



# Programación orientada a aspectos

Jesus Felipe Chavarro Muñoz  
Juan Felipe Cárdenas Morales

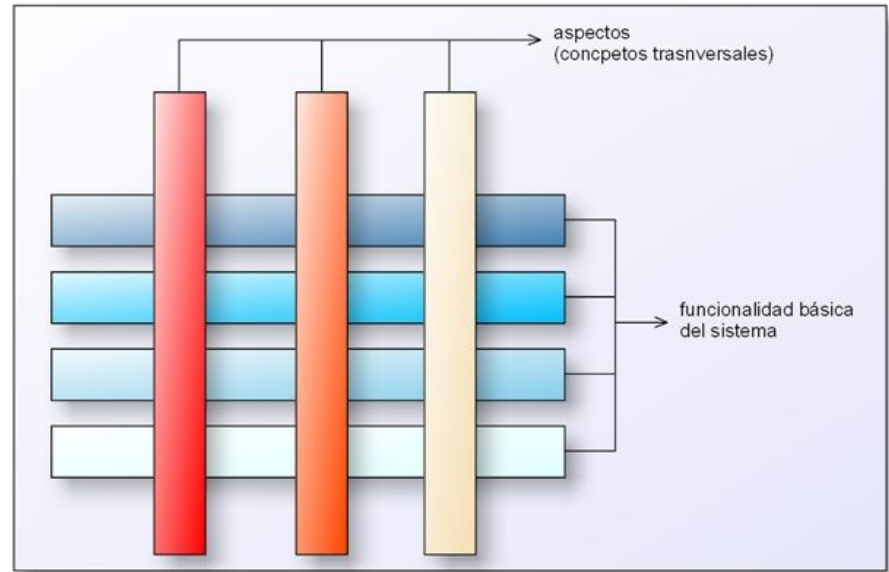


# Índice

1. Introducción
2. Conceptos claves
  - a. cross-cutting
  - b. Aspectos
  - c. Pointcut - Punto de corte
  - d. Advice
3. Implementación
  - a. waver
4. Ventajas
5. Desventajas
6. Criticas
7. Lenguajes de programación
8. Referencia

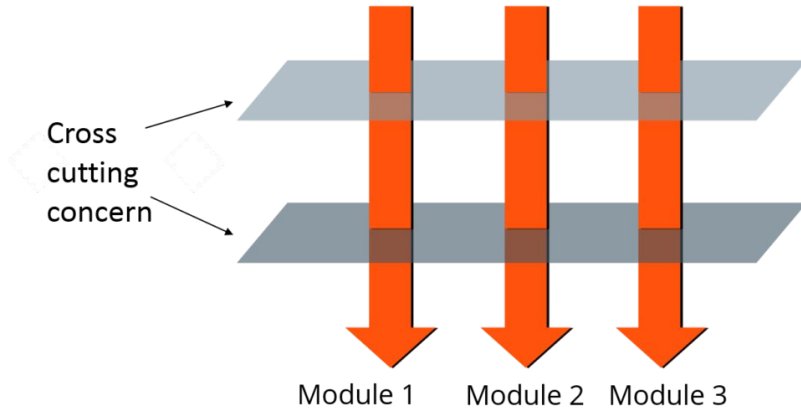
# Introducción

La POA es un paradigma que intenta aumentar el nivel de modularidad a la hora de desarrollar un programa, intentando formalizar y representar cuales son los elementos que están presentes en todo el programa.



