Acabamos de ver como usar Java equals y hashcode

Spring Singleton vs Prototype es una de las preguntas clásicas de Spring Framework y a muchas personas les surgen dudas sobre como funciona el scope o ámbito de los beans que generamos. Vamos a intentar aclararlo con un ejemplo sencillo. Supongamos que tenemos la siguiente clase de servicio con su interface.

```
package com.arquitecturajava;

public interface ServicioTareas {
 public void lanzar();
}

package com.arquitecturajava;

public class ServicioTareasImpl implements ServicioTareas{

public void lanzar() {

 System.out.println("Tarea1");
 System.out.println("Tarea2");
 System.out.println("Tarea3");

 }

}
```

Se trata de una sencilla clase que ejecuta tres tareas. La vamos a dar de alta en el

application-context.xml.

```
&amp;lt;?xml version="1.0" encoding="UTF-8"?&amp;gt;
&amp;lt;beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans.xsd"&gt;

 &amp;lt;beans&amp;gt;

 &amp;lt;bean id="tareas"
class="com.arquitecturajava.ServicioTareasImpl" /&amp;gt;

 &amp;lt;/beans&amp;gt;

&amp;lt;/beans&amp;gt;
```

Hecho esto podemos en un programa main instanciar la clase de servicio con Spring Framework y ejecutarla.

```
package com.arquitecturajava;

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

```
public class Principal {

public static void main(String[] args) {

ApplicationContext contexto = new ClassPathXmlApplicationContext(
 "/application-context.xml");
 ServicioTareas tareas = contexto.getBean(ServicioTareas.class);
 tareas.lanzar();

}

}
```

El programa se ejecutará y mostrará por pantalla las distintas tareas:

```
tarea1
tarea2
tarea3
```

Sin embargo es evidente que se trata de una situación poco elegante. Sería mucho mas flexible diseñarlo con dos conceptos ServicioTareas y Tarea.

Vamos a ver como quedarían las clases y los interfaces:

```
package com.arquitecturajava;

public interface ServicioTareas2 {
 public void lanzar();
 public void add(Tarea t);

}
```

```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;

public class ServicioTareas2Impl implements ServicioTareas2{

private List&amp;lt;Tarea&amp;gt; lista= new
ArrayList&amp;lt;Tarea&amp;gt;();
 public void lanzar() {
 for(Tarea t:lista) {

 t.ejecutar();
 }
 }
 public void add(Tarea t) {

 lista.add(t);
 }
```

```
}
```

Como vemos el código es sencillo a nivel del Servicio ya que ahora almacenamos un ArrayList de tareas para mejorar la flexibilidad. Nos queda por ver el código fuente de la Tarea:

```
package com.arquitecturajava;

public interface Tarea {

 public void setMensaje(String mensaje);
 public String getMensaje();
 public void ejecutar();
}
```

```
package com.arquitecturajava;

public class TareaImpl implements Tarea {

 private String mensaje;

 public void ejecutar() {

 System.out.println(mensaje);
```

```
 }
 public String getMensaje() {
 return mensaje;
 }

 public void setMensaje(String mensaje) {
 this.mensaje = mensaje;
 }
```

Realizadas estas operaciones simplemente tenemos que dar de alta los beans en el application-context.xml.

```
&amp;lt;?xml version="1.0" encoding="UTF-8"?&amp;gt;
&amp;lt;beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans.xsd"&gt;

 &amp;lt;beans &amp;gt;

 &amp;lt;bean id="tareas"
class="com.arquitecturajava.ServicioTareasImpl"&amp;gt;

 &amp;lt;/bean&amp;gt;
 &amp;lt;bean id="tarea"
class="com.arquitecturajava.TareaImpl"/&amp;gt;
 &amp;lt;/beans&amp;gt;

&amp;lt;/beans&amp;gt;
```

Es momento de instanciar las clases desde un programa y ejecutar el código:

```
package com.arquitecturajava;

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

public class Principal02 {

 public static void main(String[] args) {

 ApplicationContext contexto = new ClassPathXmlApplicationContext(
 "/application-context.xml");
 ServicioTareas2 tareas = contexto
 .getBean(ServicioTareas2.class);

