



# DESARROLLO DE SOFTWARE I

**Eric Gustavo Coronel Castillo**

[gcoronelc.blogspot.com](http://gcoronelc.blogspot.com)

[ecoronel@uch.edu.pe](mailto:ecoronel@uch.edu.pe)



**TEMA: Spring Core con XML**



En esta sesión  
comprenderás como  
funciona el patrón Inyección  
de Dependencias y su  
implementación con Spring  
Framework utilizando XML.

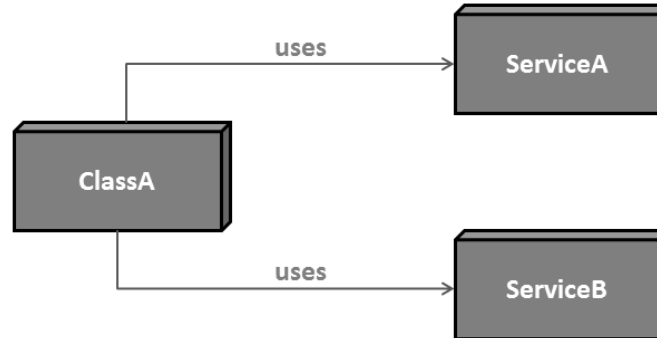




## PROBLEMA

Como todo patrón, comienza planteando un **problema** para el que plantea una solución.

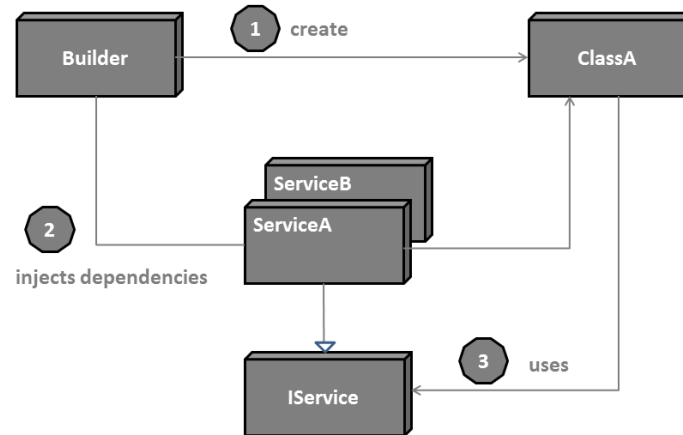
Muchas veces, un componente tiene dependencias de servicios o componentes cuyos tipos concretos son especificados en tiempo de diseño.





## SOLUCIÓN

La **solución** pasa por delegar la función de seleccionar una implementación concreta de las dependencias a un **componente externo**.





## SOLUCIÓN

Existen tres maneras de implementar la inyección de dependencias:

- Inyección basada en constructor
- Inyección basada en métodos setters
- Inyección basada en interfaces



## SOLUCIÓN

### Inyección basada en constructor

Las dependencias se inyectan utilizando un constructor con parámetros del objeto dependiente.

Éste constructor recibe las dependencias como parámetros y las establece en los atributos del objeto.



## SOLUCIÓN

### **Inyección basada en métodos setters**

En este tipo de DI, se utiliza un método setters para inyectar una dependencia en el objeto que la requiere.

Se invoca así al setter de cada dependencia y se le pasa como parámetro una referencia a la misma.





## SOLUCIÓN

### Inyección basada en interfaces

Aquí se utiliza una interfaz común que otras clases implementan para poderles luego inyectar dependencias.

El módulo **Core** es el más importante de Spring. Es el que provee el Contenedor DI. Este contenedor nos permite aplicar el patrón Dependency Injection en nuestras aplicaciones.



## LOS BEANS

```
<beans ... >
    ...
    <bean id="miPrimerBean" class="paquete.Ejemplo"/>
    ...
</beans>
```



## BEAN SCOPES

- **Singleton:** Se crea una sola instancia del bean por DI container.
- **Prototype:** Permite crear cualquier cantidad de instancias del bean.
- **Request:** Se crea automáticamente una instancia del bean por cada request. Podemos modificar el bean y solo será modificado para el request en el que nos encontramos trabajando, los otros request no verán estos cambios. Solo válido utilizando ApplicationContext para aplicaciones web.
- **Session:** Se crea automáticamente una instancia del bean por cada session. Podemos modificar el bean y solo será modificado para la session en la que nos encontramos trabajando, las otras session no verán estos cambios. Solo válido utilizando ApplicationContext para aplicaciones web.
- **Global Session:** Similar al Session, es utilizado por portlets para compartir la session entre los distintos portlets. En caso de que no estemos utilizando portlets este scope degrada al scope session. Solo válido utilizado ApplicationContext

## INSTANCIANDO EL CONTEXTO

```
String contexto = "/contexto/contexto.xml";  
BeanFactory beanFactory;  
beanFactory = new ClassPathXmlApplicationContext(contexto);
```

## INYECCIÓN POR MÉTODOS SETTERS

```
<beans ... >

  <bean id="meInyectan" class="paquete2.UnBean"/>

  <bean id="meInyectanTambien" class="paquete3.OtroBean"/>

  <bean id="beanEjemplo" class="paquete.Ejemplo">
    <property name="unBean"><ref bean="meInyectan"/></property>
    <property name="otroBean" ref="meInyectanTambien"/>
    <property name="valorEntero" value="1"/>
  </bean>

</beans>
```

## INYECCIÓN POR CONSTRUCTOR

```
<beans ... >

  <bean id="unBean" class="paquete2.UnBean"/>

  <bean id="otroBean" class="paquete3.OtroBean"/>

  <bean id="beanEjemplo" class="paquete.Ejemplo" >
    <constructor-arg><ref bean="unBean"/></constructor-arg>
    <constructor-arg ref="otroBean"/>
    <constructor-arg type="int" value="1"/>
  </bean>

</beans>
```