15ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 7

Introducción de Aplicaciones Empresariales con Spring Framework.

Mtro. ISC Miguel Ángel Sánchez Hernández

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

DGTIC

Educación Continua
1971 - 2021

Objetivo

Aprender los diferentes tipos de configuración que nos ofrece Spring para un Bean así como su ciclo de vida en el contenedor

Lo que veremos

- Configuraciones por medio de XML
  - Constructores, Propiedades y Bean internos
  - Conectar Colecciones
  - Auto-conexiones
  - Métodos de Fabrica
- La vida de un Bean

Configuración de un Bean por XML

Un contenedor Spring IoC gestionara uno o más beans. Ocuparemos los metadatos que le proporcionaremos al contenedor por definiciones XML.

Los metadatos que contiene son:

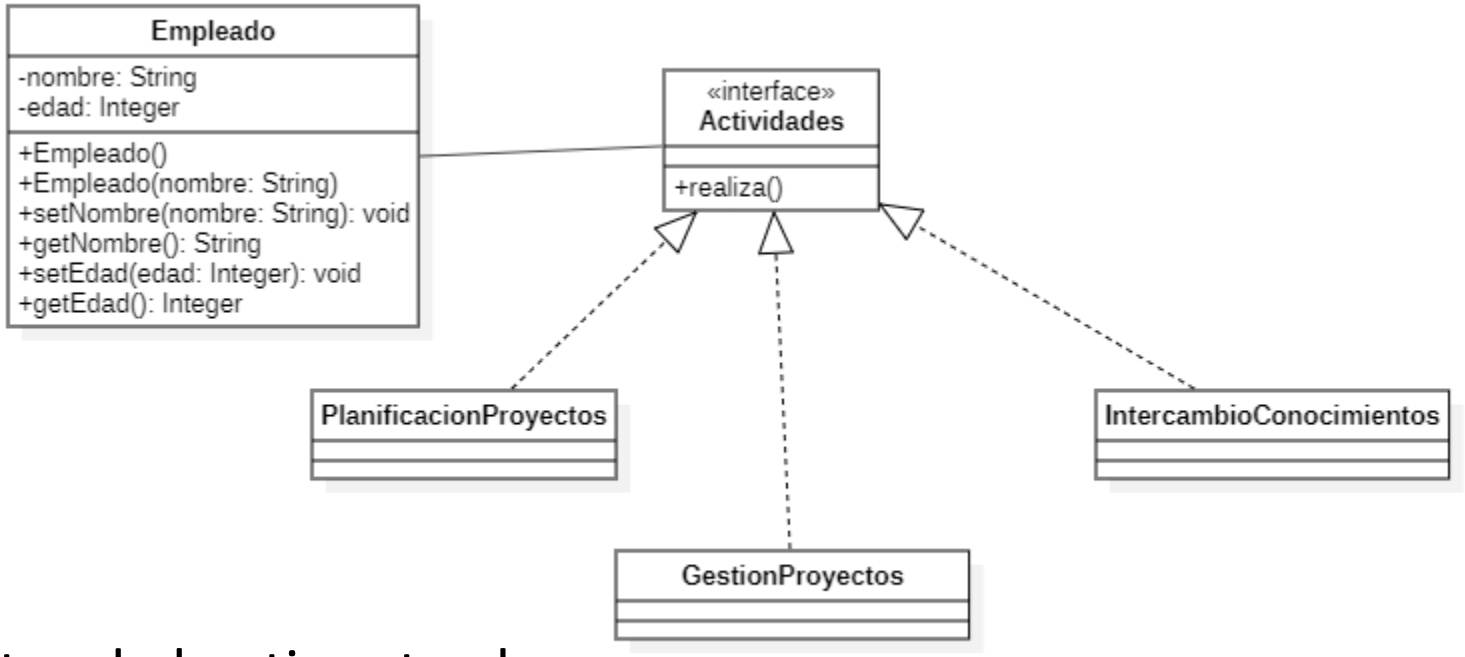
- **Nombre del paquete:** Ubicación de la clase de la implementación del Bean.
- **Configuración del comportamiento del bean:** Alcance, devoluciones del ciclo de vida etc.
- **Referencias:** Denominado también colaboraciones o dependencias, son las interacciones con los otros beans para realizar una lógica de negocio.
- **Ajustes de configuración:** Cantidad de conexiones de un Bean que administra unos grupos de conexiones.

