

# El Framework Spring



## **TEMA 1: Introducción a Spring**

- Spring es un framework de desarrollo de aplicaciones para la plataforma Java.
- Es un framework de código abierto y existe una gran comunidad trabajando por su actualización y mejora.
- Está muy extendido en el mundo empresarial ya que permite gestionar aplicaciones de gran tamaño y sobre todo permite comunicar muy eficientemente dicha aplicación con los diferentes servicios externos a los que se conecta.
- Se creó pensando en simplificar los sistemas Java EE que interactúan con varios sistemas. En ese momento los modelos más extendidos se basaban en Java Servlet y EJBs.
- La primera versión se creó en 2004. Actualmente se encuentra en la versión 6.0.
- Se trata de un framework modular, es decir, podemos incluir sólo los módulos que necesitamos en nuestro proyecto, no es necesario utilizar todo el framework.
- **Spring es un framework basado en el paradigma de inyección de dependencias.**
- **Una serie de objetos (beans) son creados al inicio del proyecto y serán inyectados dentro de los componentes que los requieran de modo que dicha clase no tenga que construir el objeto.**
- La creación de estos beans estará definida dentro de un fichero xml o por medio de anotaciones Java.
- Dentro de Spring disponemos de varios módulos, estos son los más importantes:
  - Spring Core: se trata del módulo principal. Es el único obligatorio para el trabajo con el framework. Contiene la inyección de dependencias y la configuración y uso de objetos Java.
  - Spring Context: de donde se obtiene el contexto para interactuar con Spring.
  - AOP: programación orientada a aspectos.
  - DAO: interacción con los sistemas de bases de datos.
  - MVC: para gestionar toda la web.
  - Security: para el control de seguridad.

La documentación está disponible en:

<https://spring.io/projects>

## Primer Proyecto Spring (Hola Mundo)

- Dependencias de Maven necesarias:

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>6.1.13</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>6.1.13</version>
  </dependency>
</dependencies>
```

- Creamos la clase HolaMundo en el paquete spring.prueba con el siguiente código:

```
package es.hola;

public class HolaMundo {
    public void decirHola() {
        System.out.println("Hola mundo desde Spring");
    }
}
```

Ya tenemos una clase simple con un método.

Ahora vamos a comenzar a comprender la filosofía de Spring.

- Utilizar el framework Spring para gestionar un bean (objeto) de la clase HolaMundo.
- Después obtendremos ese bean del SpringContext (contexto de spring) y llamaremos al método decirHola.

Para ello realizaremos estos pasos:

- Crearemos el archivo **context.xml** para la configuración del bean, dentro del paquete src de nuestro proyecto o en la carpeta Java si estamos creando un proyecto Maven con Eclipse. Si usamos IntelliJ debemos situar el archivo en la carpeta resources.

```
<?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.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

<bean id="holita" class="es.hola.HolaMundo"/>
</beans>
```

En este xml creamos todos los beans de nuestro proyecto, en este caso sólo uno que está asociado a la clase HolaMundo.

- Ahora completamos el código de la clase App de la siguiente forma:

```
package es.hola;

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

public class App {
    public static void main(String[] args) {
        ApplicationContext context = new
            ClassPathXmlApplicationContext("context.xml");

