Framework SpringBoot

Índice

1. Java Annotations.

Antes de crear formularios HTML que trabajen con datos dinámicos facilitados por el usuario (Comprobando a partir de validaciones), o incluso usando datos almacenados en un BD, es necesario saber cómo SpringBoot inyecta la información en los objetos sin necesidad de instanciarlos uno a uno en la clase Main.

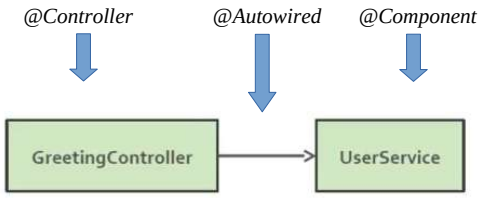
Annotations: se ayudan de metadatos para especificar una funcionalidad al lenguaje. Ya conocemos del 1° curso de Programación --> **@Override:** se usa para forzar al compilador a comprobar en tiempo de compilación que estás sobrescribiendo correctamente un método, y de este modo evitar errores en tiempo de ejecución, los cuales serían mucho más difíciles de detectar.

En cada apartado veremos algunas nuevas como:
@Component: nombre para vincular el bean de la clase (Se puede poner nombre a continuación de @Component). Si tenemos varios beans es mejor dejar el @Component sin nombre, entonces automáticamente vinculará el nombre de la clase pero empezando por minúsculas.

@Autowired: agrega, automáticamente mediante injection, las dependencias de otros componentes. Esta anotación se utiliza para 'inyectar' una instancia de una clase dentro de otra clase

@Qualifier: Sirve para inyectar una clase cuando 2 o más clases implementan la misma Interface.

Como puedes observar más abajo, el módulo “GreetingController” depende del módulo “UserService” (Los módulos pueden crearse con la anotación @Component, con @Controller, entre otros….), siendo tarea del framework instanciar objetos de ambos módulos. A partir de la anotación @Autowired se crea dicha inyección de dependencia (Se inyecta un módulo a otro).




```
@Controller
public class GreetingController {

  @Autowired
  private UsersService usersService;

  @RequestMapping("/greeting")
  public String greeting(Model model) {

    model.addAttribute("name",
       usersService.getNumUsers()+" users");

    return "greeting_template";
  }
}
```

```
@Component
public class UsersService {

  public int getNumUsers(){
    return 5;
  }
}
```

Vamos a ver un ejemplo usando @Component, @Autowired, @Qualifier: https://www.youtube.com/watch?v=RpKBt1mrgKY&ab_channel=MitoCode

En temas posteriores veremos otras como:

Tema 3: @Controller, @RequestMapping, @GetMapping, @PostMapping, @PutMapping

Tema 4: @Entity, @Id, @GeneratedValue, @DeleteMapping

Tema 5: @Secured, @{/logout}

1.1. @Component

A nivel de Spring Framework esta anotación simplemente registra un bean dentro del framework.

EJERCICIO a realizar.

Vamos a crear un ejemplo sencillo para ver como se automatiza el proceso. Para ello:

1. Crea un nuevo proyecto con el starter de Spring llamado Tema3.
2. Copia el package `proyectovehiculos` y renombralo a `proyectovehiculosmejorado.`
3. Copia el archivo XML del proyecto anterior y borra cada unos de los bean, dejando solo la etiqueta `<beans>...</beans>`
4. Inserta el siguiente código a tu XML para que el proyecto Spring reconozca las annotations:

```
<!--Nombre del package donde están las clases-->
```

```
<context:component-scan
base-package="proyectovehiculosmejorado"></context:component-scan>
```

5. Crea una nueva Interface llamada VehiculoMejorado con 1 atributo y 2 métodos:
   int numPasajeros=2;
   public String getNombre();
   public int getNumPasajeros();
6. Crea una clase llamada NaveEspacial que hereda la Interface VehiculoMejorado para probar las Java Annotations. Los métodos tendrán lo necesario para mostrar el mensaje: "Halcón Milenario tiene un total de 6 astronautas."
7. Creamos la clase Main llamada PruebaVehiculoMejorado siguiendo los pasos que realizabamos en los proyectos Legacy.
   9.1. Paso 1: Cargar archivo XML:

```
ClassPathXmlApplicationContext miContexto = new
ClassPathXmlApplicationContext("applicationContext.xml");
```

   9.2. Paso 2: Creamos la Java Annotation de tipo **@Component** en la clase NaveEspacial (Recuerda que puedes ponerle nombre o dejarlo vacío)
   9.3. Paso 3: Hacemos la llamada al bean y lo usamos.
   Debe aparecer por pantalla: Halcón Milenario tiene un total de 6 astronautas.
   9.4. Paso 4: Cerramos el bean.

