

INGENIERÍA WEB Y COMPUTACIÓN EN LA NUBE

# Bloque3: Parte servidora (backend)

TEMA 3.1: SPRING Y JAVA EE

Boni García

[boni.garcia@urjc.es](mailto:boni.garcia@urjc.es)



# Índice de contenidos

1. Java Enterprise Edition
2. Maven
3. Spring

# Índice de contenidos

## 1. Java Enterprise Edition

- Introducción
- Evolución de Java EE
- Arquitectura de Java EE 7
- Servidor de aplicaciones
- Empaquetado de aplicaciones Java EE

## 2. Maven

## 3. Spring

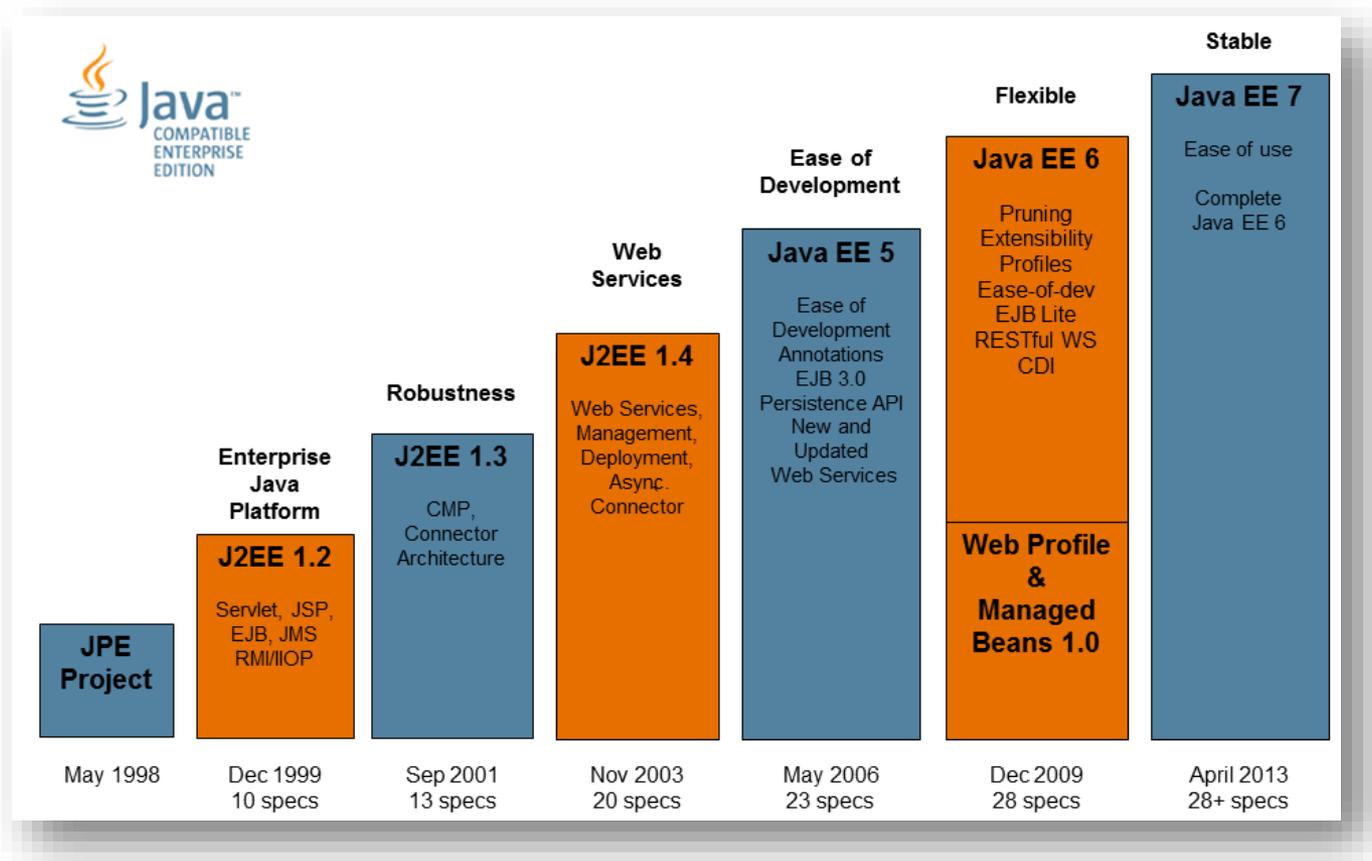
# 1. Java Enterprise Edition

## Introducción

- Java Enterprise Edition apareció a finales de los 90 para el desarrollo de aplicaciones **empresariales**
  - Necesita acceso a datos
  - Aplica la lógica del negocio
  - Añade capas de presentación a la información
  - Se comunica con sistemas externos
- Actualmente, las aplicaciones empresariales más extendidas son las aplicaciones **web**

