

¿Qué es Spring Framework?
---------------------------

Un Framework es un marco o entorno de trabajo de código libre compuesto por reglas y herramientas que facilitan enormemente el desarrollo de aplicaciones. Spring es el entorno de trabajo más popular para crear aplicaciones Java. Ofrece múltiples módulos que abarcan una gran variedad de servicios que facilitan la implementación de elementos de aplicaciones: Spring Test, Spring ORM, Spring JDBC, Spring MVC, Spring Security, Spring AOP...

Spring es la alternativa al desarrollo de aplicaciones JEE (Java Enterprise Edition), más simple y más ligera y nos permite desarrollar aplicaciones en menor tiempo.

### ¿Por qué se utiliza Spring?

1999-2003 → para desarrollar aplicaciones Java en el entorno empresarial se utilizaba **J2EE** pero según iban saliendo nuevas versiones la complejidad de esta tecnología era exponencial y aumentaba mucho y los componentes como los EJB (Enterprise JavaBEans) daban muchos problemas y eran complejos y muy lentos en su implementación. Consumían muchos recursos.

A partir de 2004→ **Spring**, lo creó Rod Johnson para solucionar los problemas que habían surgido para desarrollar aplicaciones J2EE. Era simple y ligero y los desarrolladores se pasaron a utilizar este framework.

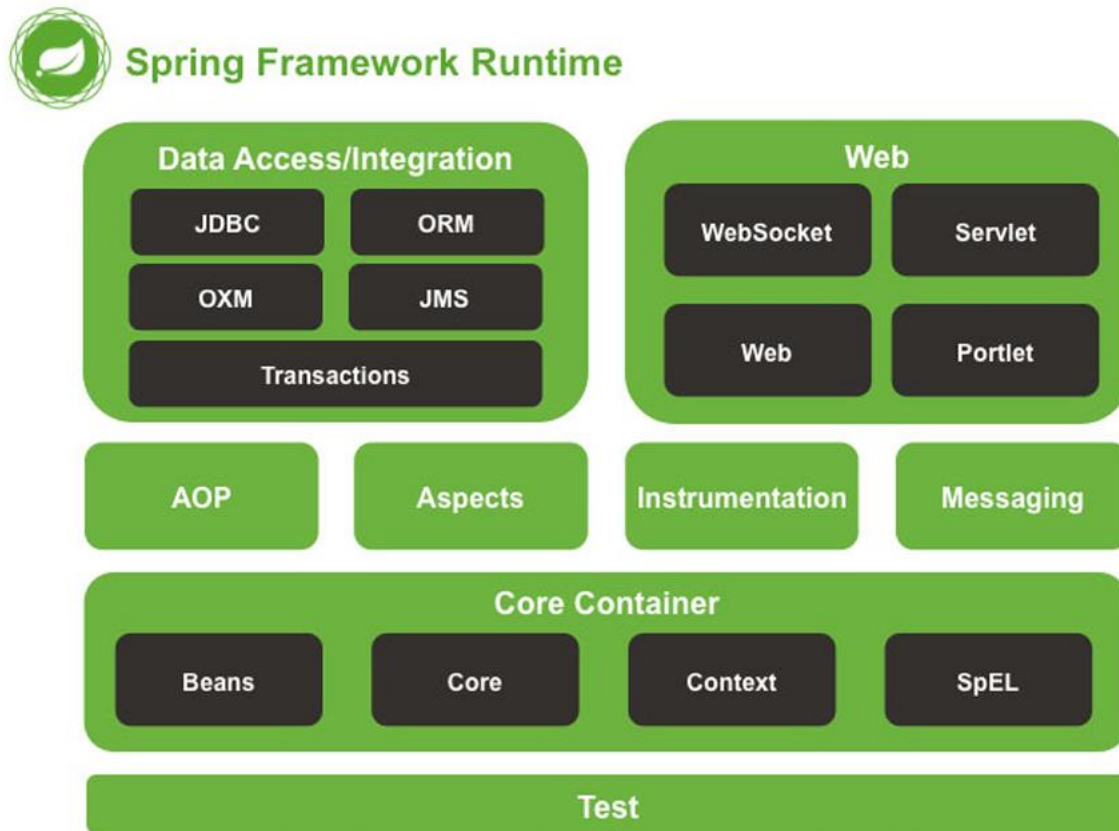
Los dueños de Java entonces vieron las características que hacían tan simple y sencillo a Spring y decidieron implementar esas características en Java Enterprise, y en 2005 sale la versión 5 de Java Enterprise y el manejo de los componentes pasó de tener una implementación muy lenta a ser mucha más rápida en ejecución y consumir menos recursos gracias a que JEE y Spring compartían muchas cosas.

