El uso de Spring Testing es algo cada día más importante ya que poco a poco las metodologías de TDD se van imcorporando más en los desarrollos . Spring es un framework de Inyección de dependencia y para hacer aunque se la prueba unitaria más sencilla del mundo necesitaremos preconfigurar algunas cosas antes de poder ejecutarla sobre Spring Framework. El primer paso será configurar el proyecto de Maven para poder disponer de las dependencias de Spring.

Acabamos de ver como usar Java equals y hashcode

```
        <dependencies>
                <!--
https://mvnrepository.com/artifact/org.springframework/spring-core -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-core</artifactId>
                        <version>5.2.9.RELEASE</version>
                </dependency>
                <!--
https://mvnrepository.com/artifact/org.springframework/spring-context
-->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-context</artifactId>
                        <version>5.2.9.RELEASE</version>
                </dependency>

                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                        <version>4.13</version>
                        <scope>test</scope>
```

```
                </dependency>

                <!--
https://mvnrepository.com/artifact/org.springframework/spring-test -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-test</artifactId>
                        <version>5.2.9.RELEASE</version>
                        <scope>test</scope>
                </dependency>

        </dependencies>
```

En este caso hemos añadido tanto las dependencias de Spring como las dependencias de Junit. El siguiente paso es generar un Servicio que tenga la anotación @Service. Este servicio devolverá una lista de personas.

```
package com.arquitecturajava.testing;

public class Persona {

        private String nombre;
        private int edad;
        public String getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
        public int getEdad() {
                return edad;
        }
```

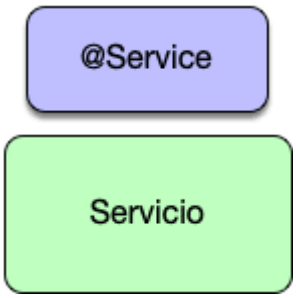
```
	public void setEdad(int edad) {
		this.edad = edad;
	}
	public Persona(String nombre, int edad) {
		super();
		this.nombre = nombre;
		this.edad = edad;
	}
}
```

Vamos a ver el código del Servicio:


```
package com.arquitecturajava.testing;

import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Service;

@Service
public class Servicio {

	public List<Persona> buscarTodas() {
```

```
                Persona p = new Persona("pepe", 20);
                Persona p2 = new Persona("ana", 30);

                List<Persona> lista = Arrays.asList(p, p2);
                return lista;
        }
}
```

Como vemos hemos añadido la anotación de @Service lo que convierte a la clase de Servicio en un Servicio de Spring Framework a disposición del inyector de dependencia.

