educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 3 - Java Architect

Módulo 29

educación IT
CODE YOUR FUTURE

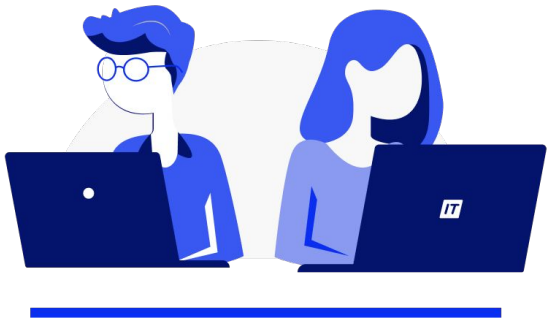
Programación orientada a aspectos con AspectJ

Programación orientada a aspectos

La Programación Orientada a Aspectos - en inglés *Aspect Oriented Programming* -, es un paradigma de programación, relativamente reciente, cuya intención es permitir una adecuada **modularización de las aplicaciones** y posibilitar una mejor **separación de responsabilidades.**

Gracias a la POA se pueden encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas y así eliminar las dependencias entre cada uno de los módulos.

De esta forma, se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables.



Programación orientada a objetos

La Programación Orientada a Objetos (POO) significó un gran avance en la ingeniería del *software*, porque su modelo de objetos es una forma muy eficaz de representar problemas reales. La tarea principal en este paradigma es **descomponer el dominio del problema y encapsular cada concepto en objetos, dotarlos de propiedades y hacerlos interactuar entre ellos.**

Sin embargo, en algunas oportunidades, la complejidad de la solución necesaria para resolver el problema supera las capacidades de la POO.

En estos casos, ni la programación procedural ofrece una solución adecuada. Es aquí donde la **POA** entra en juego, proporcionando una **alternativa para capturar de forma clara las propiedades y comportamientos** que se requieren en la aplicación. Por lo tanto, la POO sigue siendo fundamental en el proceso de desarrollo del *software*, pero en aquellos casos en los que no es suficiente, la POA puede proporcionar una solución satisfactoria.

Separar incumbencias

La Programación Orientada a Aspectos engloba una serie de conceptos y tecnologías que buscan abordar un problema que ha sido reconocido desde hace tiempo en el desarrollo de software: la **separación de incumbencias *(concerns).***

La POA intenta **separar componentes y aspectos unos de otros**, proporcionando mecanismos que hagan posible abstraerlos y componerlos para formar todo el sistema. En definitiva, lo que se persigue es implementar una aplicación de forma eficiente y fácil de entender.

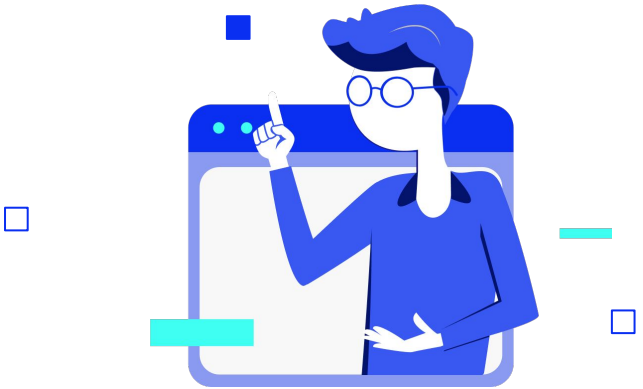
POA es un desarrollo que sigue al paradigma de la orientación a objetos, y como tal, **soporta la descomposición orientada a objetos, además de la procedimental y la descomposición funcional.** Pero, a pesar de esto, POA no se puede considerar como una extensión de la POO, ya que **se puede utilizar con diferentes estilos de programación.**

Principio Separación de Incumbencias/Asuntos

Este principio fue identificado en los años 70 y plantea que un problema dado involucra varias incumbencias que deben ser identificadas y separadas.

Las incumbencias son los diferentes temas o asuntos de los que es necesario ocuparse para resolver el problema. Una de ellas es la función específica que debe realizar una aplicación, pero también surgen otras, como por ejemplo: distribución, persistencia, replicación y sincronización.

Al separar las incumbencias, se disminuye la complejidad a la hora de tratarlas y se puede cumplir con requerimientos relacionados con la calidad como adaptabilidad, mantenibilidad, extensibilidad y reusabilidad.