## 1. Java Enterprise Edition

### Evolución de Java EE



## 1. Java Enterprise Edition

### Evolución de Java EE

- Características comunes J2EE 1.x:
  - Resultó ser una tecnología complicada con tiempo de desarrollo largo
  - Por esta razón aparecieron tecnologías como Spring, con el objetivo de permitir un desarrollo mucho más ligero
- A partir de Java EE 5 se simplificó el modelo de desarrollo:
  - Anotaciones en el código (en vez de ficheros de configuración XML)
  - Programación con objetos simples (*Plain Old Java Objects*, **POJOs**)
  - Convención sobre configuración: Lo que no se especifica, se considera que tiene el valor por defecto (que suele ser razonable)
  - Java EE puede ser usada de forma completa o junto con el framework **Spring**
  - Más información: <http://docs.oracle.com/javaee/7/tutorial/>

# 1. Java Enterprise Edition

## Arquitectura de Java EE 7

- Java EE es un conjunto de **especificaciones** de librerías que cualquier empresa o comunidad de software libre pueden implementar
- Las especificaciones de Java EE se definen en una organización llamada Java Community Process (**JCP**) creada en 1998
- Cada especificación se denomina Java Specification Requests (**JSR**)
  - Por ejemplo, JSR 317 corresponde con JPA 2.0
- Existen JSR *umbrella* (paraguas) que agrupan diferentes JSRs
  - Por ejemplo, el JSR 316 corresponde con Java EE 6

# 1. Java Enterprise Edition

## Arquitectura de Java EE 7

- Java EE 7 es una especificación *umbrella* (paraguas) definida por el JSR 342 que contiene 38 especificaciones, por ejemplo:
  - Java Servlet 3.1 (JSR340)
  - Java Server Pages 2.3 (JSR245)
  - Java API for JSON Processing (JSR353)
  - Java Persistence API 2.1 JPA (JSR 338)
  - ...

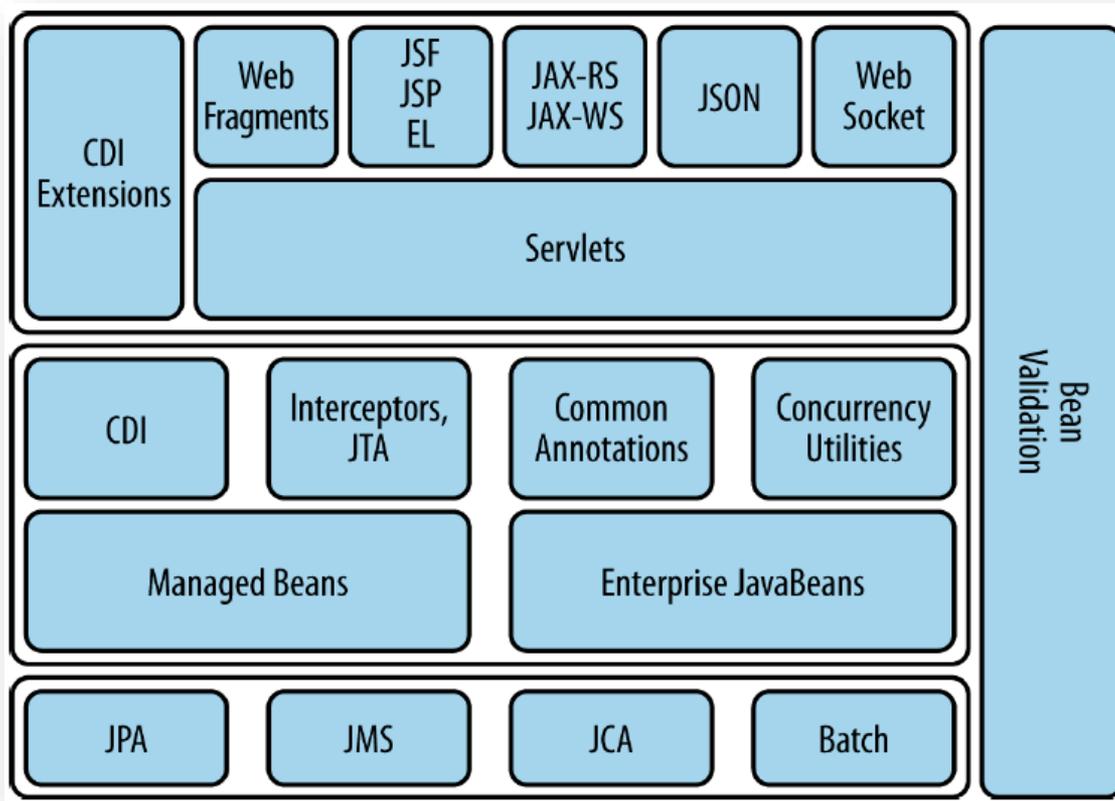
# 1. Java Enterprise Edition

## Arquitectura de Java EE 7

- Para reducir el tamaño de Java EE se han creado los **perfiles** (*profiles*) que implementan un subconjunto de la especificación completa Java EE
- El único perfil definido es el Web Profile con las especificaciones necesarias para el desarrollo de aplicaciones web
  - JSP, Servlets, JSF, JPA, EJB Lite, etc...

