Beans

¡Comencemos!

¿Qué aprenderás?

- Conocer la inversión de control (IoC).
- Conocer los beans en Spring.
- Integrar el uso de beans en Spring.
- Conocer el uso de anotaciones.
- Crear un proyecto utilizando Maven y Spring Tools Suite.

Introducción

Se enseñará lo que son los beans, cómo se utilizan y cómo se almacenan, además de crear nuestro primer proyecto basado en Spring. Desde aquí aprenderemos muchos nuevos conceptos con relación a Spring, donde se explicarán de manera detallada.

¡Vamos con todo!

¿Qué es IoC (Inversión de control)?

La inversión de control (Invertion of Control, en inglés), es un principio que maneja spring que permite inversión de dependencia sobre objetos, vale decir, cambia el concepto de que el programador tenga que incorporar y crear clases, se encarga una entidad externa de incluirlos en tiempo de ejecución.

Es un cambio en la forma de cómo se controla la aplicación, esto significa que por ejemplo, antes el programador era quien incorporaba clases u objetos (utilizando *new nombre_de_clase*), ahora se encarga spring de incorporarlos en tiempo de ejecución del proyecto.

Entonces, spring posee un contenedor de IoC y es él que se encarga de hacer la inyección de dependencias de las clases y objetos, sin la necesidad de escribir la llamada directamente en el código.

Este contenedor, además de crear los objetos, se encarga de relacionarlos, configurarlos y manejar su ciclo de vida completo (desde la creación hasta la destrucción de estos), es el núcleo de la aplicación.

¿Qué es un Beans?

Recordemos que no es más que un objeto (Plain Old Java Object, POJO) que almacena datos de nuestro programa, este es manejado por el contenedor de Spring, este objeto debe cumplir con lo siguiente:

- Debe tener un constructor por defecto (sin argumentos).
- Debe tener todas sus atributos privados.
- Debe tener para cada atributo su método getter y setter.

¿Cómo manejo un Beans?

Existen dos tipos de manejo de Beans por parte del contenedor de Spring, la primera forma de manejarlos es mediante un archivo *xml*, entonces, tenemos una clase Persona que tiene dos atributos nombre y edad,

```
package com.beansEjemplos.beansxml;
public class Persona {
   private String nombre;
   private int edad;
   public Persona(){}
   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 String toString() {
      return "Nombre: "+ this.nombre + ", Edad:"+ this.edad;
   }
```

```
}
```

Para conectarlo por *xml*, se debe crear el archivo beans para llamar a la clase, entonces, creamos nuestro archivo *beans.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
   <bean id="Persona" class="com.beansEjemplos.beansxml.Persona">
      <property name="nombre" value="Cristian XML" />
      <property name="edad" value="29" />
   </bean>
</beans>
```

Beans hace referencia a la clase que generé, hace la llamada a la ruta a la que está conectada y al nombre de la clase.

Esta etiqueta beans, posee propiedades, que no son más que los atributos de mi clase, el cual los manipula con los métodos *get* y *set*.

Los archivos fueron creados en base a la siguiente estructura:

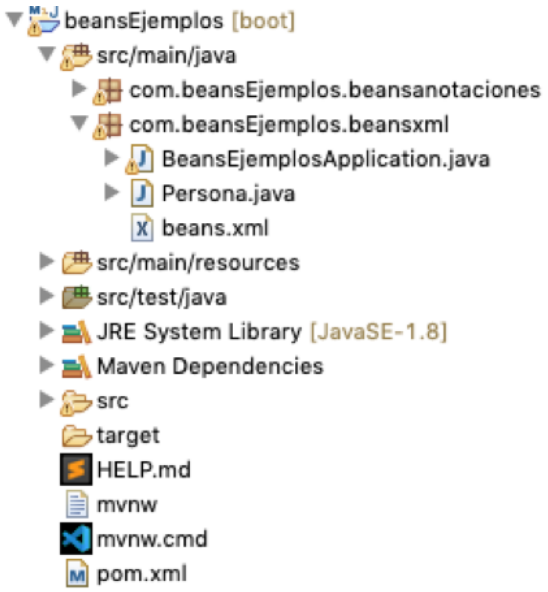

Imagen 1. Estructura de archivos.
Fuente: Desafío Latam.

Creamos el método Main (*BeansEjemplosApplication*) y conectamos todo.