### Ventajas:

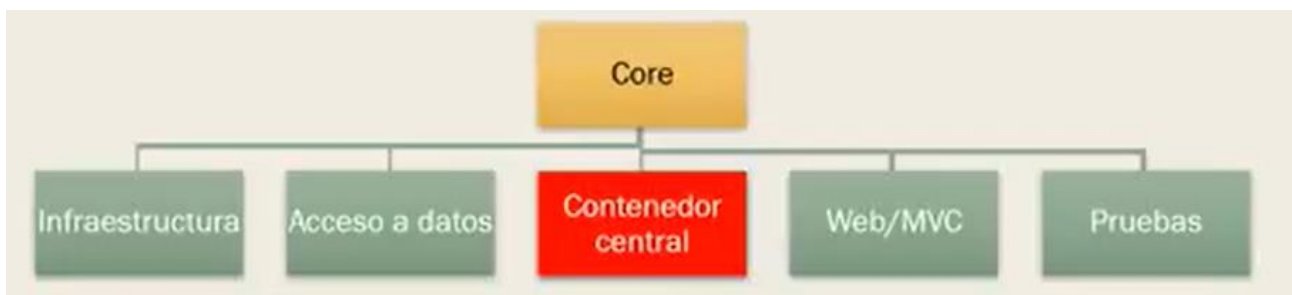
- Inyección de dependencias. Permite la separación de los módulos de un programa Java. Los cambios sólo serán necesarios en un punto y no en muchos.
- Desarrollo sencillo
- Minimiza el código repetitivo
- Simplifica el acceso a datos: escribir menos código
- Programación orientada a aspectos (AOP): permite modularizar nuestro programa y separar las distintas tareas que se llevan a cabo.

## Estructura

Los módulos de Spring pueden resumirse en el siguiente diagrama:



El módulo principal de Spring es Spring Core Container (núcleo):



- Contenedor central: realiza el trabajo relacionado a la lectura de ficheros config, creacion de Beans, manejo de propiedades y dependencias, ...
- Infraestructura: transacciones, logeos, seguridad de las aplicaciones,...
- Acceso a datos: modelos de JDBC, transacciones (módulos OXM y JMS), ORM.
- Web/MVC: Acceso web remoto, programación distribuida, integración de otras tecnologías como JSF
- Pruebas: Unit, Mock, Integration

## ¿Qué es Spring Boot?

Como hemos visto, Spring tiene una gran cantidad de módulos que implica muchas configuraciones que pueden necesitar mucho tiempo. La solución a esta dificultad inicial es Spring Boot. Es una extensión de Spring Framework que simplifica drásticamente las tareas para el desarrollo de aplicaciones Java. Es decir, nace para crear aplicaciones Spring simplificando la configuración y desarrollo. Tiene un servidor web Tomcat integrado y no requiere de configuración XML.

### Requisitos necesarios

- ✓ **JDK** a partir de la versión 17. Descarga en Oracle.com  
**Para conocer la versión de java instalada:** `java -version`
- ✓ **Servidor Java:** Tomcat ( embebido en Spring Boot)
- ✓ **IDE:** Eclipse STS. Descarga Spring STS: <https://spring.io>
- ✓ **Postman:** nos permite probar, documentar APIs. Descarga postman.com
- ✓ **Maven:** Gestión de dependencias
- ✓ **BD:** MySQL

## Conceptos importantes

### Principio de inversión de control (IOC)

Se basa en invertir el flujo de control del software. Es decir, el framework tendrá esa responsabilidad y en vez de que nosotros tengamos que ocuparnos de las dependencias a otros objetos, él se ocupa de las mismas.

Antes de IoC, las clases dentro de una aplicación solían ser responsables de crear sus propias dependencias (por ejemplo, crear instancias de otros objetos que necesitaban). Esto hacía que las clases estuvieran fuertemente acopladas y más difíciles de probar y mantener.