 Tarea t1= contexto.getBean(Tarea.class);
 t1.setMensaje("tarea1");
 Tarea t2= contexto.getBean(Tarea.class);
 t2.setMensaje("tarea2");
 Tarea t3= contexto.getBean(Tarea.class);
 t3.setMensaje("tarea3");
 tareas.add(t1);
 tareas.add(t2);
 tareas.add(t3);
 tareas.lanzar();
 }

}
```

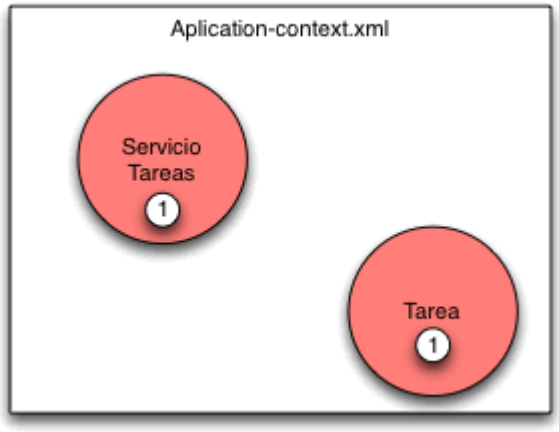
```
}
```

El resultado será el siguiente:

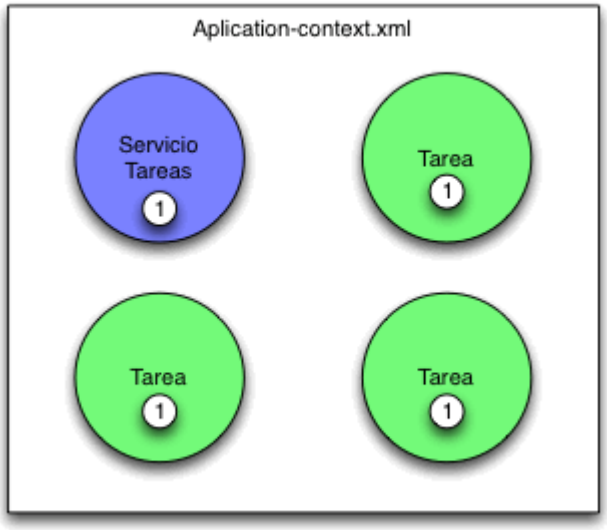
```
tarea3
tarea3
tarea3
```

Spring Singleton

Lamentablemente no es el que queríamos y tenemos un problema. Esto se debe a que todos nuestros beans son “singleton” es decir solo existe una única instancia de ellos .



Por lo tanto solo tenemos un objeto tarea que se ejecuta con el último texto. Para solventar estos temas tenemos que modificar el ámbito (scope) de la Tarea y convertirla en “prototype” de esta manera cada vez que pidamos un objeto tarea a Spring se creará un nuevo bean.



El código del application-context.xml será el siguiente:

```
&amp;lt;?xml version="1.0" encoding="UTF-8"?&amp;gt;
&amp;lt;beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd"&gt;

&amp;lt;beans &amp;gt;

&amp;lt;bean id="tareas"
class="com.arquitecturajava.ServicioTareas2Impl"
scope="prototype"&amp;gt;

&amp;lt;/bean&amp;gt;
&amp;lt;bean id="tarea" class="com.arquitecturajava.TareaImpl"
scope="prototype"/&amp;gt;
&amp;lt;/beans&amp;gt;
```

```
&amp;lt;/beans&amp;gt;
```

Ahora el resultado será el correcto:

```
tarea1
tarea2
tarea3
```

La mayor parte de las veces usaremos el scope de Singleton que es el scope por defecto. Sin embargo en otras situaciones podemos querer usar el scope prototype que nos asegura que se generará una nueva instancia cada vez que solicitemos el bean a Spring. El scope de prototype es habitual cuando deseamos mantener estado.

Otros artículos relacionados:

Spring Organización

Spring XML vs Anotaciones