reciente creación que permite simplificar la creación de aplicaciones Spring usando el principio de convección sobre configuración
- El uso de Spring Boot en aplicaciones web es especialmente interesante ya que incorpora un servidor de aplicaciones **Tomcat** embebido junto a la propia aplicación
- La versión estable de Spring Boot (marzo 2018) es la 2.0.0 (está basada en Spring 5)

https://projects.spring.io/spring-boot/

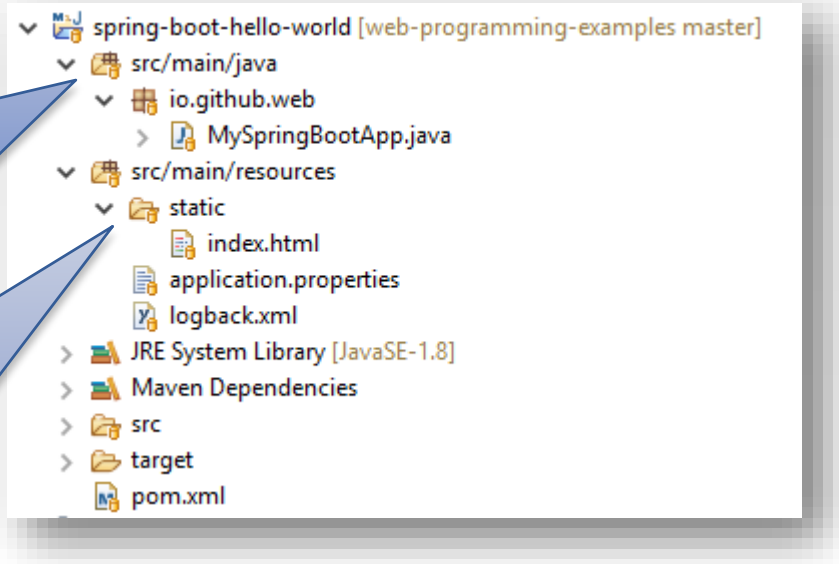
Spring Boot

Hello World

- Vamos a ver un ejemplo simple de aplicación web basada en Spring Boot (*hello world*)

Los componentes activos del lado servidor de nuestra aplicación web serán beans de Spring

Por convención, los recursos estáticos de nuestra aplicación web (HTML, CSS, JavaScript) están alojados en la carpeta `static`



Spring Boot

Hello World

- Empaquetado como JAR
- Declaramos un proyecto padre a nivel Maven
- Java 8 y UTF-8
- Dependencia de aplicación web
- Plugin de Spring Boot (`mvn spring-boot:run`)

pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>io.github.web</groupId>
   <artifactId>hello-world</artifactId>
   <version>1.0.0</version>
   <packaging>jar</packaging>

   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.0.0.RELEASE</version>
   </parent>

   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <java.version>1.8</java.version>
   </properties>

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

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

Spring Boot

Hello World

Clase principal. Al ejecutar esta clase se arrancará el servidor Tomcat embebido y se desplegará la aplicación en dicho servidor

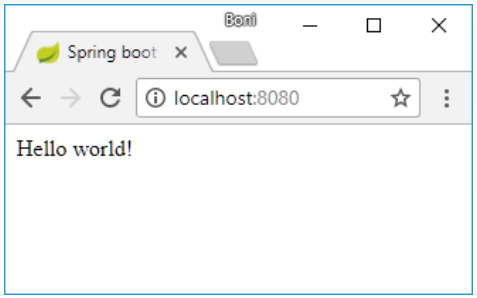
MySpringBootApp.java

```java
package io.github.web;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MySpringBootApp {

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

Nuestra aplicación publica una única página web estática. La página `index.html` es el recurso por defecto de la aplicación

index.html

```html
<!DOCTYPE html>
<html>
<head>
<title>Spring boot - hello world</title>
</head>

<body>Hello world!
</body>
</html>
```



Spring Boot

Hello World

application.properties

```
server.port=8080
```

Fichero de configuración de Spring Boot

logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
   <include resource="org/springframework/boot/logging/logback/base.xml" />

   <logger name="org.springframework.web" level="INFO" />
</configuration>
```

Fichero de configuración para trazas (*logging*)

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.0.RELEASE)

2018-03-13 23:57:58.081  INFO 8980 --- [           main] io.github.web.MySpringBootApp            : Starting
MySpringBootApp on LAPTOP-T9O4060I with PID 8980 (D:\projects\web-programming-examples\spring-boot-hello-
world\target\classes started by boni in D:\projects\web-programming-examples\spring-boot-hello-world)
2018-03-13 23:57:58.085  INFO 8980 --- [           main] io.github.web.MySpringBootApp            : No active profile
set, falling back to default profiles: default
2018-03-13 23:57:58.182  INFO 8980 --- [           main] ConfigServletWebServerApplicationContext : Refreshing
org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext@3cc2931c: startup
date [Tue Mar 13 23:57:58 CET 2018]; root of context hierarchy
2018-03-13 23:58:00.657  INFO 8980 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat
initialized with port(s): 8080 (http)
...
2018-03-13 23:58:02.780  INFO 8980 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : FrameworkServlet
'dispatcherServlet': initialization completed in 33 ms
```