# Conceptos claves

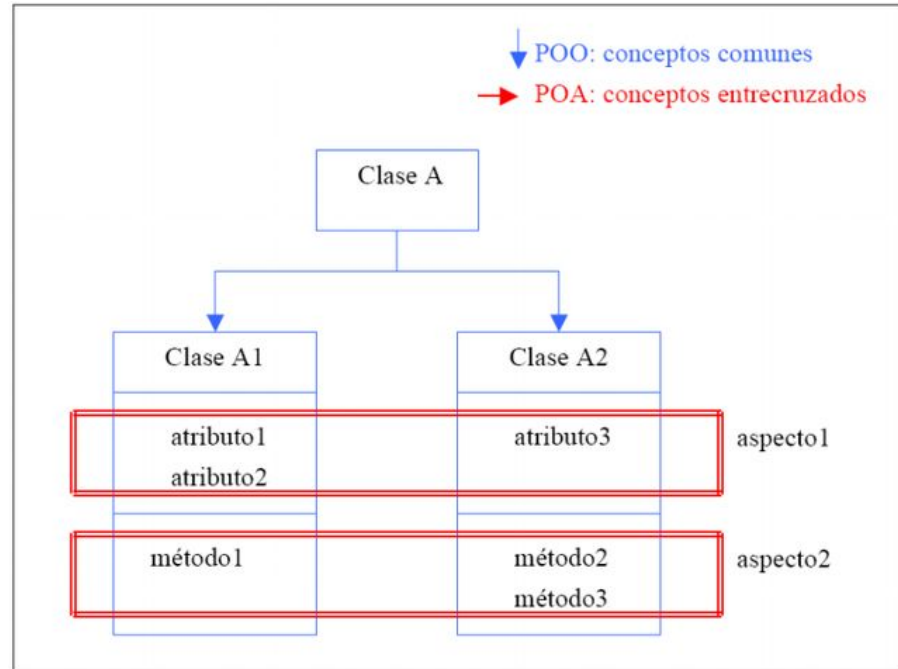
## Cross-Cutting Concern



**Cross-cutting:** Funcionalidades transversales, estas funcionalidades no pueden ser descompuestas facilmente, lleva a codigo duplicado o fuertes dependencias entre sistemas.

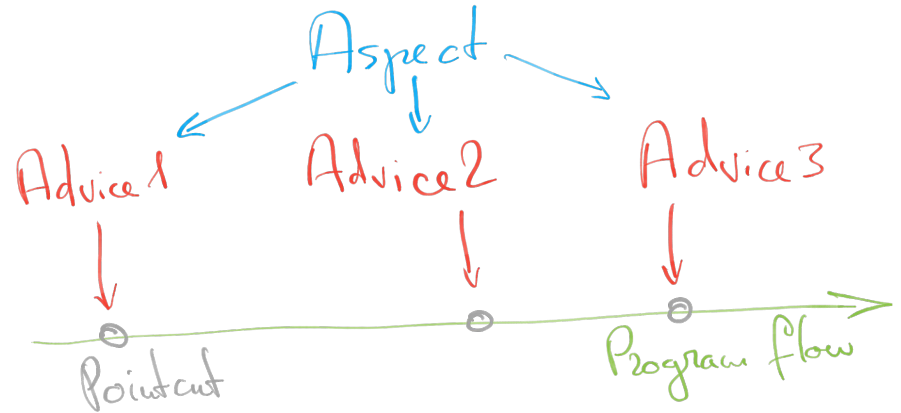
# Aspectos

Es una funcionalidad transversal (cross-cutting) que se va a implementar de forma modular y separada del resto del sistema. Un ejemplo común es el logging o registro de sucesos.



## Pointcut - Punto de corte

Es un punto en la ejecución de un programa en donde un aspecto necesita ser aplicado, este permite saber cuando es aplicado o ejecutado un Advice





# Advice

Un advice(en programación) es una clase o función que modifica a otras funciones o procedimientos.

Un advice (en POA)es la implementación del aspecto, es decir, contiene el código que implementa la nueva funcionalidad.



# Tipos de Advice

Aunque un pointcut define el momento en el que se ejecuta un advice, este se puede ejecutar antes, durante o después del pointcut. Los tipos de advice son:

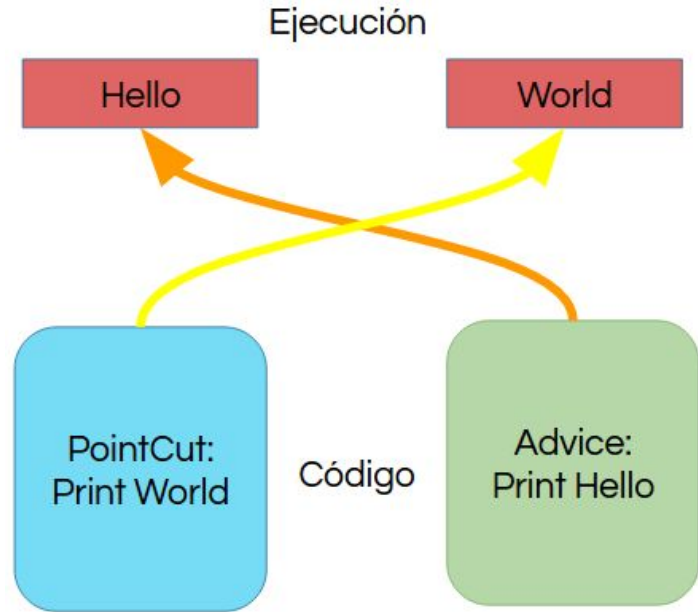
- `before ()`
- `around ()`
- `after ()`



# Advice before

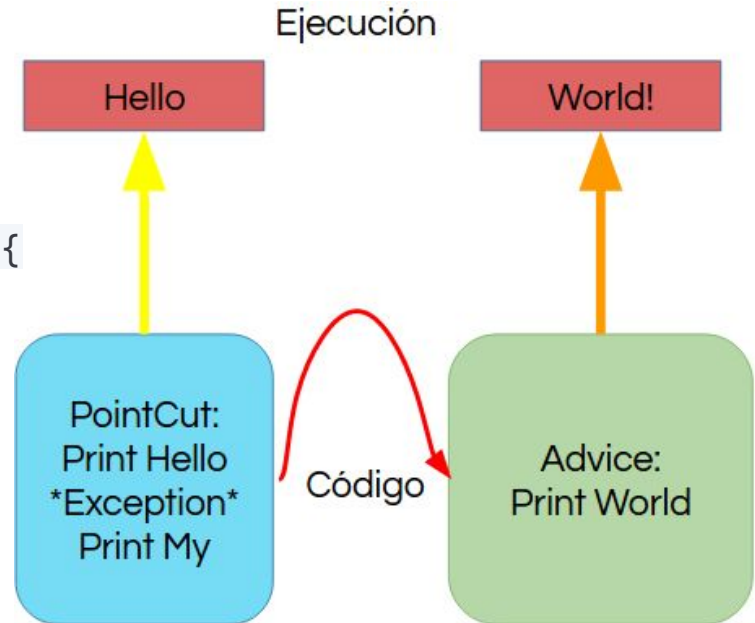
```
pointcut nombreDePunto():  
execution(* NombreClase.metodo(..));
```

```
before() : nombreDePunto(){  
    //CODIGO  
}
```



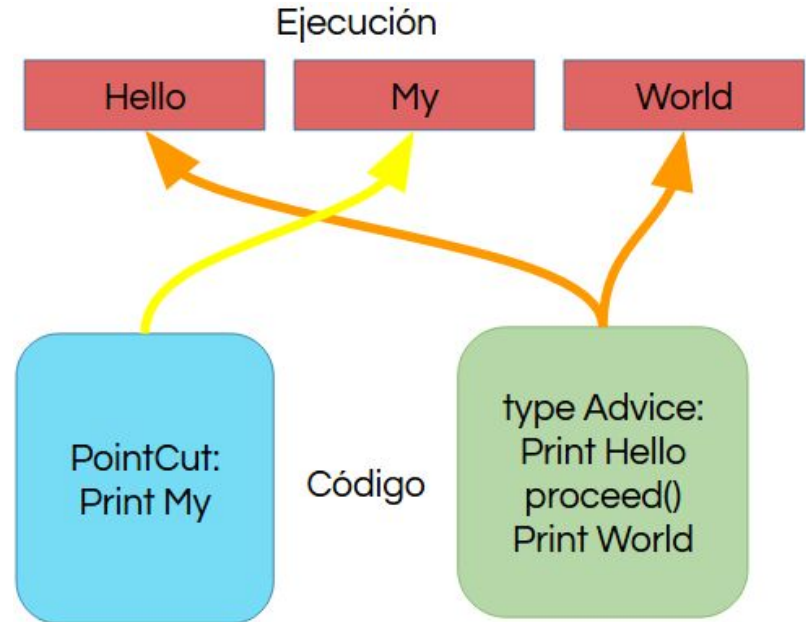
# Advice after

```
after(): nombreDePunto() {  
    //CODIGO  
}  
after() returning(Type X) : nombreDePunto(){  
    //CODIGO  
}  
after() throwing() : nombreDePunto() {  
    //CODIGO  
}
```