Con IoC, las clases ya no controlan cómo se obtienen sus dependencias. En su lugar, un **contenedor** o **framework**, como Spring, se encarga de la gestión de estos objetos y los proporciona cuando es necesario.



El objeto A depende de una funcionalidad de B. Y como B está gestionado por el contenedor IOC, es el Framework el que le pasa una instancia de B para que lo use.

### Inyección de dependencias (DI)

Es uno de los patrones utilizados para implementar IOC, en ella, los objetos no se preocupan por instanciar sus dependencias.

Hay tres tipos comunes de inyección de dependencias:

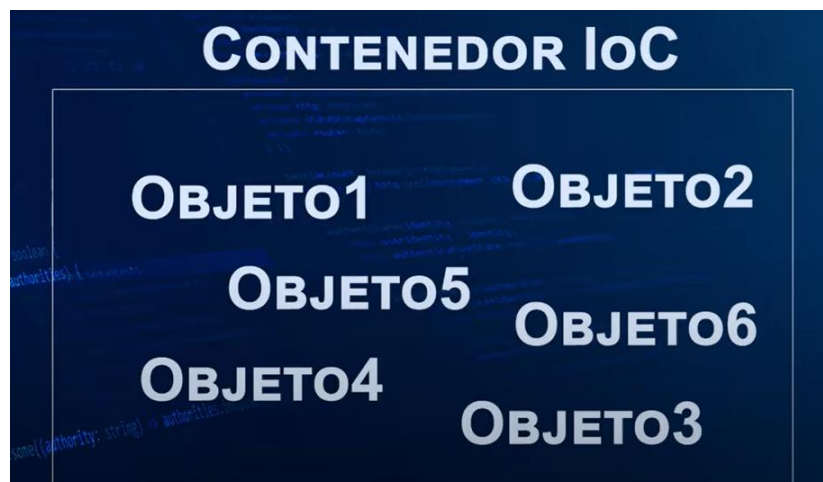
- **Inyección por constructor:** Las dependencias se proporcionan a través del constructor de la clase.
- **Inyección por setter:** Las dependencias se inyectan a través de métodos setter.
- **Inyección por interfaz:** Las dependencias se inyectan a través de métodos definidos en una interfaz, aunque este es menos común.

### Beans

Los beans no son más que objetos creados y administrados por el contenedor de Spring. Es decir, objetos que Spring maneja por nosotros.

### Application Context

Es el contenedor central de Spring y configura, instancia y maneja todos los beans de la aplicación. De tal forma que cuando un objeto necesite una dependencia, Spring le dará una instancia de la misma.



La configuración de beans se puede hacer con ficheros xml y también con anotaciones de Java.

### Anotaciones

Son etiquetas que se añaden a clases, métodos, variables, etc. en un programa Java.

Comienzan por @ y sirven para añadir metadatos a las clases. ¿Y qué son los metadatos? Conjunto de datos que describe a un objeto: su contenido y/o propósito.

Se procesan en tiempo de compilación y de ejecución.

Ya conocemos una anotación de Java: **@Override**, ¿recordáis?

