# Framework Spring (Legacy)

## Índice



---

## 1. ¿Qué es Maven?

Aunque Maven se puede utilizar con diversos lenguajes (C#, Ruby, Scala...) se usa principalmente en proyectos Java, donde es muy común ya que está escrita en este lenguaje. De hecho, frameworks Java como Spring y Spring Boot la utilizan por defecto, por lo que conviene conocerla si trabajas en la plataforma Java y, desde luego, con Spring.

Maven utiliza convenciones sobre dónde colocar ciertos archivos para el proceso de build de un proyecto, por lo que solo debemos establecer las excepciones y por lo tanto simplifica mucho el trabajo. Además, es una herramienta declarativa. Es decir, todo lo que definamos (dependencias en módulos y compontes externos, procesos, orden de compilación, plugins del propio Maven...) se almacena en un archivo XML que Maven lee a la hora de funcionar.
Otras alternativas, como Gradle no utilizan archivos XML, sino de otro tipo, pero usan los mismos conceptos de Maven.

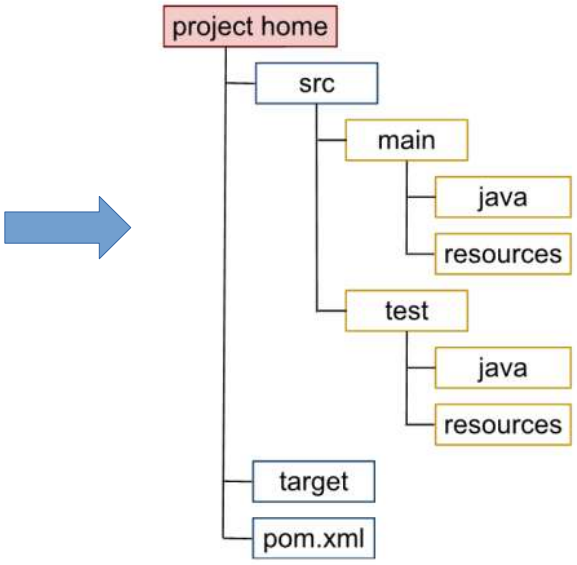
Con Maven se puede:

- ✔ Gestionar las dependencias del proyecto, para descargar e instalar módulos, paquetes y herramientas que sean necesarios para el mismo.
- ✔ Compilar el código fuente de la aplicación de manera automática.
- ✔ Empaquetar el código en archivos .jar o .zip.
- ✔ Instalar los paquetes en un repositorio (local, remoto o en el central de la empresa)
- ✔ Generar documentación a partir del código fuente.
- ✔ Gestionar las distintas fases del ciclo de vida de las build: validación, generación de código fuente, procesamiento, generación de recursos, compilación, ejecución de test …

¿Qué es el archivo pom.xml?

La unidad básica de trabajo en Maven es el llamado Modelo de Objetos de Proyecto conocido simplemente como POM (de sus siglas en inglés: Project Object Model).
Se trata de un archivo XML llamado pom.xml que se encuentra por defecto en la raíz de los proyectos y que contiene toda la información del proyecto: su configuración, sus dependencias, etc..

