Tabla de Contenidos

- Servicios e Impuestos
  - Tipos de IVA e implementación
- Spring @Qualifier
- Otros artículos relacionados:

Acabamos de ver como usar Java equals y hashcode

Spring @Qualifier es una de las anotaciones más prácticas de Spring Framework cuando queremos añadir versatilidad a como realizamos un @Autowired en nuestros componentes. Vamos a ver un ejemplo sencillo para ello el primer paso va a ser generarnos un proyecto con Spring Boot y su configuración por defecto. Aquí podemos ver las dependencias de Maven.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>com.arquitecturajava</groupId>
	<artifactId>cualificadores</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>jar</packaging>

	<name>cualificadores</name>
	<description>Demo project for Spring Boot</description>

	<parent>
```

```
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>1.5.10.RELEASE</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>

	<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncod
ing>
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-
test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
<groupId>org.springframework.boot</groupId>
```

```
                                <artifactId>spring-boot-maven-
plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

</project>
```

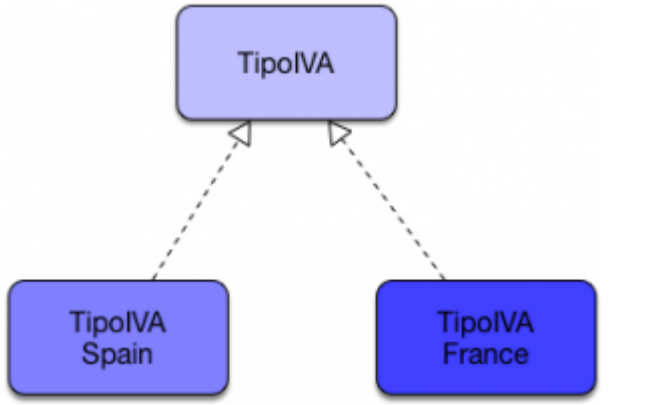
Una vez realizado el primer paso imaginemonos que tenemos dos Servicios y cada uno de estos servicios se va a encargar de gestionar los impuestos orientados a un país en concreto.





Servicios e Impuestos

Es muy probable que aunque tengamos dos Países existan un montón de impuestos en común uno de ellos es habitualmente el IVA. En España nosotros tenemos un IVA del 21% pero en Francia podría ser un IVA del 25% por poner un ejemplo. Así pues nos puede venir bien definir un interface TipoIVA que soporte varias implementaciones . Cada una de ellas asociada un Pais.



Tipos de IVA e implementación

Normalmente cuando esto sucede debemos construir el interface y las implementaciones.

```
package com.arquitecturajava.cualificadores;

public interface TipoIVA {

        public double calcular(double importe) ;
}
```

```
package com.arquitecturajava.cualificadores;

import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component
@Qualifier("france")
public class TipoIVAFrance  implements TipoIVA{
```

```
	@Override
	public double calcular(double importe) {
		// TODO Auto-generated method stub
		return importe *1.25;
	}

}
```

```
package com.arquitecturajava.cualificadores;

import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

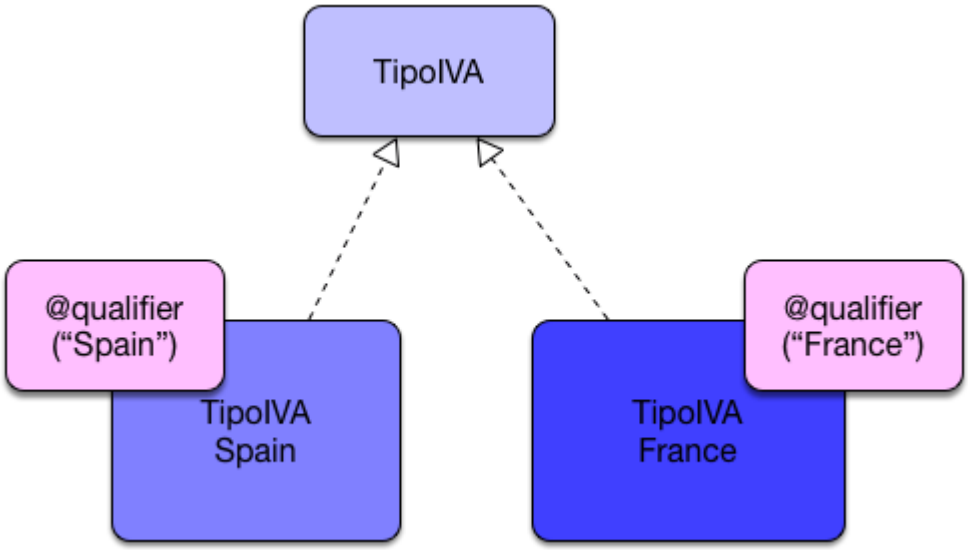
@Component
@Qualifier("spain")
public class TipoIVASpain  implements TipoIVA{

	@Override
	public double calcular(double importe) {
		// TODO Auto-generated method stub
		return importe *1.21;
	}

}
```

Spring @Qualifier

En este caso hemos usado la anotación Qualifier para diferenciar cada una de las implementaciones que tenemos del concepto de TipoIVA.



El siguiente paso es crear las clases de Servicio correspondiente , así como una sencilla clase de negocio que nos ayude a definir los gastos .