# 1. Java Enterprise Edition

## Arquitectura de Java EE 7



# 1. Java Enterprise Edition

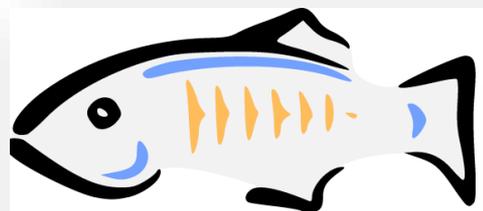
## Servidor de aplicaciones

- Un servidor de aplicaciones es un framework que proporciona infraestructura para el despliegue, ejecución y gestión de aplicaciones
- Típicamente consiste en un servidor web (HTTP) que ejecuta aplicaciones dinámicas en el lado servidor
- En el mundo Java también se conoce a los servidores de aplicaciones como contenedores (*containers*)
  - Como Java EE es un superconjunto de Java SE, cualquier aplicación Java EE puede usar la API de Java SE

# 1. Java Enterprise Edition

## Servidor de aplicaciones

- **Contenedores Java EE:** cumple con la especificación Java EE completa (o al menos el perfil Web)



Glassfish 4.0  
(Java EE 7)



WildFly 8.0  
(Java EE 7)

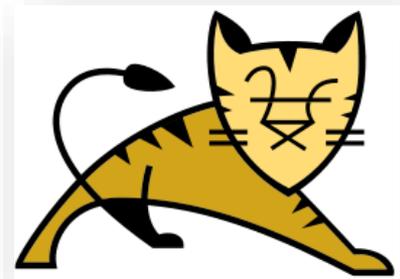


TomEE 1.6  
(Java EE 6 Web Profile)

# 1. Java Enterprise Edition

## Servidor de aplicaciones

- **Contenedores web:** ofrecen la APIs de Servlets y JSPs. Se le pueden añadir otras librerías Java EE complementarias (excepto EJB)



Apache Tomcat 8  
(Servlets 3.1 y JSPs 2.3)



Eclipse Jetty 9.1  
(Servlets 3.1 y JSPs 2.3)

# 1. Java Enterprise Edition

## Empaquetado de aplicaciones Java EE

- 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 (servlets, etc) y la aplicación web (HTML, CSS, Java-Scripts, imágenes, etc)

# Índice de contenidos

## 1. Java Enterprise Edition

## 2. Maven

- Introducción
- Gestión de dependencias
- Gestión del ciclo de vida
- Ejemplo: creación de un proyecto Maven desde Eclipse
- Ejemplo: uso de log4j en una aplicación Java

## 3. Spring

# 2. Maven

## Introducción

- Maven es una herramienta que permite la automatización del ciclo de vida de aplicaciones Java
  - Compilación, ejecución, pruebas, despliegue, gestión de dependencias...
- Desarrollado por la fundación Apache
- Software libre (licencia Apache 2.0)
- Versión actual estable (octubre 2015): 3.3.3
- Otras herramientas similares: Ant, Gradle