Aplicación

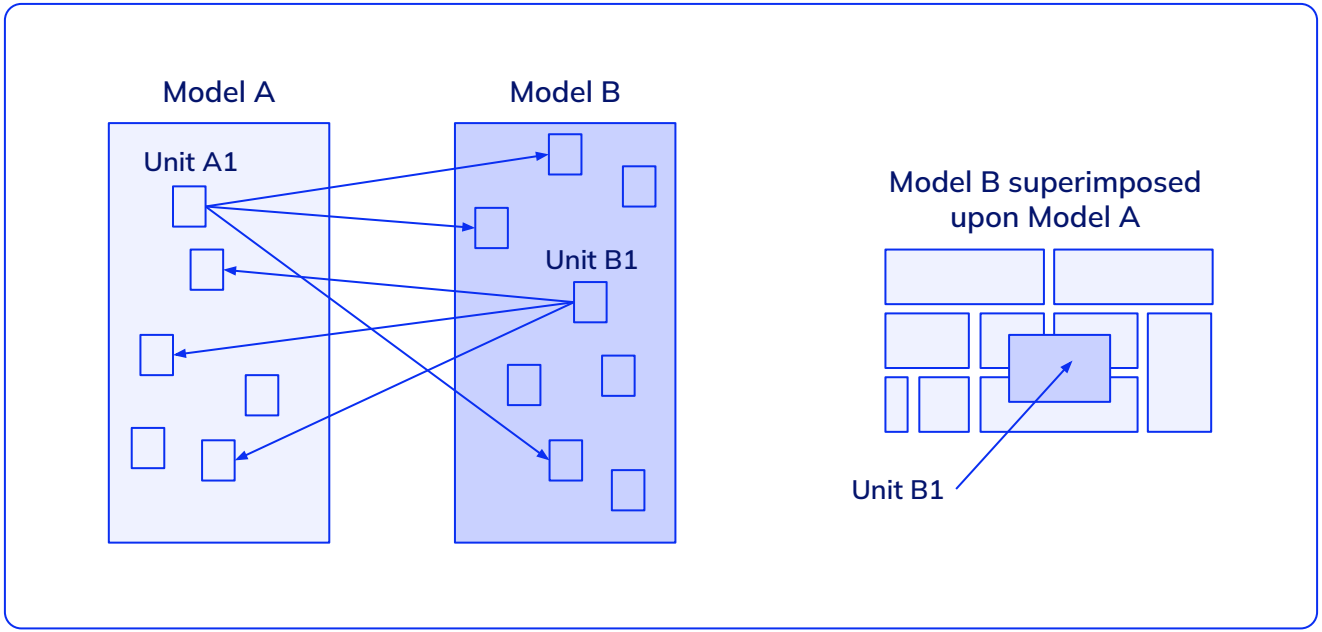
El principio puede aplicarse de distintas maneras. Por ejemplo, separar las fases del proceso de desarrollo puede verse como una separación de actividades de ingeniería en el tiempo y por su objetivo. Definir subsistemas, objetos y componentes son otras formas de poner en práctica el principio de separación de incumbencias. Por eso podemos decir que se trata de un **principio rector omnipresente en el proceso de desarrollo de software.**

Las **técnicas de modelado** que se usan en la etapa de diseño de un sistema se basan en partirlo en varios **subsistemas** que resuelvan parte del problema o correspondan a una parte del dominio sobre el que trata.

La desventaja de estas particiones es que muchas de las incumbencias a tener en cuenta para cumplir con los requerimientos no suelen adaptarse bien a esa descomposición.

El problema aparece cuando una incumbencia afecta a distintas partes del sistema que no aparecen relacionadas en la jerarquía. En ese caso, la única solución suele ser escribir código repetido que resuelva esa incumbencia para cada subsistema.

Ejemplo


Model A
Model B
Unit A1
Unit B1
Model B superimposed upon Model A
Unit B1

Fundamentos de la POA

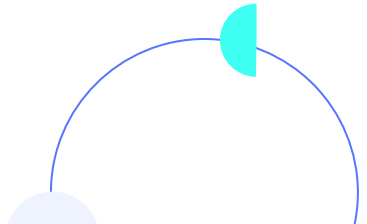
Para hacer un programa orientado a aspectos, es necesario definir los siguientes elementos:

1. Un lenguaje para definir la **funcionalidad básica.** Este lenguaje se conoce como **lenguaje base.** Suele ser un lenguaje de propósito general, como C++ o Java.
2. El **lenguaje de aspectos** define la forma de los aspectos – por ejemplo, los aspectos de AspectJ se programan de forma muy parecida a las clases en Java.
3. El **compilador** se encargará de **combinar los lenguajes**. El proceso de mezcla se puede retrasar para hacerse en tiempo de ejecución, o hacerse en tiempo de compilación.

