Spring con anotaciones

Podemos utilizar anotaciones para minimizar el código del archivo XML de configuración de los beans de Spring. Para ponerlo en práctica puedes hacer una copia del primer proyecto que creaste con el nombre “hola”.

Ejemplo HolaMundo con Spring y anotaciones

El primer paso será eliminar la entrada <bean> y añadir la línea resaltada en rojo.

```
<?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">

       <context:component-scan base-package="es.hola"/>
</beans>
```

Con la línea añadida estamos especificando que los beans están configurados mediante anotaciones en las propias clases. En la propiedad base-package estamos especificando el paquete que contiene los beans. Ninguna clase que esté fuera del paquete mencionado o sus subpaquetes, será reconocida.

La clase HolaMundo quedaría así:

```
package es.hola;

import org.springframework.stereotype.Service;

@Service("holita")
public class HolaMundo {
       public void decirHola() {
              System.out.println("Hola mundo desde el Spring con anotaciones");
       }
}
```

Con la anotación @Service estamos indicando que se trata de un servicio de Spring con el nombre de objeto (calificador) “holita”.

La clase App no cambia con respecto a la versión anterior.

Proyecto autos (Motor y Vehiculo) con anotaciones (inyección de dependencias)

```java
package es.autos;

import org.springframework.stereotype.Service;

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

	public Motor() {
		this.tipo = "Gasolina";
		this.caballos = 100;
	}

	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;
	}
}
```

```java
package es.autos;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

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

	public Vehiculo() {
		this.marca = "Ford";
		this.modelo = "Fiesta";
	}

	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;
        }

        @Autowired
        public void setMotor(Motor motor) {
                this.motor = motor;
        }
}
```

La clave de la inyección de dependencias en este ejemplo está en la siguiente anotación:

```
@Autowired
```

No se puede especificar el tipo de búsqueda del bean (byName, byType), se realiza automáticamente la búsqueda por tipo.

El 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">

        <context:component-scan base-package="es.autos"/>

        <context:annotation-config/>

</beans>
```

Al añadir <context:annotation-config /> a la configuración de Spring, estamos activando la conexión de beans (inyección de dependencias por medio de anotaciones). Activa las anotaciones en beans ya registrados.

Con la línea <context:component-scan base-package="es.autos"/> configuramos Spring para que detecte de forma automática los beans y los declare automáticamente por nosotros sin tener que abrir las etiquetas <bean> … </bean> para cada uno de ellos.

Para que una clase sea detectada automáticamente como bean, debe ser anotada utilizando una de estas anotaciones:

@Component	componente de uso general.
@Controller	controlador MVC de Spring.
@Repository	repositorio de datos.

@Service	servicio
@Otra	cualquier anotación personalizada que extienda @Component

Resolviendo ambigüedades en la inyección de dependencias

Vamos a dejar de nuevo la clase Motor sin anotar y con un constructor que reciba dos argumentos, tal como estaba en la primera versión.

```
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;
	}
}
```

También vamos a cambiar el archivo de configuración de la siguiente manera:

```
<?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">

	<context:component-scan base-package="es.autos"/>
	<bean id="motor1" 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="motor2" class="es.autos.Motor">
		<constructor-arg index="0" value="gasolina" type="java.lang.String"/>
		<constructor-arg index="1" value="100" type="int"/>
	</bean>
	<context:annotation-config/>
</beans>
```

Hemos creado dos objetos de la clase Motor llamados motor1 y motor2 configurados dentro del archivo context.xml.

El objeto vehículo será creado automáticamente por estar anotado con @Service.

Al intentar hacer la inyección de dependencias con @Autowired, arroja un error que trae el siguiente texto:

```
No qualifying bean of type [taller.Motor] is defined: expected single matching bean but found 2: motor1,motor2
```

Está claro que el contexto de Spring no sabe cuál de los dos objetos queremos inyectar.

La solución está en la anotación @Qualifier.

```
	@Autowired
	@Qualifier("motor2")
	publicvoidsetMotor(Motor motor) {
		this.motor = motor;
	}
```

Uso de configuración basada en clases Java

Es posible crear los beans dentro de un archivo de configuración anotado con @Configuration

```
package es.autos;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class CrearBeans {
	@Bean
	public Motor motor1() {
		return new Motor("Gasolina", 100);
	}

	@Bean
	public Motor motor2() {
		return new Motor("Diesel", 95);
	}

	@Bean
	public Vehiculo vehiculo() {
		Vehiculo v = new Vehiculo();
		v.setMarca("Suzuki");
		v.setModelo("Ignis");

		// Inyección de dependencias.
		v.setMotor(motor2());

		return v;
	}
}
```

Cada uno de estos métodos equivale a una entrada de tipo <bean> … </bean> en el archivo de contexto de Spring.

El bean obtiene su ID del nombre del método y todo lo que suceda dentro de la implementación del método conduce a la creación del bean que será retornado.

El fichero de configuración quedaría así:

```
<?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">

       <context:component-scan base-package="es.autos"/>

</beans>
```

En las clases Motor y Vehiculo eliminaremos todas las anotaciones.

Ejercicio práctico sobre Spring 1

Mediante Spring y la inyección de dependencias, crea los beans fiesta (de tipo Fiesta) y local (de tipo Local). La fiesta tendrá un método para celebrar y otro para terminar la fiesta.

La fiesta se celebra en un local, habrá que inyectar un local a la fiesta.

Declara los beans y realiza la inyección de dependencias mediante XML.

Ejercicio práctico sobre Spring 2

Realiza una copia del ejercicio anterior minimizando el código del archivo XML. Declara los beans y gestiona la inyección de dependencias por medio de anotaciones.

Ejercicio práctico sobre Spring 3

Realiza una copia del ejercicio anterior y actualiza el proyecto para registrar los beans por medio de una clase anotada con @Configuration