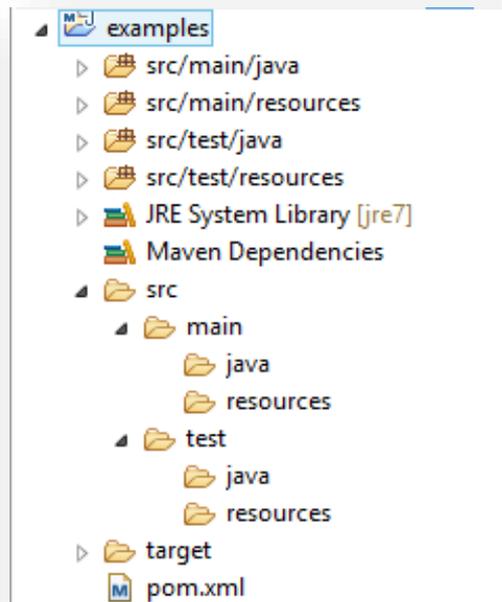
<https://maven.apache.org/>

# SPRING Y JAVA EE

## 2. Maven

### 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/test/resources`: recursos para tests

# SPRING Y JAVA EE

## 2. Maven

### Introducción

- Un proyecto Maven debe incluir en su raíz un fichero `pom.xml` (*project object model*). La estructura básica es como sigue:

```
<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>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

## 2. Maven

### Gestión de dependencias

- Maven se encarga de descargar las dependencias simplemente añadiendo la dependencia 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`

# 2. Maven

## Gestión de dependencias

- Las dependencias pueden tener los siguientes ámbitos:

principales

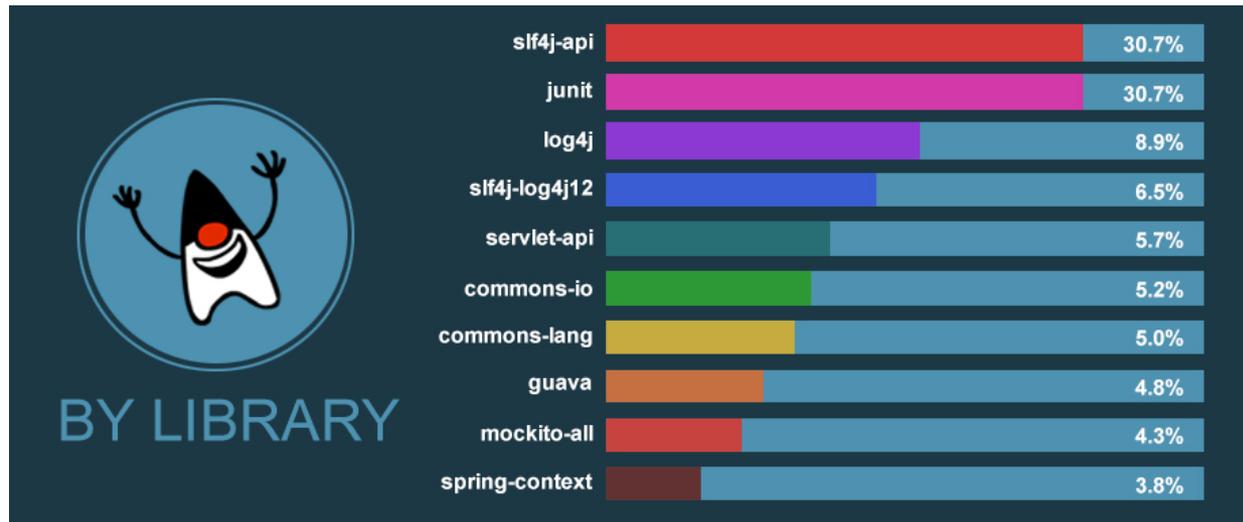
- `<scope>compile</scope>`: Dependencia visible para todas las clases del proyecto (dentro de `main` y de `test`). Opción por defecto (no es necesario ponerla explícitamente)
- `<scope>test</scope>`: Dependencia visible sólo para los tests (dentro de carpeta `test`)
- `<scope>provided</scope>`: Dependencia necesaria en tiempo de compilación pero no en runtime
- `<scope>runtime</scope>`: Dependencia no necesaria en tiempo de compilación pero sí en runtime
- `<scope>system</scope>`: La dependencia está fuera del repositorio local Maven (en la ruta indicada por `<systemPath>/path-to/lib.jar</systemPath>`)