1.2. @Autowired: Inyección a partir de constructor, campo y método

Spring @Autowired es una de las anotaciones más habituales cuando trabajamos con Spring Framework ya que se trata de la anotación que permite inyectar unas dependencias con otras dentro de Spring .

En este apartado vemos lo fácil que es realizar una inyección de dependencia con esta annotation en comparación a utilizar el XML de configuración. Esta Java Annotations se encarga de inyectar directamente la dependencia.
Los pasos a seguir son:

1. Crea la Clase TransporteAuxiliar y la Interface PasajerosExtra a inyectar (Dependencia)
   En la Interface añade el método: public int getPasajerosExtra(); //Cuya implementación en la clase recien creada será de 8 pasajeros.
2. Crea un constructor por defecto y crea el setter para la inyección en la clase NaveEspacial a partir del atributo:
   private PasajerosExtra pasajerosExtra_NaveDos;

3. Configura la dependencia con @Autowired:
   Añade en la Clase TransporteAuxiliar la annotation @Component
   Autowired usando constructor: Vamos a realizar la inyección de dependencia desde el constructor: Crea un constructor con el atributo creado anteriormente. Añadele encima del constructor la anotación @Autowired.

4. Modifica el método:

```
public int getNumPasajeros() {
        return pasajerosExtra_NaveDos.getPasajerosExtra()+numPasajeros;
}
```

5. Lanza la clase Main (Se han debido inyectar esos 8 pasajeros extra a los 2 ya iniciales gracias a @Autowired): Debe aparecer "`Halcón milenario tiene un total de 10 astronautas`".

6. **Autowired usando campo de clase:** Prueba a comentar/borrar el constructor y añade encima de PasajerosExtra la anotación @Autowired. E*l framework va a buscar **automáticamente** si existe alguna clase que implemente la Interface PasajerosExtra. Si la encuentra, es de esa clase de donde obtiene la inyección de dependencia.*

7. **Autowired usando método:** Para ver que realmente se está inyectando el campo "`pasajerosExtra_NaveDos`" deja comentado @autowired y lanza la clase Main: Dará error al no encontrar el método para construir / dar valor al objeto --> `Exception in thread "main" java.lang.NullPointerException`

Crea el método setPasajerosExtra_NaveDos y añadele la anotación @Autowired. Realiza otra vez la prueba. La inyección funciona debido a que encuentra el método que realiza la inyección de dependencia.

Hemos visto en el apartado 6 que se puede conseguir sin crear un método. ¿Qué opción usamos entonces? Depende del número de constructores, métodos setter y campos creados. (Intentaremos balancear el número de apariciones de dichas opciones para generar un código más limpio)

1.3. @Qualifier

Expongamos el siguiente caso: Dos o más clases implementan la misma Interface ¿Qué clase debe usar SpringBoot para inyectar la dependencia? --> Esto se soluciona usando @Qualifier.

Antes de ver un ejemplo, lee la siguiente explicación:
https://www.arquitecturajava.com/spring-qualifier-utilizando-autowired/

Los pasos a seguir son:

1. Crea la Clase TransporteAuxiliarDos_Qualifier implementando la Interface PasajerosExtra a inyectar (Dependencia)
   Modifica el método: public int getPasajerosExtra() // Este nuevo transporte puede llevar 2 pasajeros más que el anterior. De esta forma veremos que método se lanza cada vez.