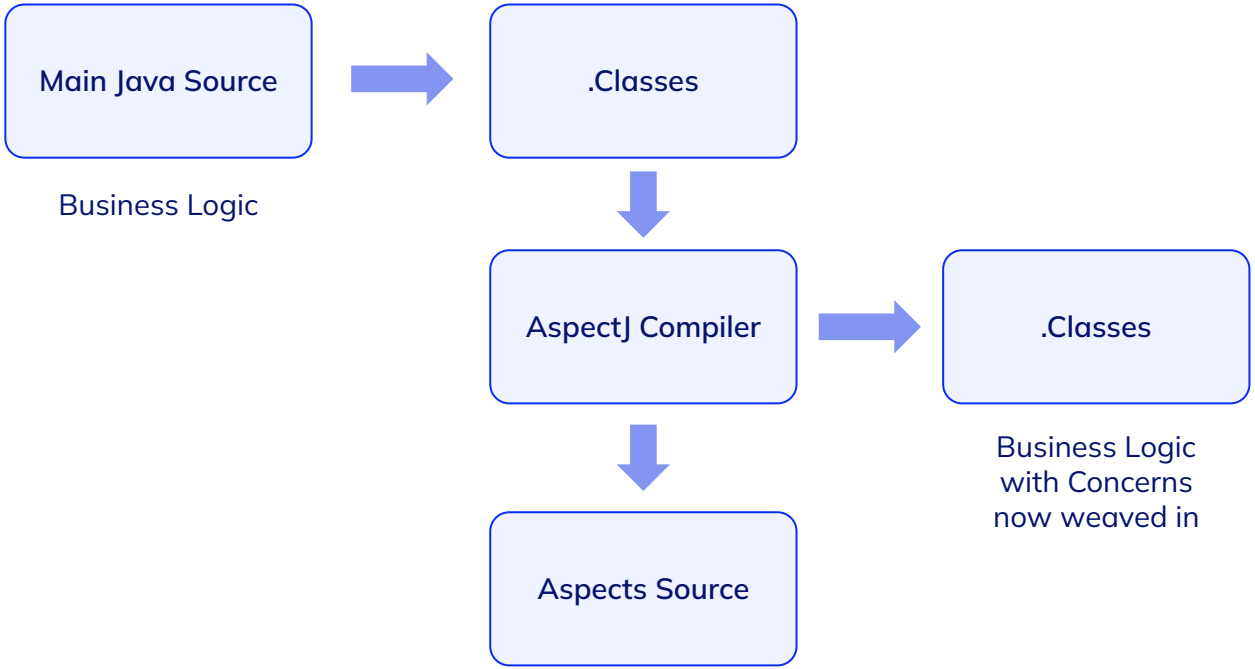
Características de POA

De la consecución de estos objetivos se pueden obtener las siguientes **ventajas:**

1. Un código menos enmarañado, más natural y más reducido.
2. Una mayor facilidad para razonar sobre las materias, ya que están separadas y tienen una dependencia mínima.
3. Más facilidad para depurar y hacer modificaciones en el código.
4. Se consigue que un conjunto grande de modificaciones en la definición de una materia tenga un impacto mínimo en las otras.
5. Se tiene un código más reusable y que se puede acoplar y desacoplar cuando sea necesario.



Proceso de AspectJ



¿Qué puedo hacer con POA?

La programación orientada a aspectos intenta formalizar y representar de forma concisa los **elementos que son transversales a todo el sistema.**

En los lenguajes orientados a objetos, la estructura del sistema se basa en la idea de **clases y jerarquías de clases. La herencia permite modularizar el sistema**, eliminando la necesidad de duplicar código. No obstante, siempre hay aspectos que son transversales a esta estructura: el ejemplo más clásico es el de control de permisos de ejecución de ciertos métodos en una clase, veámoslo a continuación.