# SPRING Y JAVA EE

## 2. Maven

### Gestión de dependencias

- Las dependencias Maven más usadas en proyectos Java de GitHub son:

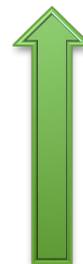


<http://blog.takipi.com/githubs-10000-most-popular-java-projects-here-are-the-top-libraries-they-use/>

## 2. 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 por Maven son:
  - `mvn compile`: Compila el código fuente Java
  - `mvn test`: Ejecuta las pruebas unitarias
  - `mvn package`: Empaqueta los binarios (JAR, WAR)
  - `mvn install`: Instala el binario 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`

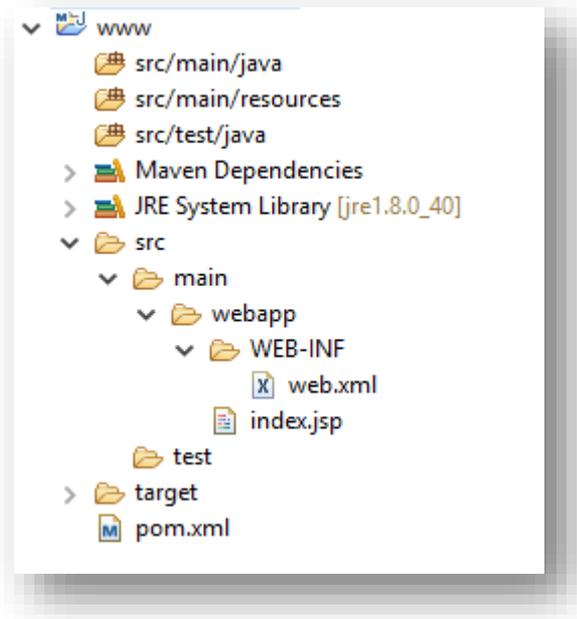


# SPRING Y JAVA EE

## 2. Maven

### Arquetipos

- Un **arquetipo** es una plantilla para un determinado tipo de proyecto Maven
- Ejemplo de proyecto nuevo creado con el arquetipo `maven-archetype-webapp`

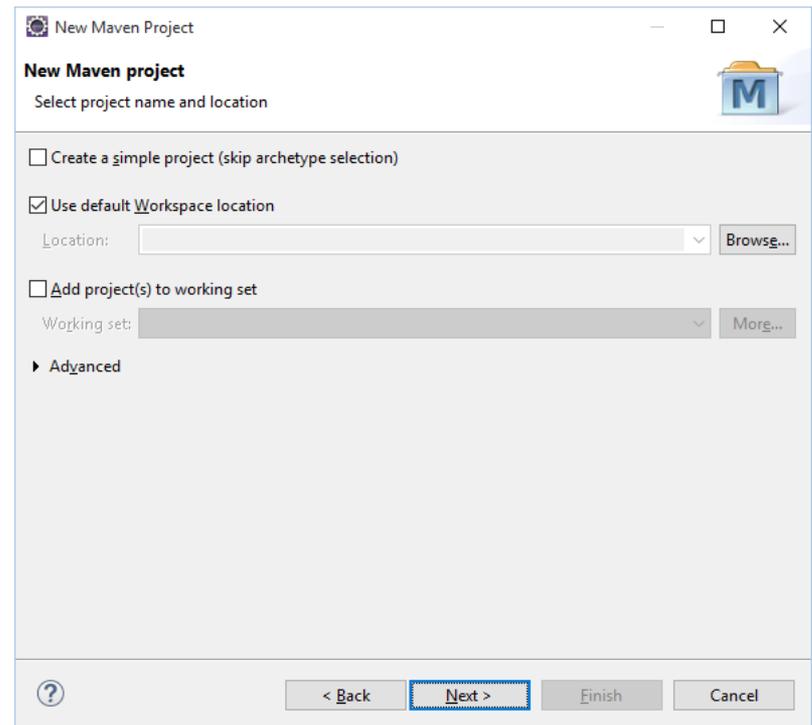
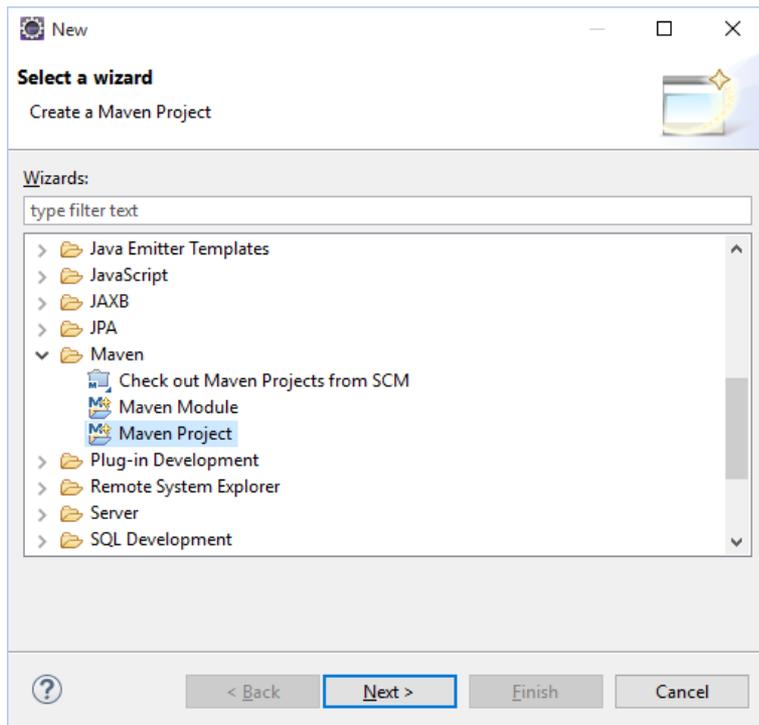


# SPRING Y JAVA EE

## 2. Maven

### Ejemplo: creación de un proyecto Maven desde Eclipse

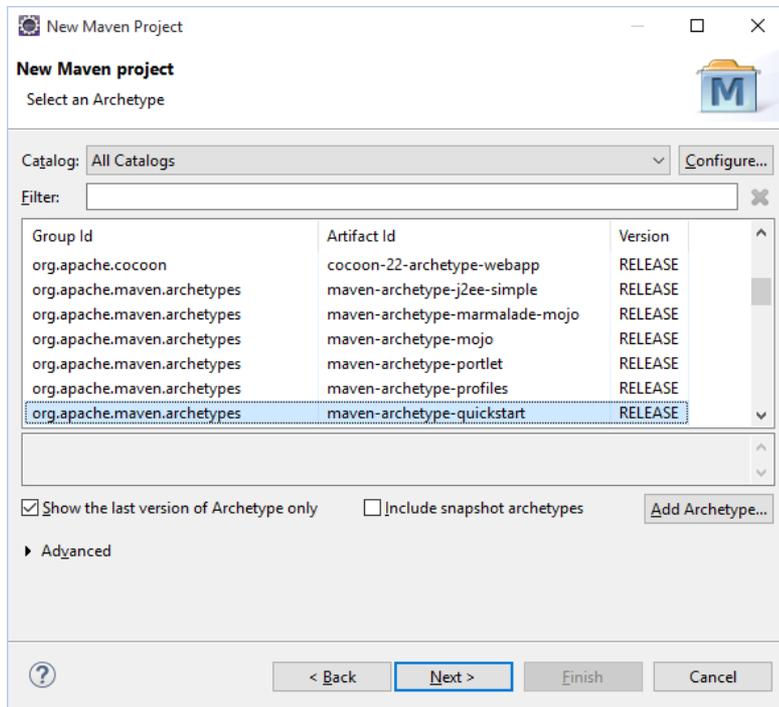
- File → New → Other



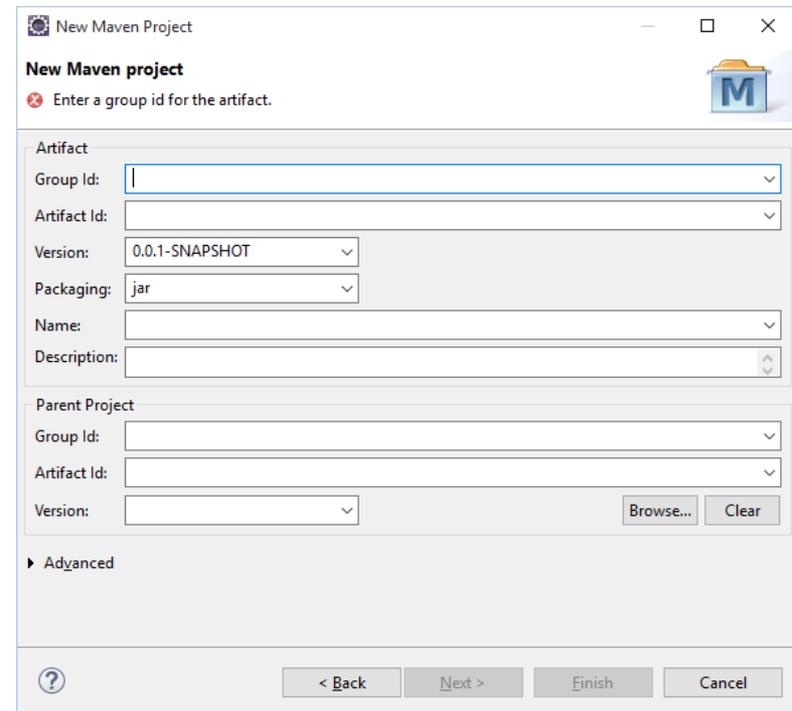
# SPRING Y JAVA EE

## 2. Maven

### Ejemplo: creación de un proyecto Maven desde Eclipse



Con arquetipo



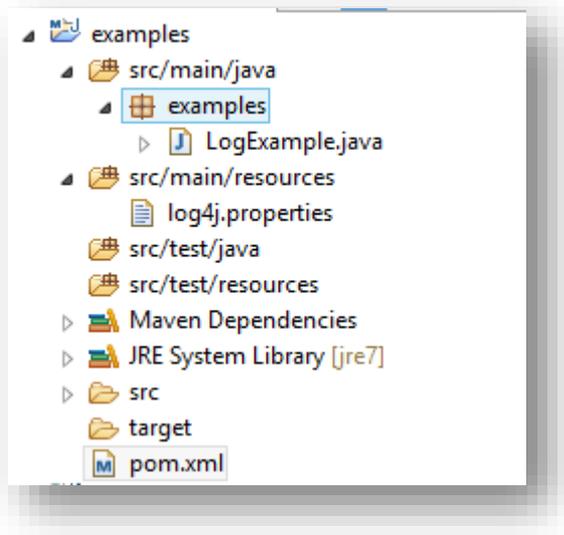
Sin arquetipo

# SPRING Y JAVA EE

## 2. Maven

### Ejemplo: uso de log4j en una aplicación Java

- Estructura del proyecto:



- pom.xml:

```
<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>examples</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.17</version>
    </dependency>
  </dependencies>
</project>
```

# Índice de contenidos

1. Java Enterprise Edition
2. Maven
3. Spring
  - Introducción
  - Spring vs Java EE
  - Inyección de dependencias
  - Arquitectura de una aplicación web Spring
  - Spring Boot

# 3. Spring

## Introducción

- Spring es un **framework** de desarrollo de aplicaciones empresariales basado en tecnologías Java
- El objetivo fundamental de Spring es **simplificar** el desarrollo Java
  - La primera versión fue escrita por Rod Johnson y descrita en su libro *Expert One-on-One J2EE Design and Development* (octubre 2002)
- *Open source* (licencia Apache 2.0)
- Versión estable (octubre 2015): 4.2.3



<http://spring.io/>

## 3. Spring

### Introducción

- Spring tiene una estructura modular
- Algunos de los proyectos Spring más significativos son:

- Núcleo de Spring
- Proporciona inyección de dependencias
- Incluye Spring MVC

Spring  
Framework



- Simplifica el despliegue de aplicaciones
- Convención sobre configuración

Spring  
Boot



- Proporciona acceso a bases de datos
- Relacionales o no relacionales

Spring  
Data



- Soporte para autenticación y autorización

Spring  
Security



- Hay todavía más: <https://spring.io/projects>

## 3. Spring

### Spring vs Java EE

- Java EE es un conjunto de librerías estándar en Java
- Spring es un framework software libre que se apoya en algunos estándares Java EE
- Hay desarrolladores que sólo usan Java EE y otros que combinan Spring y Java EE en sus aplicaciones

# 3. Spring

## Spring vs Java EE

- Esquema típico de una aplicación Java EE:



Para implementar la aplicación web se pueden usar librerías de Java SE y las librerías de Java EE proporcionadas por el servidor de aplicaciones

# 3. Spring

## Spring vs Java EE

- Esquema típico de una aplicación Spring:



Para implementar la aplicación web se pueden usar librerías de Java SE, las librerías proporcionadas por el contenedor web y el framework Spring

# 3. Spring

## Inyección de dependencias

- A los **componentes** de la aplicación Spring se los denomina **beans**
- Spring dispone de un sistema de **inyección de dependencias** para beans
  - Crea un objeto por cada módulo definido
  - **Inyecta las dependencias** en los módulos que dependen de él
- En las versiones iniciales de Spring, los beans se definían en XML
- En Spring 2.5 se introdujo la anotación **@Component** que permite definir un componente que es automáticamente descubierto por Spring
- Si un componente depende otro, define ese componente como un atributo anotado con **@Autowired**

# SPRING Y JAVA EE

## 3. Spring

### Inyección de dependencias

- Ejemplo:



```
@Component
public class UsersService {

    public int getNumUsers() {
        return 5;
    }
}
```

```
@Controller
public class GreetingController {

    @Autowired
    private UsersService usersService;

    @RequestMapping("/greeting")
    public ModelAndView greeting() {
        return new ModelAndView("greeting_template")
            .addObject("name", usersService.getNumUsers() + " users");
    }
}
```

# 3. Spring

## Inyección de dependencias

- En algunas ocasiones es necesario configurar los componentes de la aplicación
- Para ello, en la clase de la aplicación se puede **crear el componente manualmente** pasando cualquier configuración como parámetro en el constructor o invocando los métodos del objeto
- Se define un método anotado con **@Bean** que crea el componente

### Inyección de dependencias

- Ejemplo:

La creación manual del componente permite configurarlo

```
@SpringBootApplication
public class Application {

    @Bean
    public UserService userService() {
        return new UserService(10);
    }

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

```
@Component
public class UserService {

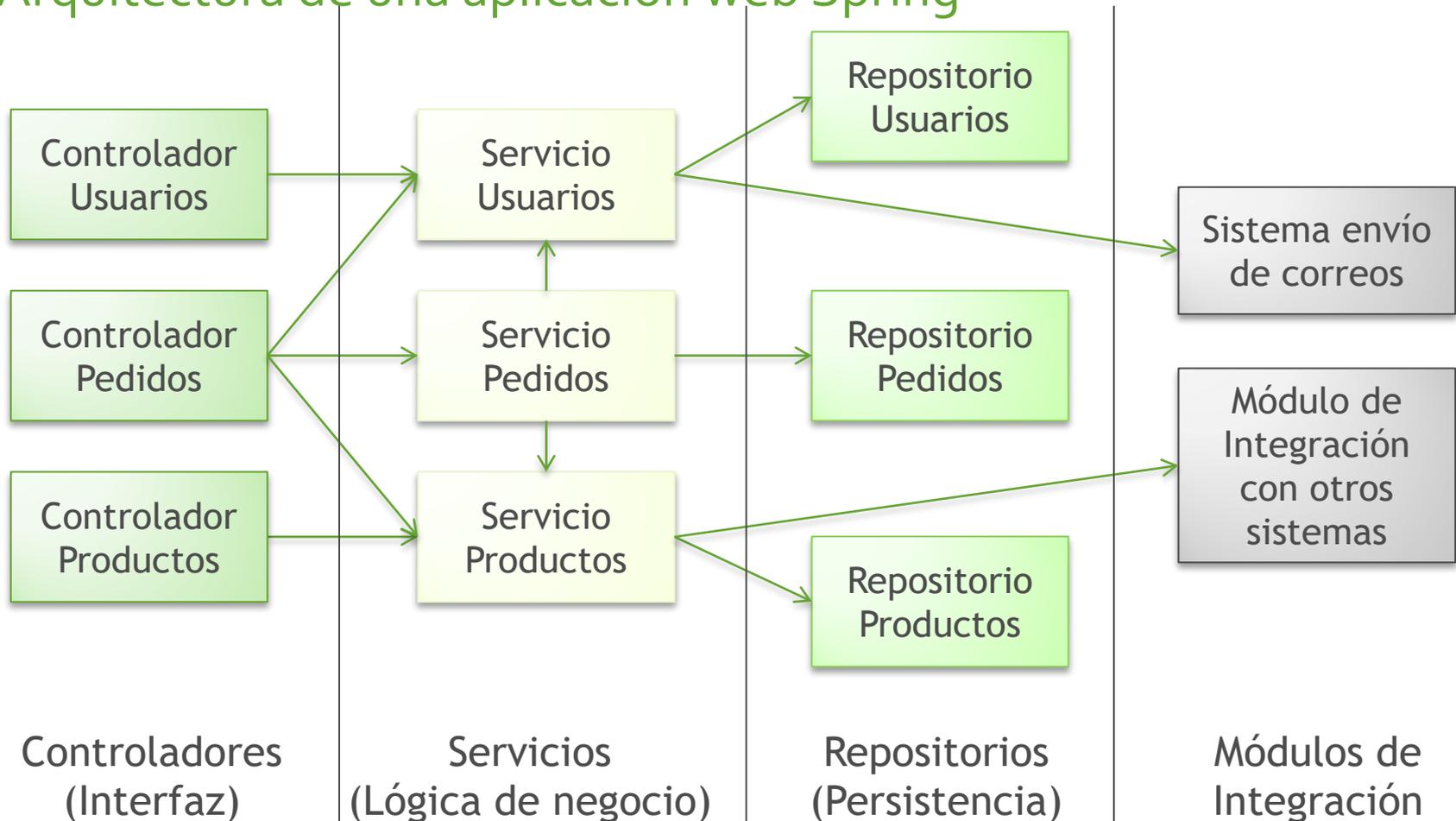
    private int numUsers;

    public UserService(int numUsers) {
        this.numUsers = numUsers;
    }

    public int getNumUsers() {
        return numUsers;
    }
}
```

