Qué es la inyección de dependencias

Aparte de un **patrón de diseño de software**, vamos a explicar qué idea hay detrás de ese nombre. Este patrón, como muchos otros, nos ayuda a separar nuestro código por responsabilidades, siendo que en esta ocasión sólo se dedica a organizar el código que tiene que ver con la creación de los objetos.

Como ya sabemos, uno de los principios básicos de la programación, y de las buenas prácticas, es la **separación del código por responsabilidades.** Pues la inyección de dependencias parte de ahí.

En el código de una aplicación con OOP (Programación Orientada a Objetos) tenemos una posible separación del código en dos partes, una en la que creamos los objetos y otra en la que los usamos. Existen patrones como las factorías que tratan esa parte, pero la inyección de dependencias va un poco más allá. Lo que dice es que **los objetos nunca deben construir aquellos otros objetos que necesitan para funcionar.** Esa parte de creación de los objetos se debe hacer en otro lugar diferente a la inicialización de un objeto.

Por ejemplo, este código no sería el mejor (pseudocódigo que sirve para cualquier lenguaje ya que no importa aquí el lenguaje de programación sino entender el concepto):

```
class programador{
	ordenador
	lenguaje

	constructor(){
		this.ordenador = new Mac()
		this.lenguaje = new ObjectiveC()
	}
}

miguel = new Programador()
```

El problema que nos encontramos es que la clase Programador está **fuertemente acoplada** con la del ordenador Mac o el lenguaje ObjectiveC. Si mañana queremos tener programadores de C que usan Windows, tal como está el código, tendríamos que crear una nueva clase Programador, porque esta no nos valdría. Ahora veamos esta alternativa de código mucho más versátil:

```
class programador{
	ordenador
	lenguaje

	constructor(ordenador, lenguaje){
		this.ordenador = ordenador
		this.lenguaje = lenguaje
	}
}

miguel = new Programador( new Mac(), new ObjectiveC() )
carlos = new Programador( new Windows(), new Java() )
```

Ahora nuestro programador es capaz de adaptarse a cualquier tipo de ordenador y cualquier tipo de lenguaje. De hecho, observarás que hemos podido crear un segundo programador llamado "carlos" que es capaz de programar en Java bajo Windows.

Claro que esto es solo un ejemplo ridículo, pero si has podido apreciar la diferencia, podrás entender el resto que viene detrás del concepto de inyección de dependencias. En realidad, es tan sencillo como apreciar que al constructor de los objetos se les están pasando aquellas dependencias que ellos tienen para poder realizar sus tareas.

El hecho en sí, de enviar por parámetros los objetos que son necesarios para que otro objeto funcione, es la inyección de dependencias.

Contenedor de dependencias

El código que has visto anteriormente es muy sencillo, pero en estructuras más complejas observarás que hay muchos objetos que dependen de otros objetos. Esos otros objetos a su vez dependen de otros, que dependen de otros. Como has observado, dado el patrón de inyección de dependencias, necesito tener listos todos los objetos de los que depende el que voy a construir, porque se los tengo que enviar por parámetro al constructor.

Por ejemplo, el programador depende del ordenador y los lenguajes, pero el ordenador depende del sistema operativo y el teclado y ratón. A su vez el teclado depende de una conexión USB y de un conjunto de teclas, la conexión USB depende de las líneas de comunicación de datos y de la electricidad, la electricidad depende de que hayas pagado tu factura el mes anterior y de que así haya tensión en la red. Así podríamos continuar hasta donde nos lleve la imaginación.

Todo eso nos indica que, para conseguir instanciar un programador necesito haber instanciado antes la red eléctrica y las líneas de comunicación del USB, la conexión USB, cada una de las teclas del teclado, el teclado, el ratón, el sistema operativo, el ordenador, los lenguajes… y cuando tengo todo eso, por fin puedo invocar al constructor de la clase Programador para obtener el objeto que quería.

¿Complicado? No. Pero sí es laborioso. Quizás tengas una docena de líneas de código, o más, para poder hacer lo que tú querías, que era instanciar un programador al que necesitas inyectarle todas las dependencias, conforme nos dicta el patrón.

La solución a esta problemática nos la trae el contenedor de dependencias, también llamado inyector de dependencias, contenedor de servicios, etc.

Básicamente es como una caja a la que le pido construir las cosas. Esa caja sabe qué tiene que hacer para construir cada uno de los objetos de la aplicación. Si queremos un programador, simplemente le pedimos al contenedor de dependencias que nos cree un objeto de esa clase y él se dedica a crearlo. Finalmente, el "dependency container" lo devuelve listo para ser usado.

```
miguel = contenedorDependencias.crear("Programador");
```

El contenedor de dependencias nos permite que la instanciación de un objeto, por muchas dependencias que tenga, vuelva a ser tan simple como una llamada a un método. ¿Dónde está la magia? el contenedor de dependencias simplemente tiene todos los objetos que

puedas necesitar para crear cualquier objeto complejo y si no cuenta en ese instante con las dependencias necesarias, sabe cómo conseguirlas en el acto.

El uso de este contenedor de dependencias ya depende del lenguaje que estés usando y la librería o framework con la que trabajes. Además, habitualmente hay que hacer algunas configuraciones básicas para que funcione. Si te fijas, en la línea anterior estoy diciendo que me cree un programador, pero en algún lugar le tendré que decir cómo puedo obtener ese objeto deseado y si tal como está ese programador será experto en Java, ObjectiveC o PHP.

Nota: Si nuestro código está correctamente escrito es muy probable que las configuraciones sean mínimas, porque podrá decidirlo a través del análisis de los tipos de datos indicados en las cabeceras de los métodos constructores, pero generalmente hay que configurar algunas cosas básicas.