Ejemplo

```
public class MiObjetoDeNegocio {
	public void metodoDeNegocio1() throws SinPermisoException {
		chequeaPermisos();
		//resto del código
		…
	}

	public void metodoDeNegocio2() throws SinPermisoException {
		chequeaPermisos();
		//resto del código
		…
	}

	protected void chequeaPermisos() throws SinPermisoException {
		//chequear permisos de ejecución
		…
	}
}
```

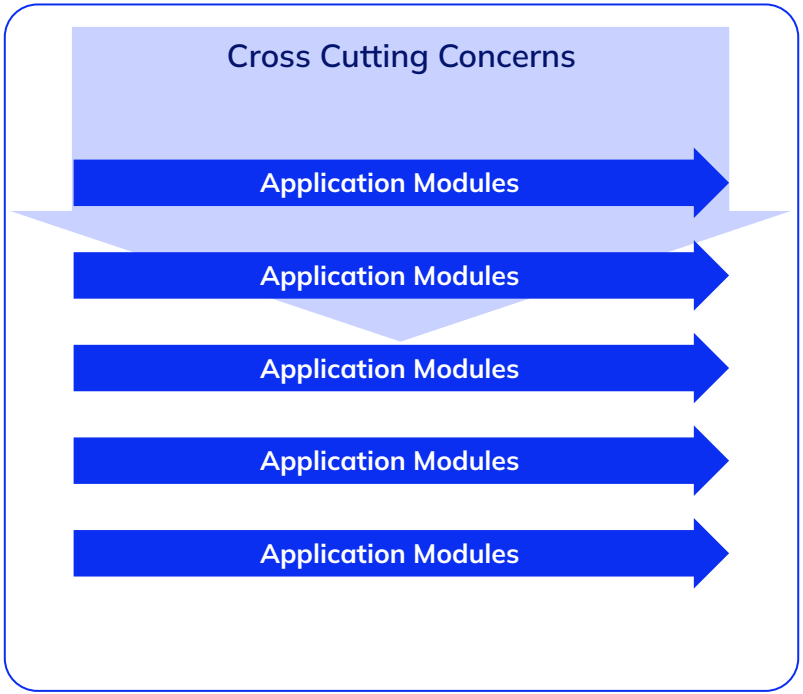
Al estructurar adecuadamente el programa se puede **minimizar la repetición de código**, pero es prácticamente imposible eliminarla.

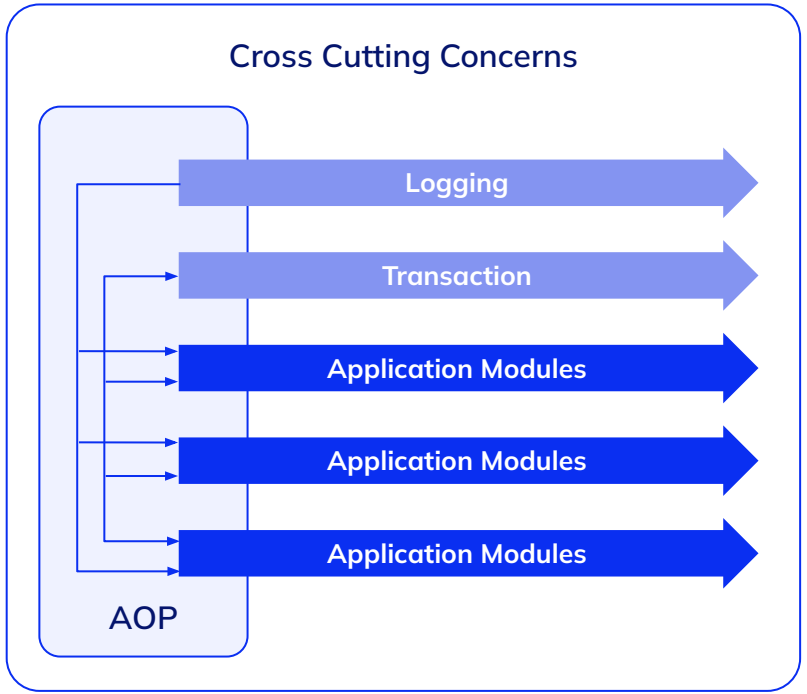
La situación se agravaría si además tuviéramos que controlar permisos en objetos de varias clases. El problema es que **en un lenguaje orientado a objetos los aspectos transversales a la jerarquía de clases no son modularizables** ni se pueden formular de manera concisa con las construcciones del lenguaje.

La programación orientada a aspectos intenta formular conceptos y diseñar construcciones del lenguaje que permitan modelar estos aspectos transversales sin duplicación de código.