# 3. Spring

## Arquitectura de una aplicación web Spring



# 3. Spring

## Arquitectura de una aplicación web Spring

- La mayoría de las aplicaciones web utilizan bases de datos para guardar su información
- Todas esas aplicaciones tienen una **arquitectura** similar que facilita su desarrollo y mantenimiento
- Los servicios y los módulos de integración suelen estar anotados con `@Service` para indicar su naturaleza
- `@Service` es similar a `@Component` (se pueden inyectar en otros componentes)

# 3. Spring

## Spring Boot

- Es una librería que facilita el desarrollo de aplicaciones con Spring
- Permite usar el servidor web **Tomcat embebido** en la aplicación
- **Simplifica la configuración** y acelera el desarrollo
- Es una librería bastante reciente, antes se implementaban las aplicaciones usando directamente Spring

## 3. Spring

### Spring Boot

- Esquema típico de una aplicación Spring Boot:



Para implementar la aplicación web se pueden usar librerías de Java SE, las librerías proporcionadas por el contenedor web y el framework Spring y Spring Boot

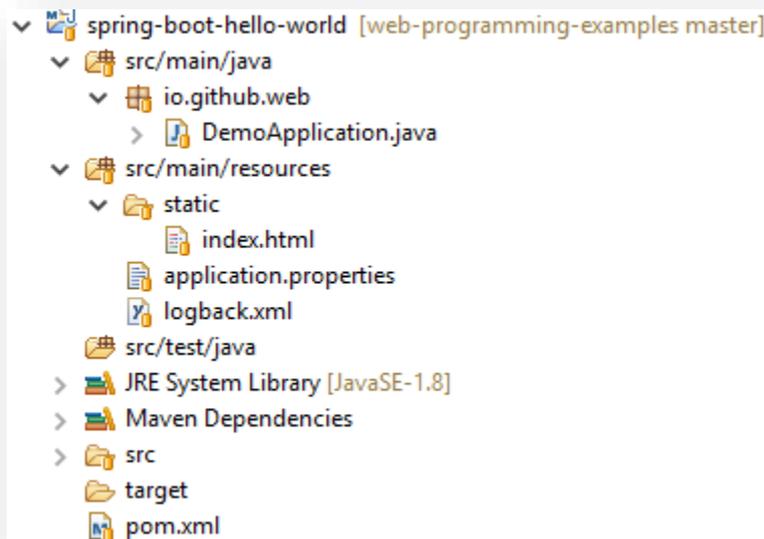
# SPRING Y JAVA EE

## 3. Spring

Fork me on GitHub

### Spring Boot

- Podemos crear aplicaciones Spring Boot de forma sencilla con la aplicación web <https://start.spring.io/>
- Vamos a ver uno de los ejemplos más simples de aplicación Spring Boot (*Hello World*)



# SPRING Y JAVA EE

## 3. Spring

### Spring Boot

- *Hello World* :

Proyecto padre del que se hereda la configuración

Java 8

Tipo de proyecto Spring que estás desarrollando

Plugin de spring-boot (sirve para empaquetar la aplicación)

pom.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>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.2.7.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 Y JAVA EE

## 3. Spring

### Spring Boot

- *Hello World* :

index.html

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

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

DemoApplication.java

```
package io.github.web;

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

@SpringBootApplication
public class DemoApplication {

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

# 3. Spring

## Spring Boot

- *Hello World* :

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>
```

application.properties

```
server.port=8080
```