```
package com.beansEjemplos;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import com.beansEjemplos.beansxml.Persona;
public class BeansEjemplosApplication {
   public static void main(String[] args) {
      ApplicationContext appContext = new
ClassPathXmlApplicationContext("com/beansEjemplos/beansxml/beans.xml");
      Persona p = appContext.getBean(Persona.class);
      System.out.println(p.toString());
   }
}
```

Si lo ejecutamos, obtendremos la siguiente salida:

Imagen 2. Resultado.
Fuente: Desafío Latam.

La segunda forma en que el contenedor de Spring maneja los beans es mediante anotaciones, entonces ¿qué son las anotaciones?

Anotaciones en Spring

Las anotaciones, son un conjunto de palabras reservadas que ayudan a manipular la aplicación, categorizando componentes de manera funcional.

Existen anotaciones denominadas *Spring Stereotypes*, que son las anotaciones maestras (anotaciones *core*) para categorizar los componentes importantes, de los cuales son solo cuatro.

- **@Component:** Indica que un objeto es de tipo componente. Es el general e informa a Spring que lo considere un componente.

- **@Controller:** Indica que un objeto es de tipo controlador, se encarga de controlar la comunicación entre el usuario y la aplicación.

- **@Repository:** Indica que es un objeto de tipo repositorio y se encarga de implementar un almacén de datos, o sea, es el que permite que se almacene información de una base de datos en el contenedor. Spring agrega otro tipo de servicios de bases de datos a este componente.

- **@Service:** Es el encargado de gestionar la operación de negocios, agrupa varias llamadas simultáneas a repositorios.

Existen otro tipo de anotaciones importantes:

- **@Configuration:** Indica que la clase posee la configuración principal del proyecto.

- **@EnableAutoConfiguration:** Indica que se aplicará la configuración automática.

- **@ComponentScan:** Ayuda a localizar elementos etiquetados con otras anotaciones.

- **@SpringBootApplication:** Engloba las tres anotaciones anteriores, @Configuration, @EnableAutoConfiguration y @ComponentScan.

- **@Autowired:** Spring realiza la inyección de dependencia sobre el atributo seleccionado.

- **@PathVariable:** Indica con qué variable de la URL se relaciona el parámetro que se está usando la anotación.

- **@Bean:** Indica que la clase seleccionada es un bean.

Para mayor información de las anotaciones, revisa el siguiente link.

Incorporando anotaciones

Basándonos en la clase persona propuesta anteriormente, haremos lo mismo pero con anotaciones.

Generamos nuestro archivo de configuración (*AppConfig.java*)

```
package com.beansejemplo2.beans;
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
@Configuration
public class AppConfig {
   @Bean
   public Persona persona() {
      return new Persona();
   }
}
```

Donde indicamos que la clase será de configuración, y el método que se encuentra en ella, será quien maneje el Bean.

Como podemos notar, esta es la única diferencia con respecto a la declaración de los Beans en XML, se reemplaza el archivo XML del ejercicio anterior a esta pequeña declaración.

Se tiene que cambiar el Main, donde ahora se llama a la clase *AppConfig*, para que maneje el bean.