2. Añade en la Clase TransporteAuxiliarDos_Qualifier la annotation @Component.

3. Lanza la clase Main PruebaVehiculoMejorado y observa el error. Se debe a que Spring no es capaz de saber qué método ha de llamar (Existen 2 clases con la misma Interface). El error de la consola es:
   ```
   No qualifying bean of type 'proyectovehiculosmejorado.PasajerosExtra' available:
   expected single matching bean but found 2:
   transporteAuxiliar,transporteAuxiliarDos_Qualifier
   ```

4. Especifica que quieres usar la clase nueva " TransporteAuxiliarDos_Qualifier ". Para ello en la clase NaveEspacial, añade a la inyección de dependencia (Ya sea por constructor, por campo o por método) la siguiente línea de código debajo de @Autowired:
   ```
   @Qualifier("transporteAuxiliarDos_Qualifier")
   ```

1.4. @Configuration (Sin archivo XML en nuestro proyecto)

Anotación encargada de definir que la clase es una clase de configuración para el framework. Con esto, podemos prescindir de hacer referencia al XML (Legacy) sustituyéndolo por un archivo .class

Recordemos la línea que vamos a eliminar:

```
//Paso 1: Cargar archivo XML:
ClassPathXmlApplicationContext miContexto = new
ClassPathXmlApplicationContext("applicationContext.xml");
```

1. Crea una clase llamada Config, la cual será nuestro archivo de configuración.
2. Añádele la anotación @Configuration a esa clase.
   Ahora es el momento de realizar en nuestra nueva clase Config la misma tarea que realizaba applicationContext.xml, recordemos que era:
   ```
   <!--Nombre del package donde están las clases-->
   ```

```
<context:component-scan base-package="proyectovehiculosmejorado">
</context:component-scan>
```

3. Usa la notación @ComponentScan("proyectovehiculosmejorado") para especificar a Spring donde está el paquete a usar.
4. Recordemos nuevamente la línea que usabamos con XML:

```
//Paso 1: Cargar archivo XML:
ClassPathXmlApplicationContext miContexto = new
ClassPathXmlApplicationContext("applicationContext.xml");
```

Ahora será:

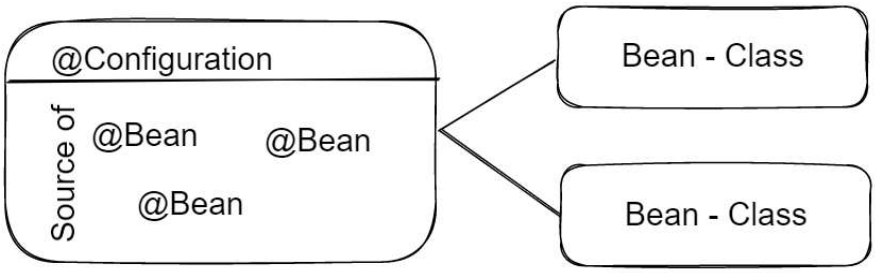
```
//Paso 1: Leer el class de configuración:
AnnotationConfigApplicationContext miContexto = new
AnnotationConfigApplicationContext(Config.class);
```

A partir de ahora podrías borrar el archivo applicationContext.xml, o si lo prefieres, déja todo el código de ese .xml comentado y prueba a lanzar la aplicación otra vez. ¿Funciona?

1.5. @Bean

Un Bean es un objeto gestionado por el framework Spring. En Spring Boot, @Bean es una anotación a nivel de método que se usa para declarar y registrar un bean en el contenedor de Spring.

Cuando una clase está anotada con @Configuration, se trata como una fuente de definiciones de beans, y cualquier método dentro de la clase que esté anotado con @Bean se utilizará para crear y configurar beans.



Es el momento de usar @Bean para pedir beans al contenedor de Spring sin usar archivos XML. Los pasos a seguir son:

1. Creamos una clase con el nombre “TransbordadorEspacialConAnotacionBean” que implementa la Interface “VehiculoMejorado”
2. Realiza la implementación de los métodos de la Interface con la información que quieras.
3. Añade un atributo de la Interface “PasajerosExtra” y un constructor con dicho atributo.

