Acabamos de ver como usar Java equals y hashcode

El uso de Spring @ComponentScan es uno de los más habituales cuando trabajamos con una aplicación de Spring . Spring se encarga de instanciar todos nuestros objetos y por lo tanto actua como una super Factoria que construye objetos . Vamos a ver un poco a detalle cuales son las opciones de Spring @ComponentScan cuando trabajamos con anotaciones.

Spring @ComponentScan

Lo primero que vamos a construir es un ejemplo con Maven de como Spring a nivel de consola es capaz de instanciar su Factoría y crear los diversos objetos.

```
<dependencies>

		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>5.2.5.RELEASE</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.springframework/spring-core -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-core</artifactId>
			<version>5.2.5.RELEASE</version>
		</dependency>

	</dependencies>
```

Una vez tenemos las dependencias de Spring configuradas en nuestro proyecto el siguiente paso es crear un programa Main. Este se encargará de instanciar la Factoría de Spring para crear los objetos que necesitamos.

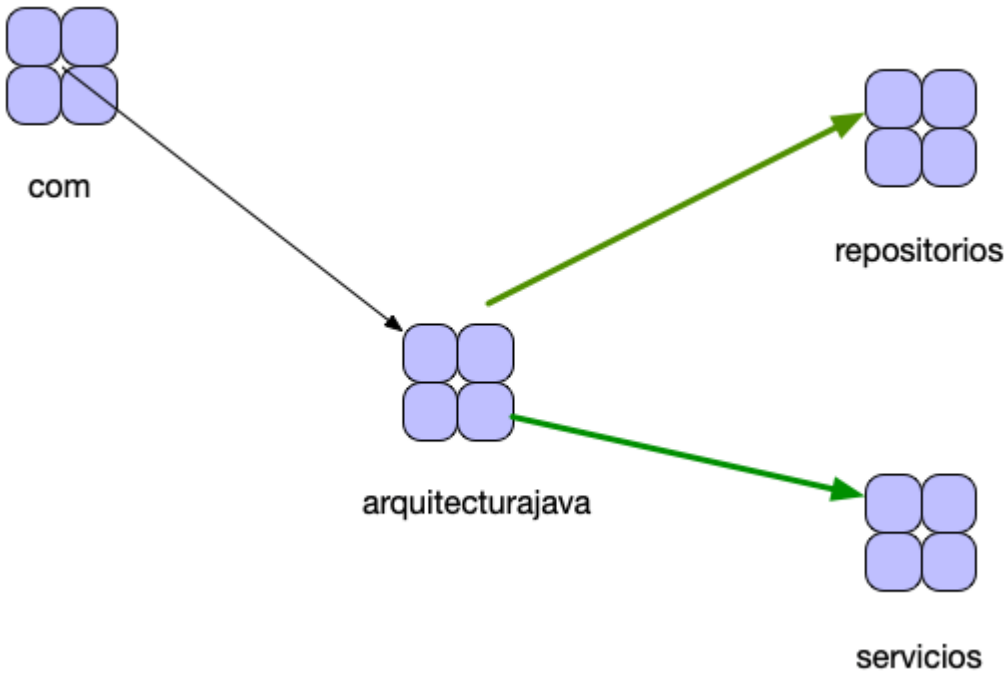
```
package com.arquitecturajava.servicios;

import
org.springframework.context.annotation.AnnotationConfigApplicationCont
ext;

public class Principal {

	public static void main(String[] args) {
		AnnotationConfigApplicationContext factoria = new
AnnotationConfigApplicationContext();
		factoria.register(SpringConfigurador.class);
		factoria.refresh();
	}

}
```

Una vez realizada esta operación podemos observar como la factoría registra el SpringConfigurador que es una clase de configuración de Spring con la cual decidimos que clases se registran utilizando @ComponentScan

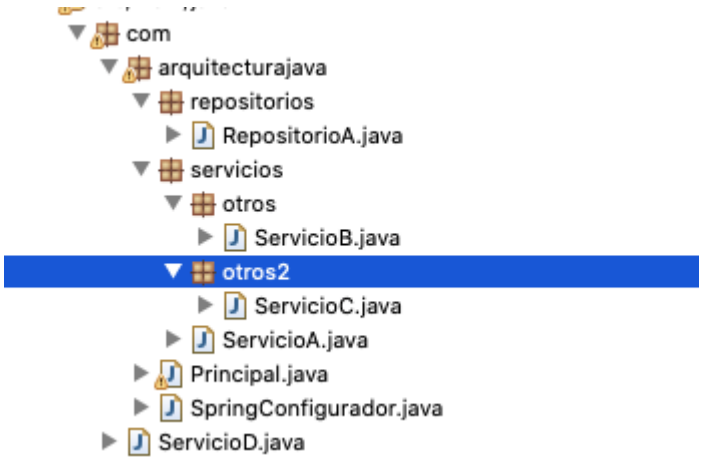
```
package com.arquitecturajava.servicios;

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

@Configuration
@ComponentScan("com.arquitecturajava")
public class SpringConfigurador {
}
```

En este caso hemos usado @ComponentScan(“com.arquitecturajava”) por lo tanto registrará

los paquetes que se encuentren a nivel de com.arquitecturajava y todos los hijos.