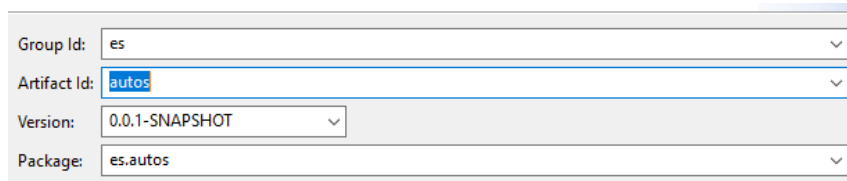
        // Creamos una instancia del bean por medio del contexto.
        HolaMundo miBean = (HolaMundo) context.getBean("holita");
        miBean.decirHola();
        ((ClassPathXmlApplicationContext) context).close();
    }
}
```

## TEMA 2: Inyección de dependencias

- La Inyección de Dependencias es un patrón de diseño orientado a objetos en el que se suministran objetos a un recurso en lugar de que dicho recurso tenga que crearlo.
- Las propiedades de los objetos que vamos a usar en nuestra aplicación se van a definir en el momento de iniciar la aplicación.
- Por lo tanto, ahora, a la hora de utilizar un objeto en concreto ya no nos tendremos que preocupar en crearlo si no sólo inyectar donde sea necesario.
- Por ejemplo, podríamos tener un objeto de conexión a base de datos creado desde el principio, el cual puede ser compartido por todos los usuarios y clases del proyecto.

### Ejemplo (inyectar un motor dentro de un vehículo)

- Creamos un proyecto llamado autos y agregamos las librerías de Spring como en el ejemplo anterior.



A screenshot of a Maven IDE configuration window. It contains four fields with dropdown menus:

- Group Id: es
- Artifact Id: autos
- Version: 0.0.1-SNAPSHOT
- Package: es.autos

- Creamos las clases Motor y Vehiculo.

```
package es.autos;

public class Motor {
    private String tipo; // Diesel, gasolina, etc.
    private int caballos;

    public Motor(String tipo, int caballos) {
        super();
        this.tipo = tipo;
        this.caballos = caballos;
    }

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public int getCaballos() {
        return caballos;
    }

    public void setCaballos(int caballos) {
        this.caballos = caballos;
    }
}
```

```
package es.autos;

public class Vehiculo {
    private String marca;
    private String modelo;
    private Motor motor;

    public String comprobarMotor() {
        if(motor != null)
        {
            return "Tipo de motor: "+this.motor.getTipo()+"\n"+
                "Caballos: "+this.motor.getCaballos();
        }
        else {
            return "NO existe motor";
        }
    }

    public String getMarca() {
        return marca;
    }
    public void setMarca(String marca) {
        this.marca = marca;
    }
    public String getModelo() {
        return modelo;
    }
    public void setModelo(String modelo) {
        this.modelo = modelo;
    }
    public Motor getMotor() {
        return motor;
    }
    public void setMotor(Motor motor) {
        this.motor = motor;
    }
}
```

- Creamos el archivo context.xml.

```
<?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.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="motor" class="es.autos.Motor">
        <constructor-arg index="0" value="diesel" type="java.lang.String"/>
        <constructor-arg index="1" value="100" type="int"/>
    </bean>
    <bean id="vehiculo" class="es.autos.Vehiculo">
        <property name="marca" value="Ford"/>
        <property name="modelo" value="Fiesta"/>
        <property name="motor" ref="motor"/>
    </bean>
</beans>
```

- Creamos la clase que nos sirva de testeo de la clase Vehiculo.

```
package es.autos;

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

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context = new
        ClassPathXmlApplicationContext("context.xml");

        Vehiculo vehiculo = (Vehiculo) (context.getBean("vehiculo"));

        System.out.println(vehiculo.comprobarMotor());
        System.out.println("Vehiculo marca: " + vehiculo.getMarca());
        System.out.println("Vehiculo modelo: " + vehiculo.getModelo());

        ((ClassPathXmlApplicationContext) context).close();
    }
}
```

## El archivo context.xml

Dentro de una etiqueta <beans ....></beans> definiremos todos los objetos que se crearán al inicio de la aplicación para posteriormente ser inyectados en las clases que los requieran.

Esto se puede hacer mediante constructor tal como muestra este ejemplo:

```
<bean id="motor" class="es.autos.Motor">
    <constructor-arg index="0" value="diesel" type="java.lang.String"/>
    <constructor-arg index="1" value="100" type="int"/>
</bean>
```

O por getter and setter según este otro ejemplo:

```
<bean id="vehiculo" class="es.autos.Vehiculo">
    <property name="marca" value="Ford"/>
    <property name="modelo" value="Fiesta"/>
    <property name="motor" ref="motor"/>
</bean>
```

## Configurar el contenedor para la inyección de dependencias

Spring coloca las clases necesarias para la creación del contenedor en los siguientes paquetes:

org.springframework.context  
org.springframework.beans

Estos paquetes contienen dos interfaces encargados de realizar la instanciación de objetos:

BeanFactory: representan el tipo de contenedor de beans más simple.

ApplicationContext: agrega más funcionalidades específicas para ciertos productos empresariales.

Una vez especificados todos los beans que vamos a poder utilizar a lo largo de nuestro proyecto, para poder inyectarlos, tendremos que crear una instancia de ApplicationContext a partir del xml.