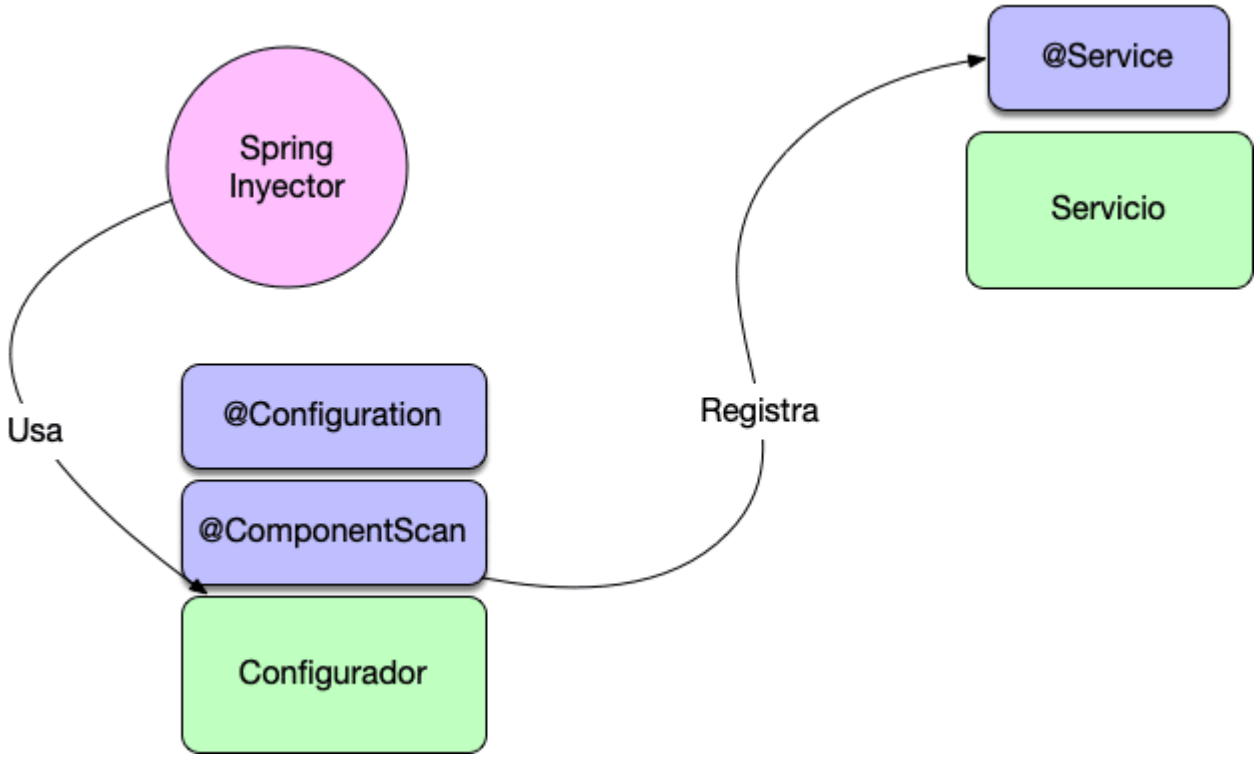


Spring Testing y configuración

Es momento de generar una clase de Configuración que sea capaz de localizar el Servicio para posteriormente inyectarlo a nivel de prueba unitaria.

```
package com.arquitecturajava.testing;

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

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

En este caso la anotación de @ComponentScan se encarga de localizar el Servicio y registrarlo.

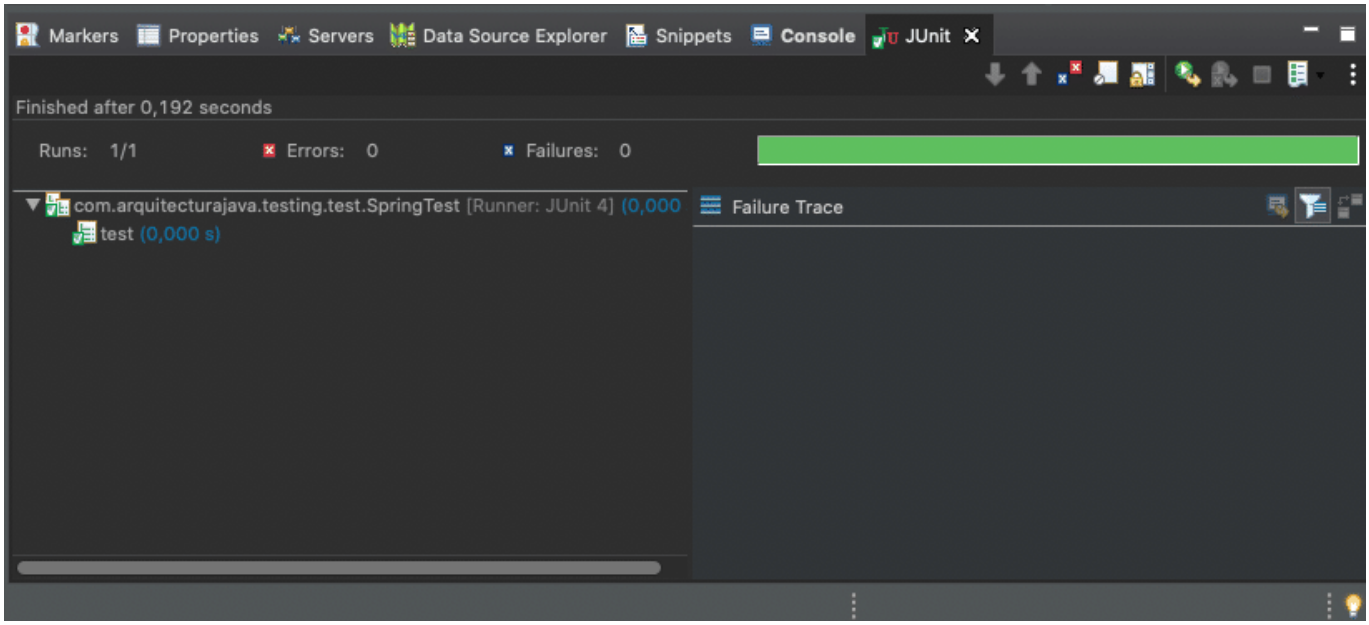


El paso que nos queda es crear la prueba unitaria y ejecutarla.

```
package com.arquitecturajava.testing.test;

import static org.junit.Assert.*;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import
org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.arquitecturajava.testing.ConfiguradorSpring;
```

```
import com.arquitecturajava.testing.Servicio;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = { ConfiguradorSpring.class })
public class SpringTest {

        @Autowired
        private Servicio miservicio;
        @Test
                public void test() {
assertEquals(2,miservicio.buscarTodas().size());
                }

}
```

En este caso hay que usar la anotación @RunWith que nos permite ejecutar pruebas unitarias sobre el paraguas de Spring Framework . Evidentemente para poder hacerlo necesitaremos declarar cual es el Contexto de Spring que vamos a utilizar en este caso ConfigurarSpring.class que se encarga de cargar en memoria la clase de Servicio. Hecho esto ejecutamos la prueba unitaria y acabamos de pasar el test más básico apoyandonos en Spring.



Cada día es más importante el uso de pruebas unitarias en el desarrollo de nuestras aplicaciones.

Otros artículos relacionados

- Spring @ComponentScan y configuración
- Spring @Autowired y la inyección de dependencias
- Spring @Transactional readonly y optimizaciones
- Curso TDD