```
package com.beansejemplo2.beans;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContex
t;
import com.beansejemplo2.beans.Persona;
import com.beansejemplo2.beans.AppConfig;
@SpringBootApplication
public class Ejemplo2Application {
   public static void main(String[] args) {
      ApplicationContext appContext = new
AnnotationConfigApplicationContext(AppConfig.class);
      Persona p = appContext.getBean(Persona.class);
      System.out.println(p.toString());
   }
}
```

Es importante mencionar, que la clase Persona también fue modificada, pero sólo para agregar valores a esta con la notación @Value la que permite agregar los valores correspondientes a los atributos. Quedando la clase de la siguiente manera:

```
package com.beansejemplo2.beans;
import org.springframework.beans.factory.annotation.Value;
```

```
public class Persona {
   @Value("Cristian Anotaciones")
   private String nombre;
   @Value("29")
   private int edad;
   Persona(){}
   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 String toString() {
      return "Nombre: "+ this.nombre + ", Edad:"+ this.edad;
   }
}
```

Obteniendo como resultado:

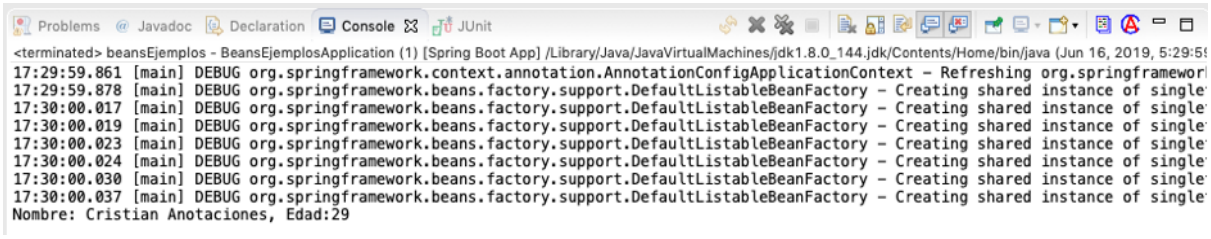

Imagen 3. Resultando usando anotaciones.
Fuente: Desafío Latam.

Para este ejemplo, la estructura de archivos quedó:

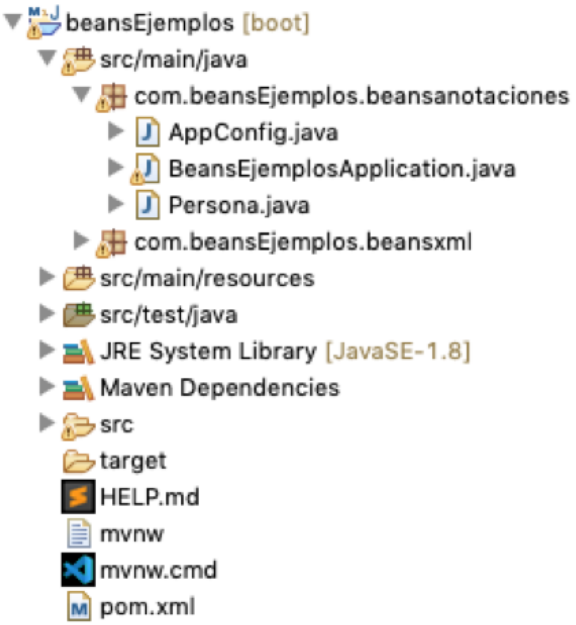

Imagen 4. Estructura resultante del proyecto.
Fuente: Desafío Latam.

Observaciones

Nos dimos cuenta, que la principal diferencia entre la estructura de archivos de beans mediante xml y la de beans mediante anotaciones son, el archivo que utilizan para conectar las respectivas clases de Persona.

En el primero se basa en un archivo XML con la configuración e información del bean mientras que en el segundo, se basa en un archivo Java que posee la configuración e información del bean. Y obviamente cambia desde el método Main la llamada que se hace en ambos casos.

Debido a esto es más engorroso utilizar ficheros XML en aplicativos grandes que tienen una gran cantidad de servicios, apareciendo desde la versión 2.5 de spring las anotaciones, para facilitarnos el trabajo.