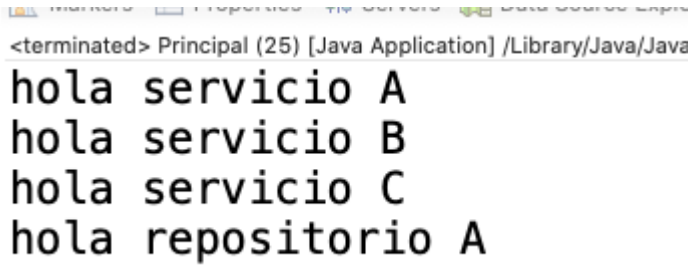
Por lo tanto si tenemos una estructura de packages como esta:



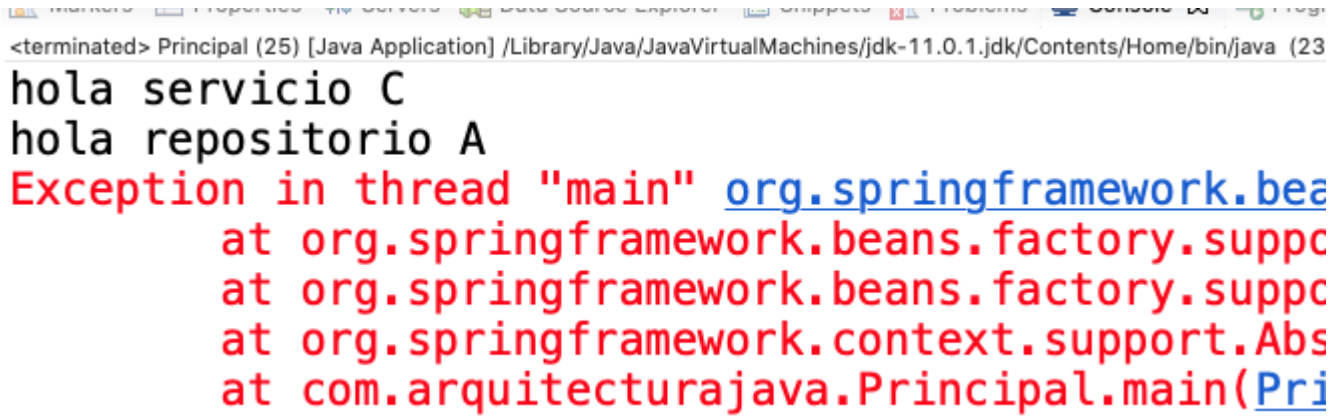
Podremos acceder a las clases que se encuentra dentro de com.arquitecturajava. Es decir tanto ServicioA como ServicioB como ServicioC son accesibles . No solo eso sino que también podemos acceder a RepositorioA ya que se encuentra dentro de com.arquitecturajava . Un programa de consola quedaría así:

```
package com.arquitecturajava;

import
org.springframework.context.annotation.AnnotationConfigApplicationCont
ext;

import com.arquitecturajava.repositorios.RepositorioA;
import com.arquitecturajava.servicios.ServicioA;
import com.arquitecturajava.servicios.otros.ServicioB;
import com.arquitecturajava.servicios.otros2.ServicioC;

public class Principal {

        public static void main(String[] args) {
                AnnotationConfigApplicationContext factoria = new
AnnotationConfigApplicationContext();
                factoria.register(SpringConfigurador.class);
                factoria.refresh();
                ServicioA servicio=factoria.getBean(ServicioA.class);
                System.out.println(servicio.mensaje());
                ServicioB servicio2=factoria.getBean(ServicioB.class);
                System.out.println(servicio2.mensaje());
                ServicioC servicio3=factoria.getBean(ServicioC.class);
                System.out.println(servicio3.mensaje());

                RepositorioA
repositorioA=factoria.getBean(RepositorioA.class);
                System.out.println(repositorioA.mensaje());
        }

}
```

Pudiendo acceder a cada una de estas clases e imprimiendo un mensaje con el servicio que es por la consola



¿Hay alguna clase a la que no podamos acceder ? . Si, la clase que se encuentra dentro del package com (ServicioD) . Esa clase no esta accesible desde nuestro Spring @ComponentScan y por lo tanto si intentamos instanciarla el programa fallará:

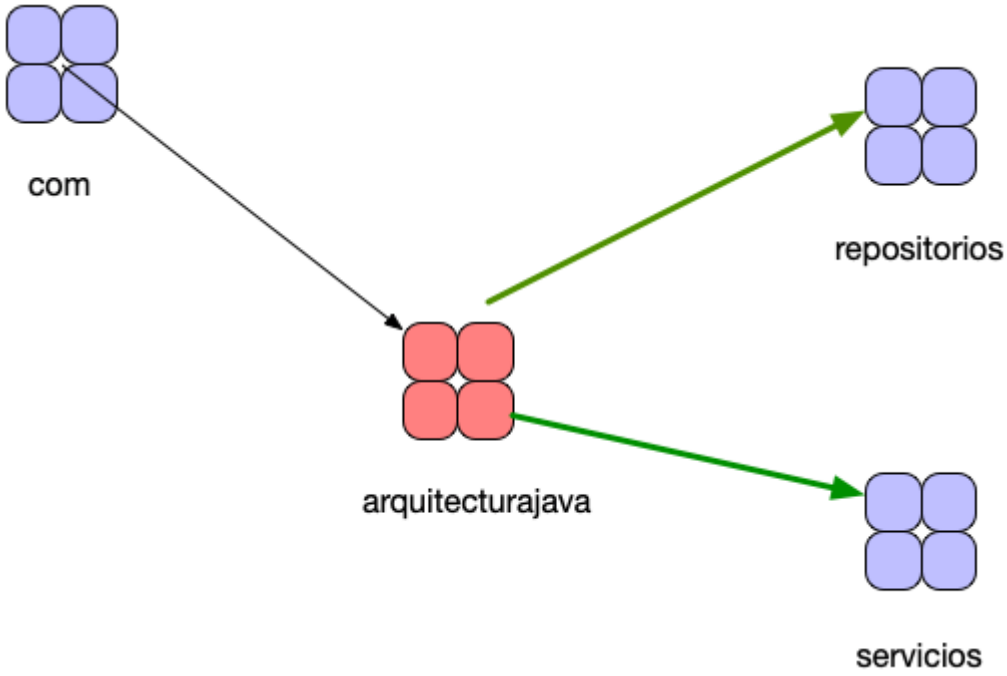


Es así como funciona el sistema de scaneo. En muchas ocasiones me encuentro con situaciones en las cuales a nivel de SpringConfigurador se apoya en * como se muestra a continuación:

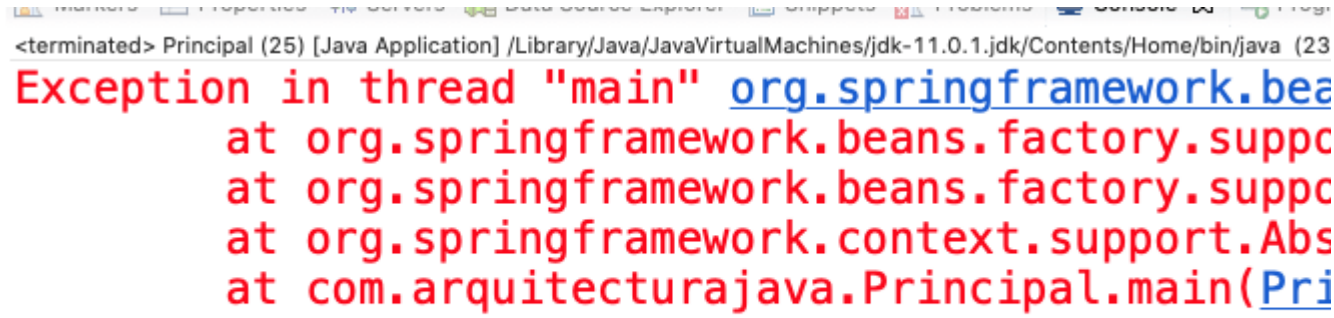
```
package com.arquitecturajava;

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

@Configuration
```

```
@ComponentScan("com.arquitecturajava.servicios.*")
public class SpringConfigurador {
}
```

En este caso se realizará un scaneo de los packages que son hijos de com.arquitecturajava.servicios . Esto nos generará un problema ya que el servicioA se encuentra dentro del package de “servicios” y no dentro de uno de sus hijos .



Por lo tanto Spring no será capaz de instanciarlo



Spring falla nada más arrancar ya que el primer servicio no puede ser localizado. El manejo de Spring @ComponentScan tiene muchas variantes por ejemplo podríamos tener una configuración como la siguiente:

```
package com.arquitecturajava;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.FilterType;

@Configuration
@ComponentScan(value="com.arquitecturajava.servicios"
,useDefaultFilters = false, includeFilters = @ComponentScan.Filter
(type = FilterType.REGEX, pattern = ".*[AB]"))
public class SpringConfigurador {
}
```

En este caso hemos elegido todos los componentes que están dentro del package de servicios pero que terminen por la letra A o la letra B . Por lo tanto si ejecutamos el programa Main los dos primeros servicios podrán ser instanciados en cambio el C fallara.

```
hola servicio A
hola servicio B
Exception in thread "main" org.springframework.bea
        at org.springframework.beans.factory.suppo
        at org.springframework.beans.factory.suppo
        at org.springframework.context.support.Abs
        at com.arquitecturajava.Principal.main(Pr
```

Aprendamos a usar mejor Spring @ComponentScan y nos permitirá ganar en flexibilidad a la hora de registrar Servicios, Repositorios etc.

Otros artículos relacionados

1. ¿Que es Spring Boot?
2. ¿Que es Spring WebFlux?
3. Spring WebInitializer
4. Spring Framework