educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 3 - Java Architect

Módulo 21

educación IT
CODE YOUR FUTURE

Spring Modules

Spring Modules

Spring Framework está **organizado en módulos** que se agrupan en:

- Core Container.
- Data Access/Integration.
- Web.
- AOP *(Aspect Oriented Programming)*, Instrumentation.
- Test.

Desde un punto de vista más genérico, Spring Framework puede verse como un soporte que proporciona **tres elementos básicos:**

1. **Servicios empresariales:** podemos hacer de manera sencilla que los objetos sean transaccionales, o que su acceso esté restringido a ciertos roles, o que sea accesible de manera remota y transparente para el desarrollador, o acceder a otros muchos servicios más, sin tener que escribir el código de manera manual.

2. **Estereotipos configurables:** se pueden anotar clases indicando, por ejemplo, que pertenecen a la capa de negocio o de acceso a datos. Se pueden configurar nuestros propios estereotipos. Por ejemplo, podríamos definir un nuevo estereotipo que indicara un objeto de negocio que además sería cacheable automáticamente y con acceso restringido a usuarios con determinado rol.

3. **Inyección de dependencias:** nos permite solucionar de forma sencilla y elegante cómo proporcionar a un objeto cliente acceso a un objeto que da un servicio que este necesita. Por ejemplo, que un objeto de la capa de presentación se pueda comunicar con uno de negocio. En Spring las dependencias se pueden definir con anotaciones o con XML.

Adicionalmente Spring tiene una serie de **proyectos independientes** basados en el *core* del Framework que complementan y amplían las funcionalidades que ofrece:

- Spring Data,
- Spring Integration,
- Spring Batch,
- Spring Security,
- Spring Web Flow,
- Spring Boot, y muchos otros.

DI / IoC

Inyección de Dependencias (DI)

¿Qué es?

Originalmente, la inyección de dependencias se llamaba *inversión de control* (muchos autores intercambian el mismo concepto al hablar de ellas), pero en 2004 Martin Fowler llegó a la conclusión de que, en realidad, no se invertía el control, sino las dependencias.

La DI *(Dependency Injection)* permite a un objeto conocer sus dependencias mediante una interfaz y no por su implementación. De esta forma, la implementación puede variar sin que el objeto dependiente se dé cuenta.

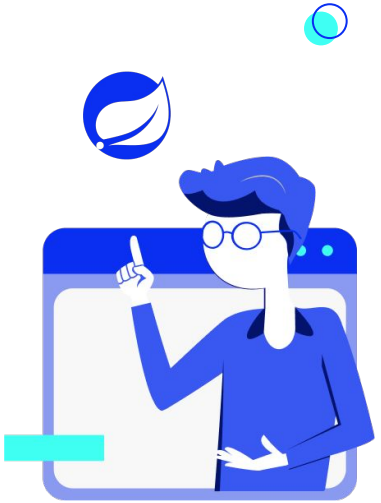
La gran ventaja de la DI es el acoplamiento débil entre objetos.

El objetivo es lograr un bajo acoplamiento entre los objetos en la aplicación. Con este patrón, los objetos no crean o buscan sus dependencias, sino que éstas son suministradas al objeto a través de un tercero (el contenedor).

El contenedor (la entidad que coordina cada objeto en el sistema) es el encargado de realizar este trabajo al momento de instanciar el objeto. Se invierte la responsabilidad en cuanto a la manera en que un objeto obtiene la referencia a otro objeto.

De esta manera, los objetos conocen sus dependencias por su interfaz. Así la **dependencia puede ser intercambiada por distintas implementaciones a través del contenedor.**

En resumen, **se programará orientado a interfaces y se inyectarán las implementaciones a través del contenedor.**

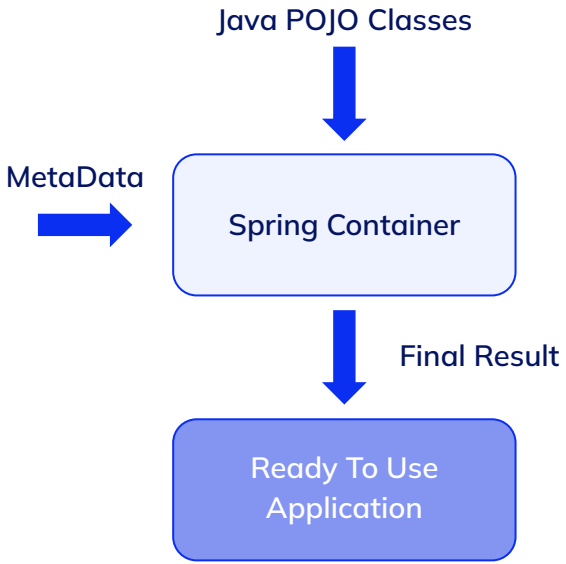


Inversión de Control (IoC)

¿Qué es?