```
package com.arquitecturajava.cualificadores;

public class Gasto {

	private double importe;

	public double getImporte() {
		return importe;
	}

	public void setImporte(double importe) {
```

```
                this.importe = importe;
        }

        public Gasto(double importe) {
                super();
                this.importe = importe;
        }

}
```

```
package com.arquitecturajava.cualificadores;

import java.util.List;

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

@Service
public class ServicioImpuestosFrance {

        @Autowired
        @Qualifier("france")
        TipoIVA tipoIva;

        public TipoIVA getTipoIva() {
                return tipoIva;
        }
```

```
	public void setTipoIva(TipoIVA tipoIva) {
		this.tipoIva = tipoIva;
	}

	public double
gastoConIva(List&amp;amp;amp;lt;Gasto&amp;amp;amp;gt; gasto) {

		return gasto.stream().mapToDouble((x)-
&amp;amp;amp;gt;tipoIva.calcular(x.getImporte())).sum();

	}

}
```

```
package com.arquitecturajava.cualificadores;

import java.util.List;

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

@Service
public class ServicioImpuestosSpain {

	@Autowired
	@Qualifier("spain")
	TipoIVA tipoIva;
```

```
        public TipoIVA getTipoIva() {
                return tipoIva;
        }

        public void setTipoIva(TipoIVA tipoIva) {
                this.tipoIva = tipoIva;
        }

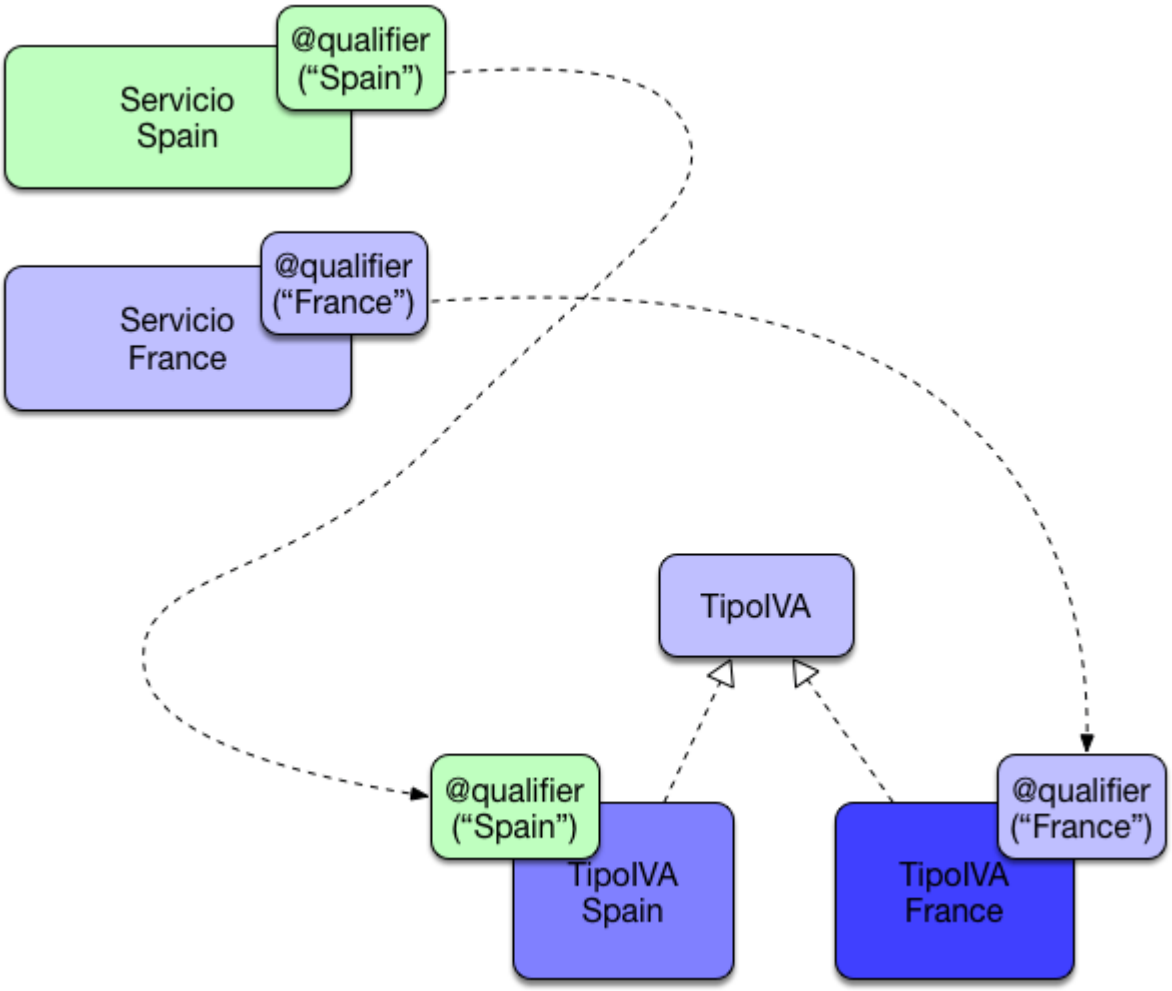
        public double
gastoConIva(List&amp;amp;amp;lt;Gasto&amp;amp;amp;gt; gasto) {