```
ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
```

A partir de la instancia de la clase ApplicationContext podemos acceder a los diferentes beans que vamos a utilizar en nuestra aplicación.

```
Vehiculo vehiculo = (Vehiculo) (context.getBean("vehiculo"));
```

## Inyección de dependencias automático (Autowiring)

```
<property name="motor" ref="motor" />
```

Con el atributo ref="motor" indicamos que la propiedad motor del objeto vehículo apuntará al objeto motor creado anteriormente. Aunque el nombre de esta propiedad se llame igual al objeto creado anteriormente, podría tener diferente nombre.

El autowiring nos permite setear las dependencias de un bean de manera automática. Puede tomar estos valores:

No: ningún autowiring es ejecutado.

byName: buscará un bean cuyo nombre sea igual a una de las propiedades del bean.

byType: buscará un bean cuyo tipo sea igual al tipo de una de las propiedades del bean.

```
<bean id="vehiculo" class="es.autos.Vehiculo" autowire="byName">  
    <property name="marca" value="Ford"/>  
    <property name="modelo" value="Fiesta"/>  
</bean>
```

En este ejemplo no especificamos valor para la propiedad motor, pero al asignar el valor byName a la propiedad autowire, automáticamente la propiedad motor del vehículo será igual al objeto motor creado anteriormente.

También es posible poner un autowire por defecto en la etiqueta beans así:

```
default-autowire="byName"
```



## Ejercicio propuesto

En este ejercicio vamos a crear un proyecto simple que representa una **Gasolinera**. Cada Gasolinera contará con un **Empleado**. Cada Empleado está a cargo de un **Coche** o ninguno y podrá llenarlo de gasolina.

Se debe crear la jerarquía de clases completa mediante la inyección de dependencias de Spring a partir de entradas `<bean> ....</bean>` en el archivo de contexto de Spring.

En la clase App accederemos a nuestra objeto Gasolinera y daremos la orden al Empleado de la misma para que llene el Coche que tiene asignado.

**NOTA:** Las acciones que vayan realizando los diferentes objetos pueden especificarse en el proyecto como mensajes en la consola.

Partiremos del siguiente modelo de clases:

```
package es.gasolinera;

public class Coche {
    private String matricula;
    private String marca;
    private String modelo;

    public Coche(String matricula, String marca, String modelo) {
        super();
        this.matricula = matricula;
        this.marca = marca;
        this.modelo = modelo;
    }

    public String getMatricula() {
        return matricula;
    }
    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }
    public String getMarca() {
        return marca;
    }
    public void setMarca(String marca) {
        this.marca = marca;
    }
    public String getModelo() {
        return modelo;
    }
    public void setModelo(String modelo) {
        this.modelo = modelo;
    }
}
```

```
package es.gasolinera;

public class Empleado {
    private String nombre;
    private String tlf;
    private Coche coche;

    public Empleado(String nombre, String tlf) {
        super();
        this.nombre = nombre;
        this.tlf = tlf;
        this.coche = null;
    }

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getTlf() {
        return tlf;
    }
    public void setTlf(String tlf) {
        this.tlf = tlf;
    }
    public Coche getCoche() {
        return coche;
    }
    public void setCoche(Coche coche) {
        this.coche = coche;
    }
}
```

```
package es.gasolinera;

public class Gasolinera {
    private String direccion;
    private Empleado empleado;

    public Gasolinera(String direccion) {
        super();
        this.direccion = direccion;
        this.empleado = null;
    }

    public String getDireccion() {
        return direccion;
    }
    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }
    public Empleado getEmpleado() {
        return empleado;
    }
    public void setEmpleado(Empleado empleado) {
        this.empleado = empleado;
    }
}
```