En inglés, *Inversion of Control (IoC)*, **es un estilo de programación** en el que **un framework o librería controla el flujo de un programa.** Esto representa un cambio con respecto a paradigmas tradicionales donde el desarrollador especifica todo el flujo del programa.

Es el principio subyacente a la técnica de *Inyección de Dependencias*, siendo términos frecuentemente confundidos.



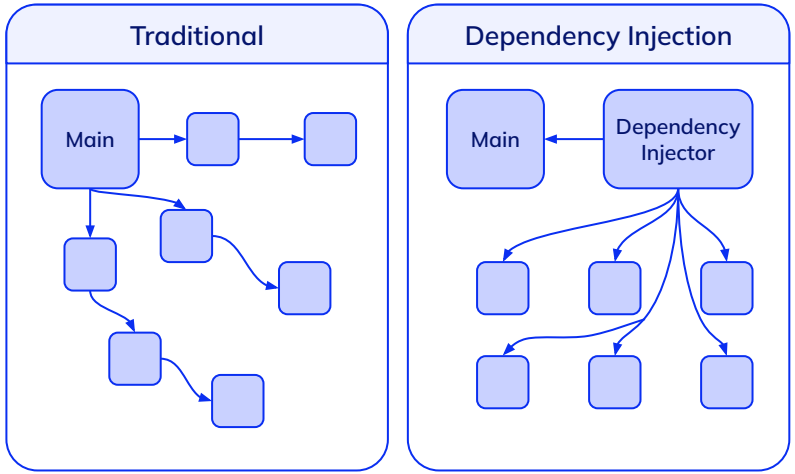
Ventajas de DI/ IoC

1. **Eliminación de instrucciones `new`:** al usar este tipo de patrón, la primera consecuencia observable es que el código permitirá que las instancias inyectadas no se generen desde dentro a través del uso de la palabra "`new`" sino que se harán por reflexión o mediante algún *framework*.
2. **Sustitución de piezas software (Legos):** como se mencionó previamente, una ventaja es la posibilidad de modificar una pieza de *software* por otra sin necesidad de reprogramar las clases que la van a utilizar. Esto puede ser interesante en desarrollos en los que es posible añadir o quitar elementos variables como plugins.

3. **Mejoras en los test:** debido a la no utilización de la instrucción `new`, se tendrá una gran ventaja para el testeo automático de aplicaciones que consiste en hacer *mocking*. Esta técnica consiste en simular piezas de *software* dependientes y reemplazarlas por otras más simples y más rápidas para ejecutar multitud de *tests*.

4. **Depuración de clases ya compiladas:** aunque se tienen grandes ventajas con el uso del *mocking*, muchas veces el problema en los test es proveer lo que no se conoce. Los aracteres ocultos, datos incorrectos, valores extraños e información corrupta pueden hacer que nuestro sistema no funcione bien y no lo hayamos previsto. En ese caso los test automáticos no funcionarán y tendremos que depurar el código para encontrar ese misterioso problema. Esto no es algo propio del patrón de inyección de dependencias, pero sí que el patrón conduce o induce a crear un diseño modular y con componentes independientes y como consecuencia a crear diferentes proyectos para cada módulo.

5. **Configuraciones:** las configuraciones también son tareas a añadir en el caso de utilizar inyección de dependencias. Al hacer la inyección, se traslada la responsabilidad de la instanciación de un código imperativo a un código declarativo. Estas declaraciones pueden ser de muchas formas como xml, anotaciones, archivos de texto, bases de datos, y etc. Dependerá del framework, o de nuestro propio algoritmo de inyección, que se use de una forma u otra.



¡Sigamos trabajando!