Definición de un Bean

Atributos o elementos de la etiqueta <bean>

- **id o name:** Le da un nombre al bean por el que nos referiremos a él en el contenedor de Spring.
- **class:** Especificamos que tipo de bean será, especificando el paquete con el nombre de la clase.
- **<constructor-arg>:** Se utiliza para dar información a Spring de cómo construir al bean por medio de un constructor.
- **<property>:** Da la posibilidad de inyector valores llamando al método setter de la propiedad del bean.

Inyectar en etiqueta <bean>



1.- Elementos de la etiqueta <bean>

Ejercicio 5: Inyectar por constructores, propiedades y beans internos

Inyectar colecciones

Spring nos proporciona cuatro elementos de configuración de propiedades que nos necesarios cuando configuramos colecciones.

- **<list>:** Enlazar una lista de valores, permitiendo repetidos.
- **<set>:** Conecta unos conjuntos de valores, donde no se permiten repetidos.
- **<map>:** Enlaza una colección de pares del tipo nombre-valor, donde nombre es de cualquier tipo.
- **<props>:** Es igual que <map>, pero la diferencia que nombre y valor son ambos String.

Ejercicio 6: Conectar colecciones

Autoconexión

Hemos estado utilizando los elementos <constructor-arg> o <property> para conectar los beans, pero Spring nos ofrece tres maneras de hacer una Autoconexión con el atributo autowire.

- **byName:** Busca en el contenedor un bean cuyo nombre o id sea el mismo a la propiedad a conectar, si no se encuentra la propiedad no se conecta.
- **byType:** Intenta conectar un único bean en el contenedor cuyo tipo corresponda a la propiedad que se desea conectar. Si no se encuentra no se conecta el bean, pero si encuentra más de un bean lanza una excepción UnsatisfiedDependenciException.
- **constructor:** Busca en el contenedor uno o más beans con los parámetros del constructor del bean que se quiera conectar, si en encuentra beans ambiguos o contructores lanza una excepción UnsatisfiedDependenciException.

Autoconexión

Para entender mejor estos conceptos, la siguiente practica toca los puntos importantes de Autoconexión.

Ejercicio 7: Autowire

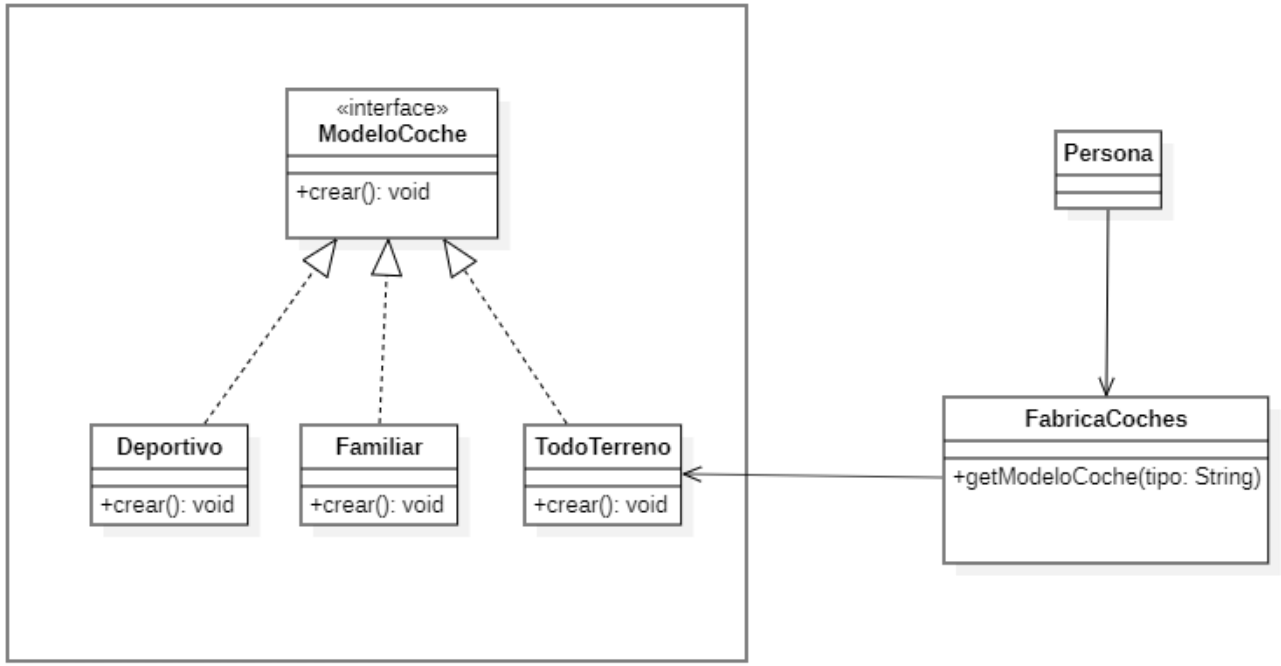
Controlando la creación de Beans

Hemos visto varias maneras de crear beans, pero Spring nos ofrece otras maneras de controlar su creación como:

- Creación desde métodos de fabrica estáticos, en lugar de constructores públicos.
- Controlar las instancias que se necesiten de un bean (scope)
- Inicializar y destruir un bean.

Métodos de Fabrica

La mayoría de las veces creamos crearemos nuestros beans invocando uno de los constructores que tenemos, pero estos beans son nuestros, que pasa si ocupamos API de terceras partes, que ocupan una fábrica estática. Ocuparemos el patrón singletón para poder llamar el atributo **factory-method** de la etiqueta bean.



Ejercicio 8: Métodos de Fabrica

Ejercicio 9: Métodos de Fabrica Dos

Alcance de un Bean (scope)

Todos los beans por defecto son instancias únicas, pero hay ocasiones que se desea una instancia única por cada ves que e pida un bean. Para lograr eso Spring con la etiqueta <bean> tiene un atributo scope, que podemos cambiarlo con los siguientes valores:

- singleton: Una única instancia del bean en con contenedor de Spring (predeterminado).
- prototype: Instancia del bean cualquier número de veces (una por uso).
- request: Limita la definición de un bean a una petición HTTP (ApplicationContext).
- session: Limita la definición de un bean a una sesión HTTP (ApplicationContext).
- application: Limita la definición de un bean en toda la aplicación (ApplicationContext).

Singleton

```
<bean id=“servicio“ class=“dgtic.core.servicio.Servicio ”>
  <property name=“empleado” ref=“empleado”>
</bean>
```

Una solo instancia

```
<bean id="empleado" class="dgtic.core.modelo.Empleado"
                    scope="singleton">
                    <property name="nombre" value="Ramon" />
                    <property name="edad" value="27" />
          </bean>
```

```
<bean id=“servicioDos“ class=“dgtic.core.servicio.Servicio ”>
  <property name=“empleado” ref=“empleado”>
</bean>
```

Prototype

Una cada vez que se necesita se crea una instancia

```
<bean id=“servicio“ class=“dgtic.core.servicio.Servicio ”>
  <property name=“empleado” ref=“empleado”>
</bean>
```

1

```
<bean id="empleado" class="dgtic.core.modelo.Empleado"
                    scope=“prototype">
                    <property name="nombre" value="Ramon" />
                    <property name="edad" value="27" />
          </bean>
```

```
<bean id=“servicioDos“ class=“dgtic.core.servicio.Servicio ”>
  <property name=“empleado” ref=“empleado”>
</bean>
```

2

Ejercicio 10: Alcance de un Bean (scope)

Inicializar y Destruir Beans

Spring nos brinda la posibilidad de inicializar y destruir un bean (hay excepciones), las dos maneras de hacerlo es implementado las interfaces **InitializingBean** y **DisposableBean**.

La segunda manera son con los atributos **init-method** y **destroy-method**. Spring no recomienda DisposableBean dado que no empareja el código de Spring.

Ejercicio 11: Inicializar y destruir un bean

Ciclo de vida de un Bean

Etapa	Descripción
Instanciar	Spring instancia el bean
Informar propiedades	Inyecta las propiedades del bean
Establecer el nombre del bean	Si se implementa BeanNameAware, se pasa el id del bean a setBeanName()
Establecer el nombre de la fábrica	Si implementa BeanFactoryAware para la fabrica del bean a setBeanFactory()
ApplicationContext	Se implementa ApplicationContextAware, se llama el método setApplicationContext()
Postprocesado (antes de inicialización)	Si hay BeanPostProccessor, se llama al método postProcessBeforeInitialization()
Inicializar beans	Se implementa InitializingBean, el método afterPropertiesSet(), o si hay existe un método personalizado se llama.
Postprocesado (despues de inicialización)	Si hay BeanPostProccessor, y existe el método postProcessAfterInitialization()
Bean listo para usarse	Ya lo puede utilizar el cliente, existirá hasta que ya no se ocupe
Destruir Bean	Se implementa DisposableBean, se llama el método destroy, o el método de destrucción mersonalizado