                return gasto.stream().mapToDouble((x)-
&amp;amp;amp;gt;tipoIva.calcular(x.getImporte())).sum();

        }

}
```

Como podemos ver cada servicio realiza un @Autowired del Tipo de Iva que necesita apoyándose en la anotación de @Qualifier



De esta forma cuando usemos los servicios en un programa de consola. Cada servicio hará un cálculo de IVA apoyándose en la implementación que considere más adecuada para su País. Vamos a ver el código del main.

```
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CualificadoresApplication implements CommandLineRunner{

	@Autowired
```

```
        ServicioImpuestosFrance servicioA;

        @Autowired
        ServicioImpuestosSpain servicioB;

        public static void main(String[] args) {
                SpringApplication.run(CualificadoresApplication.class,
args);
        }

        @Override
        public void run(String... arg0) throws Exception {

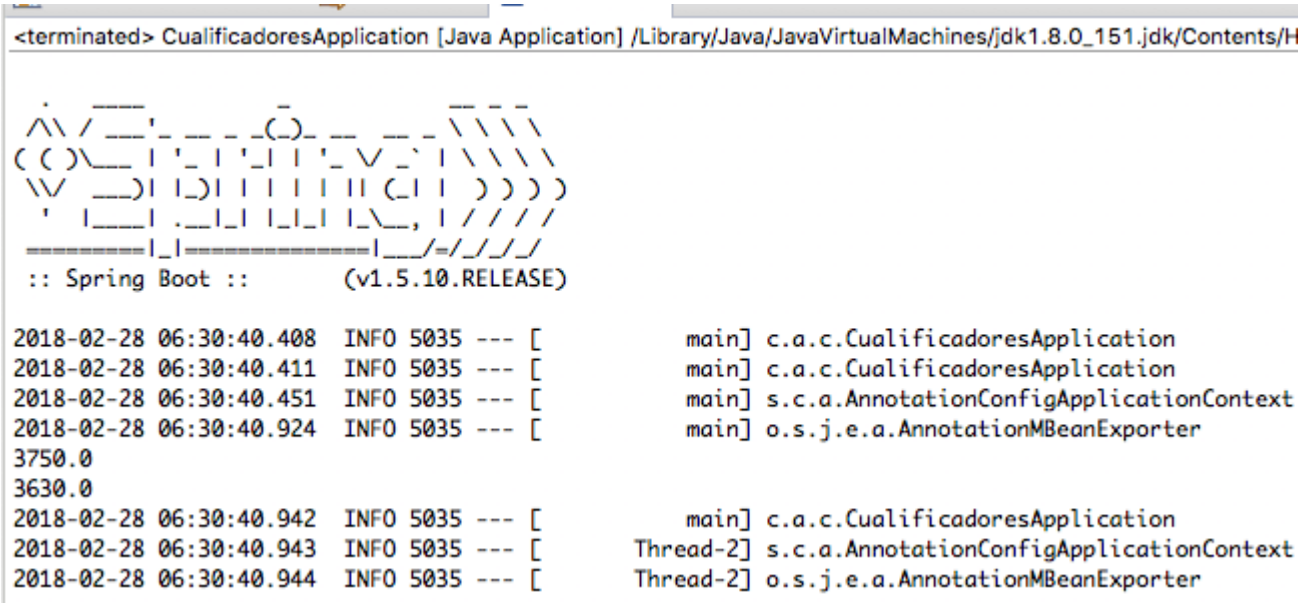
                List&amp;amp;amp;lt;Gasto&amp;amp;amp;gt; milista= new
ArrayList&amp;amp;amp;lt;Gasto&amp;amp;amp;gt;();

                milista.add(new Gasto(1000));
                milista.add(new Gasto(2000));

                System.out.println(servicioA.gastoConIva(milista));
                System.out.println(servicioB.gastoConIva(milista));

        }
}
```

Acabamos de configurar los servicios con spring @Qualifier es momento de ejecutar el programa y ver el resultado en la consola:



Cada IVA se calcula apoyándose en el algoritmo de Pais.

Otros artículos relacionados:

1. Spring @import , organizando Spring framework
2. Spring AOP y Aspectos
3. Spring Boot @Lazy Components

Enlaces externos

Anotaciones de Spring Framework