En nuestro ejemplo, se necesitaría poder especificar de alguna manera concisa que antes de ejecutar ciertos métodos hay que llamar a cierto código.

Incumbencias transversales



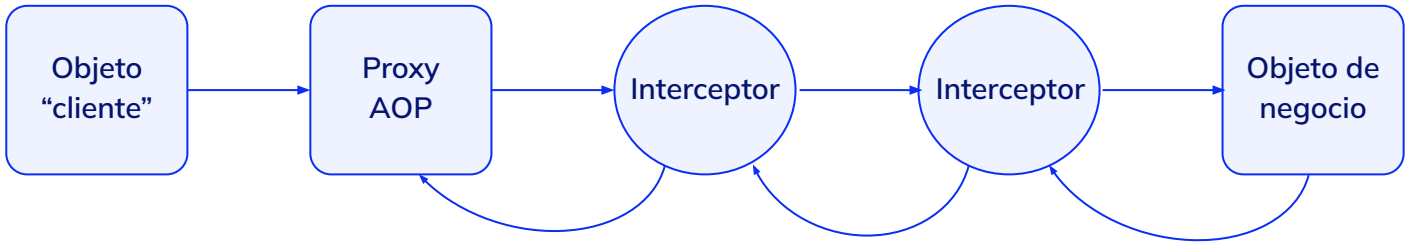


En POA los elementos transversales a la estructura del sistema se pueden modularizar gracias a las construcciones que aporta el paradigma, se les denomina ***aspectos (aspects).*** En el ejemplo anterior el control de permisos de ejecución, modularizado mediante POA, sería un aspecto.

Un consejo *(advice)* es una acción que hay que ejecutar en determinado/s punto/s de un código, para conseguir implementar un aspecto. En el ejemplo, la acción a ejecutar sería la llamada a `chequeaPermisos()`.

El conjunto de puntos del código donde se debe ejecutar un *advice* se conoce como *punto de corte* o *pointcut.* En este caso, serían los métodos `metodoDeNegocio1()` y `metodoDeNegocio2()`.

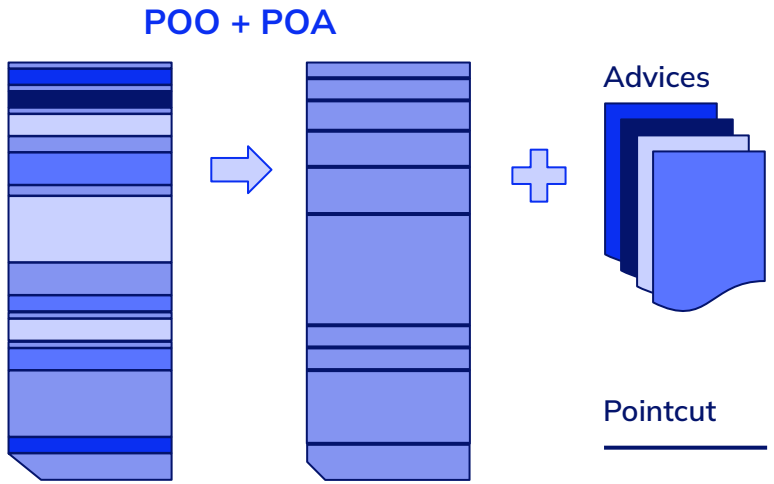
En muchos *frameworks* de POA (Spring incluido), el objeto que debe ejecutar esta acción se modela en la mayoría de casos como **un interceptor: un objeto que recibe una llamada a un método propio antes de que se ejecute ese punto del código.**



Cuando algún objeto llama a un método que forma parte del *pointcut*, el *framework* de POA se las "arregla" para que en realidad se llame a un objeto proxy o intermediario, que tiene un método con el mismo nombre y signatura pero cuya ejecución lo que hace, en realidad, es redirigir la llamada por una cadena de interceptores hasta el método que se quería ejecutar.

En algunas ocasiones nos interesará usar un interceptor para interceptar las llamadas a todos los métodos de una clase. En otras solo nos interesará interceptar algunos métodos.

En Spring, cuando deseamos interceptar las llamadas solo a algunos métodos debemos definir **un *advisor*, que será una combinación de *pointcut* (dónde hay que aplicar POA) más interceptor (qué hay que ejecutar).**



¡Sigamos trabajando!