Esta sería la estructura habitual de un proyecto Java que utiliza Maven




Este es el contenido de un archivo pom.xml sencillo de ejemplo:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>es.campusmvp</groupId>d>
  <artifactId>demo</artifactId>
  <version>1.0</version>
  <!-- Dependencias -->
  <dependencies>
          <dependency>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-web</artifactId>
          </dependency>
          <dependency>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-test</artifactId>
                    <scope>test</scope>
                    <exclusions>
                              <exclusion>
                                        <groupId>org.junit.vintage</groupId>
                                        <artifactId>junit-vintage-engine</artifactId>
                              </exclusion>
                    </exclusions>
          </dependency>
          <dependency>
                    <groupId>org.apache.tomcat.embed</groupId>          (1)
                    <artifactId>tomcat-embed-jasper</artifactId>
                    <scope>compile</scope>
          </dependency>
  </dependencies>
</project>
```

Como puedes ver, existe una serie de dependencias: la primera es el starter de Spring Boot para generar una aplicación Web, la segunda es el starter que trae todas las dependencias para poder ejecutar test (por ejemplo JUnit o Mockito) y que excluye todo lo relativo al motor antiguo de Junit. Al final añade soporte web en el servidor **Apache Tomcat**, el cual está **embebido (1)** en SpringBoot y no es necesario desplegarlo.
Apache Tomcat se lanza desde el puerto 8080 por defecto. En caso de querer cambiar el puerto, puedes hacerlo desde el archivo application.properties ubicado en la ruta:

/src/main/resources/application.properties, escribiendo la siguiente línea de código:

server.port = 9898  (O el puerto libre que prefieras)

Una vez relanzada la aplicación, podrás ver que los cambios se han efectuado.

Más info en: https://www.arteco-consulting.com/post/tutorial-de-maven

## 3. Inversión de Control

Nosotros no nos ocuparemos de añadir la instanciación de objetos a nuestro programa. Será Spring quien nos facilite dicha creación sin necesidad de modificar nuestro programa Main. Existen 3 formas de hacerlo:

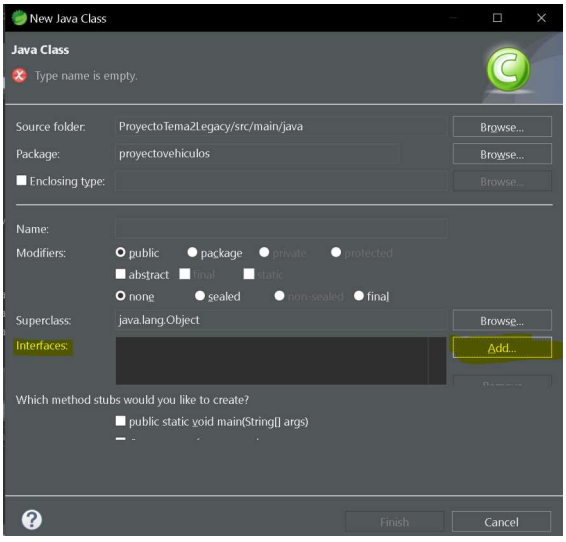
Archivo de configuracióm XML (LEGACY)
Java Source Code
Java Annotations

Para usar métodos comunes (O también para crear jerarquía de clases) podemos:

- Usar una clase abstracta de la que hereden otras.
- Crear una Interface con las cabeceras de los métodos comunes. (De esta forma obligamos a implementar estos métodos)

**Ejercicio a realizar:**

1. **Crea un paquete llamado `proyectoVehiculos`**
2. **Crea las siguientes clases: Avion, Coche, Motocross que implementen la Interface Vehiculo, la cual contiene el método public String getTareas().** *Para facilitar el proceso, al crear cada clase, escribe su nombre y añade la Interface con el menú de Eclipse:*




3. **Implementa este método en todas las clases. Coche conduce por ciudad, Avion conduce**

**por aire, Motocross conduce por caminos de tierra.**

4. **Crea una clase Main llamada PruebaVehiculos donde se prueben todas las clases.**

Inconveniente: Debes instanciar todos los objetos.

Mejora: El problema está que para cualquier objeto que yo quiera crear, debo modificar mi clase Main. Para solucionar esto, Spring trabaja con la inversión de control, gracias a sus beans (Cada uno debe tener un nombre distinto) y a sus objectFactory (ObjectFactory es responsable de crear objetos de un tipo específico)
Vamos a ver como se hacía antiguamente con Spring (Forma legacy):

Crea el siguiente archivo llamado applicationContext.xml en la ruta src/main/java:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.1.xsd
">

<!--  Creamos el bean para instanciar objetos especificando un id y donde se
localiza la clase de la que queremos construir objetos.

Esto lo haremos una única vez ya que SpringBoot genera mucho código de este tipo
de forma autónoma.-->

<!-- Otro generador de tipo Coche-->
<bean id="GeneradorVehiculos" class="proyectoVehiculos.Coche">

</bean>

</beans>
```