4. Creamos otra clase “MasPasajerosBean” que implementa la Interface `PasajerosExtra` añadiendo 3 pasajeros a su método.
5. Vamos a la clase creada en el ejemplo anterior “Config” donde se usaba @Configuration para poder prescindir de hacer referencia al XML (Legacy). Definiremos 2 beans.
6. El primer bean será de la clase “MasPasajerosBean”:

```
@Bean
public MasPasajerosBean pasajerosParaTransbordadorEspacial() {
          return new MasPasajerosBean();
}
```

7. El segundo bean será de la clase “TransbordadorEspacialConAnotacionBean” donde inyectaremos la dependencia de MasPasajerosBean.

```
@Bean
public TransbordadorEspacialConAnotacionBean transbordadorEspacialConAnotacionBean()
{    return new
TransbordadorEspacialConAnotacionBean(pasajerosParaTransbordadorEspacial());
    } // pasajerosParaTransbordadorEspacial es el id del bean que se inyecta
```

8. Creamos con la clase Main “`PruebaVehiculoMejorado2PruebaBeanTransbordadorEspacial`” para realizar las pruebas:

```
package proyectovehiculosmejorado;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class PruebaVehiculoMejorado2PruebaBeanTransbordadorEspacial {

          public static void main(String[] args) {

          //Paso 1: Leer el class de config:
          AnnotationConfigApplicationContext miContexto = new
AnnotationConfigApplicationContext(Config.class);

          // Paso 2: Hacemos la llamada al bean y lo usamos:
          VehiculoMejorado transbordador =
miContexto.getBean("transbordadorEspacialConAnotacionBean", VehiculoMejorado.class);

          System.out.println(transbordador.getNombre()+" tiene un total de
"+transbordador.getNumPasajeros()+" astronautas.");

          miContexto.close(); //Paso 3: Cerramos el bean.
          }
}
```

1.5.1 Usando desde una clase POJO la annotation @Bean

Podemos usar @Bean directamente desde una clase auxiliar POJO. Copia y lee el código facilitado. Crea una clase auxiliar llamada Persona.

```
package com.example.demo;
```

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

public class Persona {

	private String nombre;
	private String edad;
	List list = new ArrayList<>();

	public Persona(String nombre, String edad, List list) {
		super();
		this.nombre = nombre;
		this.edad = edad;
		this.list = list;
		System.out.println("gestor creado: " + getNombre() + " " + getEdad() + " "
+ list.toString());
	}

	public Persona(String nombre, String edad) {
		this.nombre = nombre;
		this.edad = edad;
		System.out.println("gestor creado: " + getNombre() + " " + getEdad());
	}

	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	public String getEdad() {
		return edad;
	}

	public void setEdad(String edad) {
		this.edad = edad;
	}
}
```

```
public class PruebaBean {
  public static void main(String[] args) {
    SpringApplication.run(PruebaBean.class, args);
  }
  @Bean
  public Persona metodoUno() {
```

```
            List<Object> list = new ArrayList<>(20);
            list.add("Vive en Sevilla.");
            list.add(" Es programadora.");
    return new Persona("Eva", "33", list);

  }

  @Bean
  public Persona metodoDos() {
    return new Persona("Paco", "37");
  }
}
```

1.6. @PropertySource y @Value.

Terminamos con estas 2 anotaciones con las cuales prescindimos aún más de archivos XML.

Nuestro objetivo es recuperar información de un archivo externo:

- @PropertySource: Permite especificar la ruta al archivo externo de propiedades.
- @Value: Permite inyectar el valor de las propiedades desde el archivo externo.

Los pasos para usar estas anotaciones son:

1. Crear un archivo externo con la información que necesitamos para nuestros atributos. Crea un arhivo txt llamado MiVehiculo en la ruta src/main/java con par clave valor. Ejemplo:
   ```
   velocidad=61.500 km/h
   precio=250000000
   ```
2. Copia la clase del ejemplo anterior “TransbordadorEspacialConAnotacionBean” y renómbrala con “`TransbordadorEspacialConAnotacionBeanEjemploPropertySourceyValue`” Crea los 2 atributos que has definido en el txt.
3. En la clase usada en el ejemplo anterior llamada “Config” añadimos la propiedad @PropertySource junto al nombre del archivo MiVehiculo para indicar donde están guardadas estas propiedades o atributos:
   ```
   @PropertySource("classpath:MiVehiculo.txt")
   ```
4. Es el momento de inyectar esos 2 atributos. Creamos ahora los atributos de arriba descritos en la clase “`TransbordadorEspacialConAnotacionBeanEjemploPropertySourceyValue`”. En esos 2 atributos se inyectarán los valores a partir de @Value:
   ```
   @Value("${nombreAtributo}")     // tipo y nombre de tu atributo    ← línea a añadir
   private String velocidad;       // línea creada en el paso 2
   ```
5. Creamos los métodos getter de esos 2 atributos con generate getters.
6. Ahora usaremos un @bean de esta clase. Para ello, en la clase Config, comenta el bean del ejemplo anterior y añade un bean de esta nueva clase:

```
//    @Bean
//    public TransbordadorEspacialConAnotacionBean
transbordadorEspacialConAnotacionBean() {
//        return new
TransbordadorEspacialConAnotacionBean(pasajerosParaTransbordadorEspacial());
//        } // pasajerosParaTransbordadorEspacial es el id del bean que se inyecta
```

```
//

@Bean
          public TransbordadorEspacialConAnotacionBeanEjemploPropertySourceyValue
transbordadorEspacialConAnotacionBean() {
                    return new
TransbordadorEspacialConAnotacionBeanEjemploPropertySourceyValue(pasajerosParaTransbord
adorEspacial());
          }
```

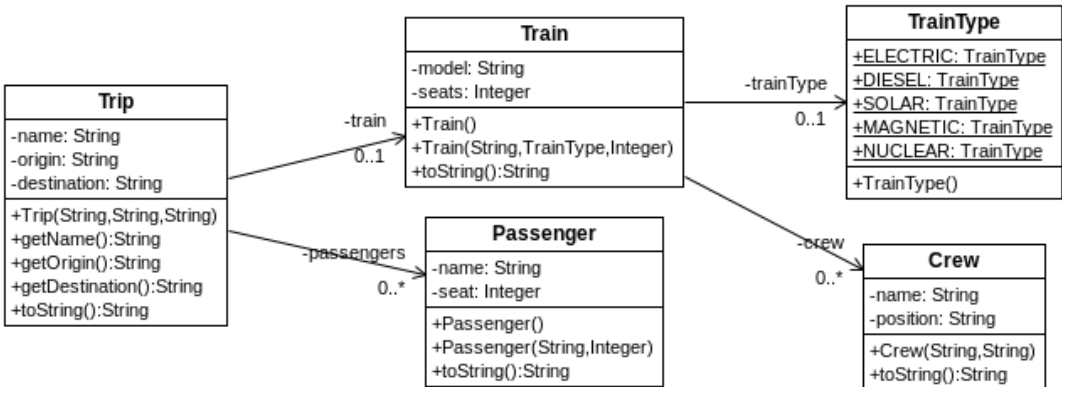
7. Creamos una clase Main llamada PruebaPropertySource_Value con únicamente 4 líneas de código para probar estas propiedades:

```
AnnotationConfigApplicationContext miContexto = new
AnnotationConfigApplicationContext(Config.class);

NombreClase transbordador = miContexto.getBean("Nombre@Bean", NombreClase.class);

System.out.println("El vehículo tiene "+transbordador.getAtributo1() +" y tiene
"+transbordador.getAtributo());

miContexto.close();
```

Ejercicio resuelto: Proyecto completo con uso de tipo Enum (TrainType).

Observa el esquema (En la siguiente página) y el código fuente (Subido a la plataforma Moodle) teniendo en cuenta que:

- Crew: personas de la tripulación del tren.
- Passenger: pasajeros del tren.
- Train: el tren que contiene la lista de la tripulación y una propiedad de tipo Enum: TrainType.
- Trip: el viaje, el cual se compone del resto, incluyendo una lista de pasajeros.



2. Tarea a realizar.

Por parejas (Si es posible) cread un ejercicio con todo lo aprendido hasta ahora. La temática es libre, intentad hacer uso de la mayoría de Java Annotations. Preparad una pequeña documentación con la explicación y capturas de la salida por pantalla.