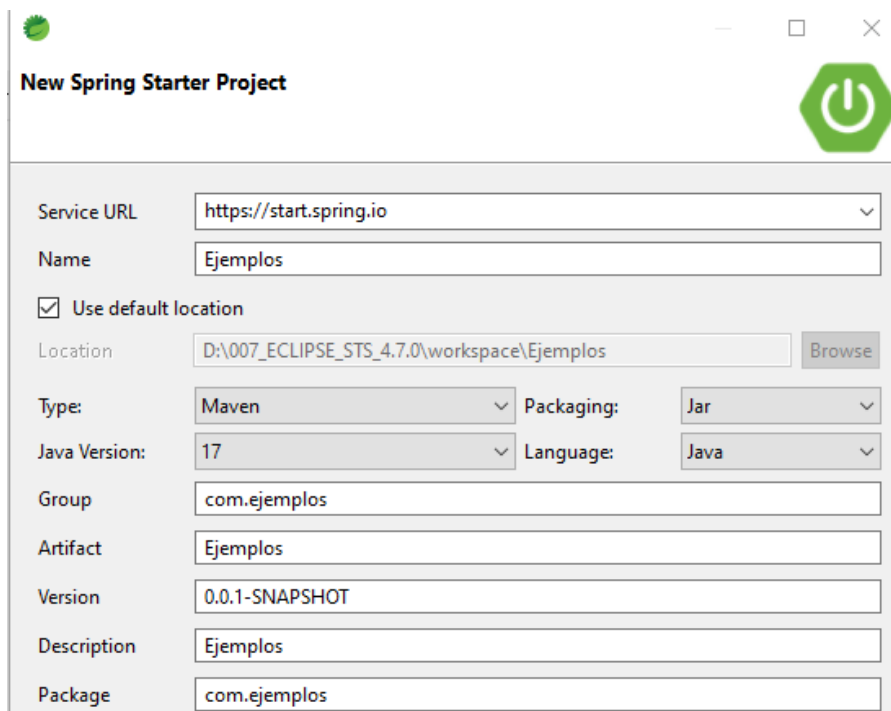
La inyección de dependencias se hace en Spring con la anotación **@Autowired**. Puede utilizarse en el constructor, en el setter o en el campo del objeto. Cuando Spring detecta la anotación @Autowired, Spring busca automáticamente un bean en el contenedor que coincida con el tipo de la dependencia y lo inyecta.

Ejemplo:

```
public class Car {  
    @Autowired // Inyección de dependencia en el atributo  
    private Engine engine;  
  
    public void drive() {  
        engine.start();  
        System.out.println("Car is driving!");  
    }  
}
```

### Creación de proyecto web con Spring Boot.

En Spring Tool Suite (STS): File / New/ Spring Starter Project



**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

Package:

**Agregamos las dependencias:** Con componentes o bibliotecas que nuestra aplicación necesita para funcionar correctamente.

## New Spring Starter Project Dependencies



Spring Boot Version:

Frequently Used:

<input type="checkbox"/> JDBC API	<input type="checkbox"/> Lombok	<input type="checkbox"/> MySQL Driver
<input checked="" type="checkbox"/> Spring Boot DevTools	<input checked="" type="checkbox"/> Spring Web	

Available:

X

▼ Template Engines

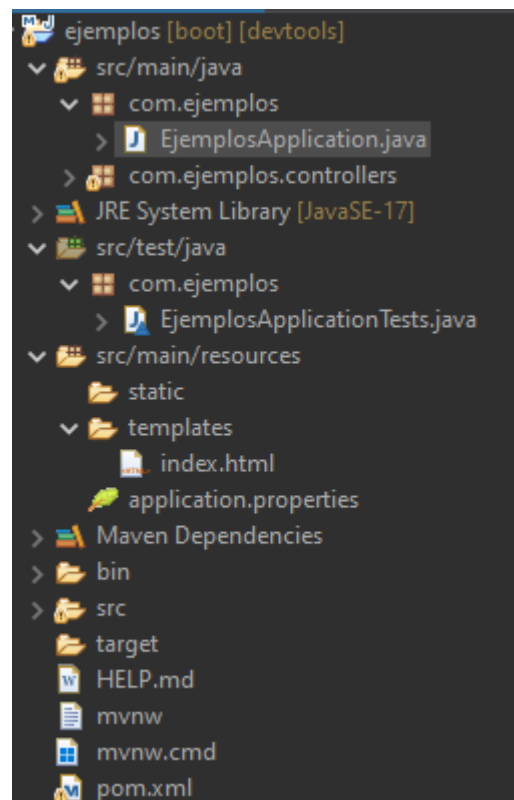
☒ Thymeleaf

Selected:

- X Spring Boot DevTools
- X Thymeleaf
- X Spring Web

Si cuando arrancamos el proyecto el puerto 8080 está ocupado, en `src/main/resources` tenemos que cambiarlo anotando: `server.port=8081` (o cualquier otro puerto).

## Estructura de proyectos



- src/main/java
  - EjemplosApplication.java: Tiene el main() que lanza la aplicación con la anotación @SpringBootApplication.
- src/test/java
  - EjemplosApplicationTests: Junit para las pruebas
- src/main/resources
  - application.properties: fichero para añadir las propiedades de configuración
  - Directorio static: Para almacenar hojas de estilos, imágenes, javascript,...
  - Directorio templates: plantillas