Copia la nueva clase Main creada con Spring y realiza la prueba (Tiene muchas similitudes con la forma de trabajar de un fichero: **1-** Apertura o Carga **2-** Uso o llamada **3-** Cierre ):

```
package proyectoVehiculos;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class PruebaVehiculos_Uno {

          public static void main(String[] args) {

          //Paso 1: Cargar archivo XML
          ClassPathXmlApplicationContext miContexto = new
ClassPathXmlApplicationContext("applicationContext.xml");

          //Paso 2: Hacemos la llamada al bean

                                              //id      //tipo el que especifique el bean
```

```
            Vehiculo v1= miContexto.getBean("GeneradorVehiculos", Vehiculo.class);

            System.out.println(v1.getTareas());

            //Paso 3: Cerramos el bean
            miContexto.close();
            }
}
```

## 4. Inyección de dependencias

Dependencias === Modularización

Unos módulos DEPENDEN de otros. Unos objetos / clases DEPENDEN de otros.

Ejemplo de la Clase Coche: El motor depende de las ruedas para poder mover el coche. Las ruedas dependen del volante para girar. Las ruedas a su vez dependen de los frenos para parar.

Spring inyectará a la clase principal los objetos necesarios. Ventaja: Si otra clase necesita esos objetos (dependencias) se podrán inyectar sin necesidad de instanciar dichos objetos.

¿CÓMO CREAR UNA DEPENDENCIA?

Existen varias posibilidades:

### 4.1. Crear dependencias a través de un constructor. (Legacy)

### 4.2. Crear dependencias a través de un setter. (Legacy)

### 4.3. Crear dependencias a través de campos / atributos (Legacy)

### 4.4. Java Annotations (@Autowired → más avanzado, siguiente Tema)

## 4.1. Mediante un constructor. (Legacy)

1) Crear la clase (Clase Mantenimiento) y la interfaz de la dependencia (Interface CreacionMantenimiento)

2) Creación del constructor en la clase (Coche) para la inyección de la dependencia.

3) Configurar la inyección de la dependencia en el archivo XML (ApplicationContext.xml)

4) Realizar la prueba en Main

Ejemplo:

1) Clase e interface:

```
public class MantenimientoAp4 implements CreacionMantenimientoAp4{
```

```
	@Override
	public String getMantenimiento() {
		return "Mantenimiento de vehículo realizado";
	}
}

public interface CreacionMantenimientoAp4 {
	public String getMantenimiento();
}


public interface Vehiculo {
	public String getTareas();
	//Apartado 4 Inyeccion de dependencias
	public String getMantenimiento();
}
```

Implementa ese método en las 3 clases: **Avion, Coche, Motocross**

2) Añadimos el código necesario a la clase Coche:

```
public class Coche implements Vehiculo{
	public String getTareas() {
		return "Conduzco por ciudad";
	}


	//Paso 1
	//Apartado 4 Inyeccion de dependencias
		@Override
		public String getMantenimiento() {
			return "Mantenimiento realizado en coche" +
creacion.getMantenimiento();
		}


		//Paso 2
		private CreacionMantenimientoAp4 creacion;


		//Paso 3
		//Creación del constructor en la clase (Mantenimiento) para la
inyección de la dependencia.

		public Coche(CreacionMantenimientoAp4 creacion) {
			this.creacion = creacion; //Inyección de la dependencia
		}

		//Paso 4 --> Vincular el bean en archivo XML

}
```

3) Archivo XML

```
<!-- Otro generador de tipo Coche-->
<bean id="GeneradorVehiculos" class="proyectoVehiculos.Coche">
```

```
		<!-- Paso 5-->
<constructor-arg ref="miMantenimiento"></constructor-arg>

</bean>

<!-- Paso 4 Vinculación del bean-->
<bean id="miMantenimiento" class="proyectoVehiculos.MantenimientoAp4">

</bean>

</beans>
```

Pasemos ahora a modificar nuestra clase Main llamada (PruebaCochesBean.java)

