Acabamos de ver como usar Java equals y hashcode

Spring es uno de los frameworks más utilizados ya que permite una gran flexibilidad a la hora de configurarlo. Una de las características menos conocidas del framework es el uso de Profiles. Los Profiles o perfiles permiten configurar grupos de elementos del framework para un “perfil” de ejecución predeterminado. Vamos a verlos a traves de un ejemplo sencillo. Para ello usaremos Maven y las siguientes dependencias:

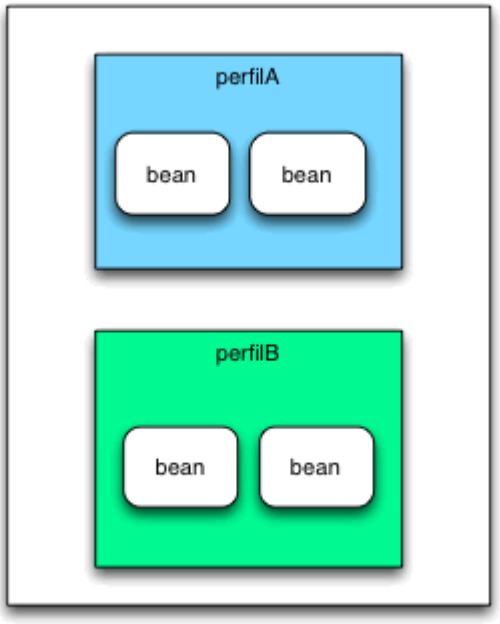
```
&amp;lt;dependency&amp;gt;
&amp;lt;groupId&amp;gt;org.springframework&amp;lt;/groupId&amp;gt;
&amp;lt;artifactId&amp;gt;spring-core&amp;lt;/artifactId&amp;gt;
&amp;lt;version&amp;gt;3.2.13.RELEASE&amp;lt;/version&amp;gt;
&amp;lt;/dependency&amp;gt;
&amp;lt;dependency&amp;gt;
&amp;lt;groupId&amp;gt;org.springframework&amp;lt;/groupId&amp;gt;
&amp;lt;artifactId&amp;gt;spring-context&amp;lt;/artifactId&amp;gt;
&amp;lt;version&amp;gt;3.2.13.RELEASE&amp;lt;/version&amp;gt;
&amp;lt;/dependency&amp;gt;

&amp;lt;dependency&amp;gt;
&amp;lt;groupId&amp;gt;org.springframework&amp;lt;/groupId&amp;gt;
&amp;lt;artifactId&amp;gt;spring-beans&amp;lt;/artifactId&amp;gt;
&amp;lt;version&amp;gt;3.2.13.RELEASE&amp;lt;/version&amp;gt;
&amp;lt;/dependency&amp;gt;
```

Realizado este primer paso nos definimos un applicationContext.xml con dos Beans:

```
&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 profile="perfilA"&amp;gt;

&amp;lt;bean id="mensaje"
class="com.arquitecturajava.ServicioMensajeA"&amp;gt;
&amp;lt;/bean&amp;gt;
&amp;lt;/beans&amp;gt;
&amp;lt;beans profile="perfilB"&amp;gt;

&amp;lt;bean id="mensaje"
class="com.arquitecturajava.ServicioMensajeB"&amp;gt;
&amp;lt;/bean&amp;gt;
&amp;lt;/beans&amp;gt;
&amp;lt;/beans&amp;gt;
```

Spring Profiles

Como podemos ver se trata de un fichero muy sencillo pero que esta configurado con dos “Profiles” diferentes.



De esta forma nosotros podemos configurar que perfil es el que Spring Framework carga.
En este caso tenemos dos clases de Servicio (A y B) que implementan el mismo interface.

```
package com.arquitecturajava;

public interface ServicioMensaje {
public String mensaje();
}
```

```
package com.arquitecturajava;
public class ServicioMensajeA implements ServicioMensaje{

public String mensaje() {
```

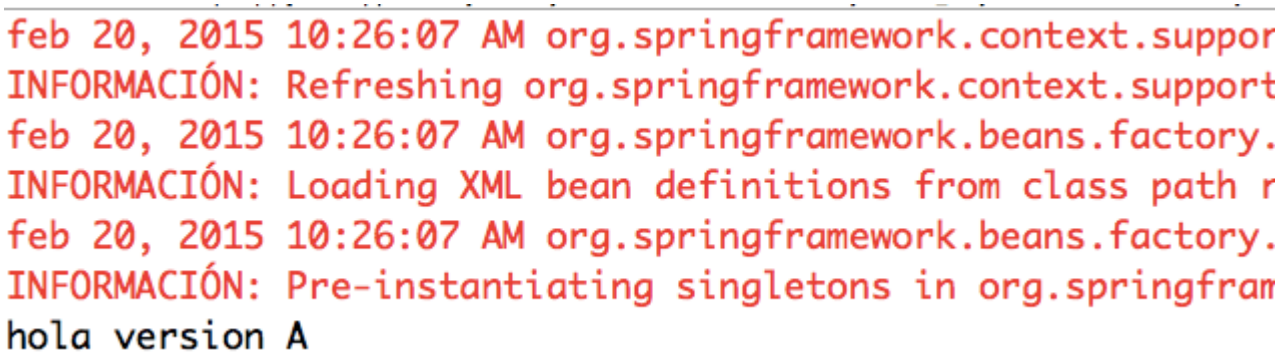
```
return "hola version A";
}
}

&amp;amp;nbsp;



package com.arquitecturajava;

public class ServicioMensajeB implements ServicioMensaje{

 public String mensaje() {

 return "hola version B";
 }
}
```

Podemos decidir que clases se cargan dependiendo del profile. Para ello usaremos una propiedad del Sistema y pasaremos a Spring Framework que profile necesitamos.

```
package com.arquitecturajava;

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

public class Principal {
public static void main(String[] args) {
```

```
System.setProperty("spring.profiles.active", "perfilA");

ApplicationContext contexto = new
ClassPathXmlApplicationContext("/application-context.xml");
ServicioMensaje mensaje = (ServicioMensaje)
contexto.getBean(ServicioMensaje.class);
System.out.println(mensaje.mensaje());
}

}
```

El resultado de configurar el “perfilA” como Profile activo será que por pantalla nos mostrará :



La configuración de profiles nos permitirá una gran flexibilidad como por ejemplo cargar unos properties u otros y decidir si estamos en producción o desarrollo o decidir que usamos una autentificación en memoria o vía LDAP etc.

Otros artículos relacionados:

Spring PropertyPlaceHoldersConfigurar

Spring responsabilidad y Organización