4) Clase Main:

```
package proyectoVehiculos;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class PruebaVehiculos_Dos {

		public static void main(String[] args) {

		//Paso 1: Cargar archivo XML
		ClassPathXmlApplicationContext miContexto = new
ClassPathXmlApplicationContext("applicationContext.xml");

		//Paso 2: Hacemos la llamada al bean

		//id                                    //tipo el que especifique el bean
		Vehiculo v1= miContexto.getBean("GeneradorVehiculos", Vehiculo.class);

		System.out.println(v1.getTareas());

		//paso 6: Mantenimiento inyectado === inyección de dependencia
		System.out.println(v1.getMantenimiento());

		miContexto.close();
		}

}
```

## 4.2. Mediante un setter. (Legacy)

1. Crear la clase (Clase Mantenimiento) y la interfaz de la dependencia (Interface CreacionMantenimiento) Ya realizado en el ejemplo anterior

2. Creación de un método setter en la clase Motocross para la inyección de dependencia.

```
public class Motocross implements Vehiculo {
		@Override
		public String getTareas() {
				return "Conduzco por caminos de tierra";
		}
```

```
	// Apartado 4 Inyeccion de dependencias
	@Override
	public String getMantenimiento() {
			return "Mantenimiento realizado en motocross (" +
creacionManten.getMantenimiento()+")";

	}

	// Paso 1 --> Creación de atributo
	private CreacionMantenimientoAp4 creacionManten;



	// Paso 2 --> Creación del setter

	public void setCreacionManten(CreacionMantenimientoAp4 creacionManten) {
			this.creacionManten = creacionManten;
	}


	// Paso 3 --> Vincular el bean en archivo XML para crear la dependencia
}
```

3. Configurar la inyección de la dependencia en el archivo XML (ApplicationContext.xml)

```
<!-- bean para el setter-->
<bean id="miMantenimientoMotocross" class="proyectoVehiculos.Motocross">
<!-- name debe ser: quitando de la palabra set y 1º letra en minúscula
ref debe ser: nombre del id del bean que realiza mantenimientos-->
	<property name="creacionManten" ref="miMantenimiento"></property>
</bean>
```

4. Realizar la prueba en Main.

```
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class PruebaVehiculos_Tres {

	public static void main(String[] args) {

			ClassPathXmlApplicationContext miContexto = new
ClassPathXmlApplicationContext("applicationContext.xml");

//NombreInterface   id del bean de Motocross
Vehiculo moto = miContexto.getBean("miMantenimientoMotocross", Vehiculo.class);

			System.out.println(moto.getTareas());

//paso 6: Mantenimiento inyectado === inyección de dependencia
			System.out.println(moto.getMantenimiento());
			miContexto.close();
	}

}
```

## 4.3. Crear dependencias a través de campos / atributos (Legacy)

A continuación se explica cómo se realizaría con Spring Legacy. Pasaremos directamente a realizarlo de la nueva forma en el siguiente PDF:

*Elijamos varios atributos de la Clase Motocross que queramos inyectar. Por ejemplo: marca, modelo. Para ello debes:*

1. *Crear los atributos en la clase con visibilidad privada.*
2. *Crea los getter y setter de cada atributo.*
3. *Crea en el bean de la Clase Motocross del XML (miMantenimientoMotocross) una propiedad para cada atributo: Este paso se hace parecido al formato de HTML:*

   *<property name="nombreDelAtributo" value="valorDelAtributo"></property>*
4. *Debido a que en el ejemplo anterior estabamos usando objetos de la interface Vehiculo.java, solo podríamos llamar a los métodos que contiene dicha interface. Dado que hemos creado nuevos métodos en la Clase Motocross, debemos usar un objeto de tipo Motocross. Crea una clase Main donde se muestre la marca y el modelo que has creado en el archivo XML. Para ello puedes ayudarte de la Clase Main del apartado 3.*

---

*Con estos ejercicios hemos visto como funcionaban antiguamente el framework Spring--> ~~Spring Legacy~~.*

*Para los próximos temas de SpringBoot puede servirte de ayuda: www.spring.io donde encontrarás muchos proyectos, documentación y tutoriales sobre el framework SpringBoot.*