# Advice around

```
void around() : nombreDePunto() {  
    //CODIGO  
}  
void around() : nombreDePunto() {  
    //CODIGO  
    proceed();  
    //CODIGO  
}
```



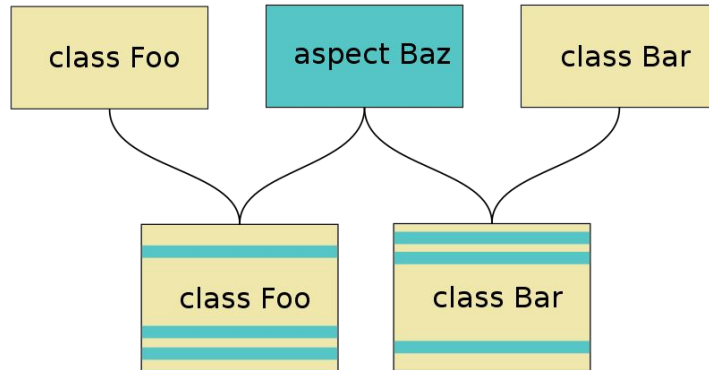


# Implementación

La programación orientada a aspectos es una **técnica transversal** (paradigma) y **no está vinculada a un lenguaje** de programación particular, pero se puede implementar con un lenguaje orientado a objetos como Python y un lenguaje estructurados como C , el único requisito previo es la existencia de un **Weaver(tejedor) de aspectos** para el idioma de destino

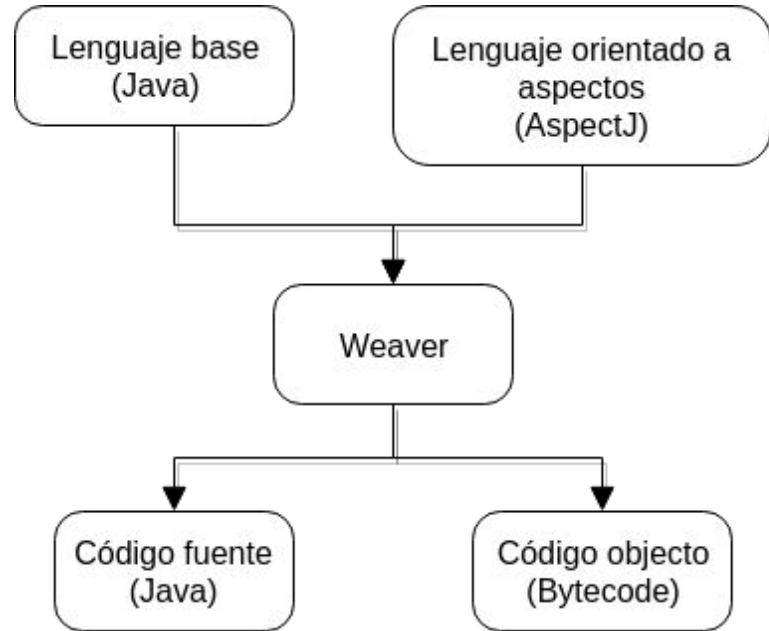
# Weaver

Es una herramienta que procesa un Lenguaje Orientado a Aspectos(LOA) y lo compila a un lenguaje (por lo general orientado a objetos) o lo compila directamente a código fuente



## weaving estático

En tiempo de compilación se añaden los advice en los respectivos pointcuts. Básicamente es un source-to-source compiler de un LOA a un lenguaje base o código objeto.



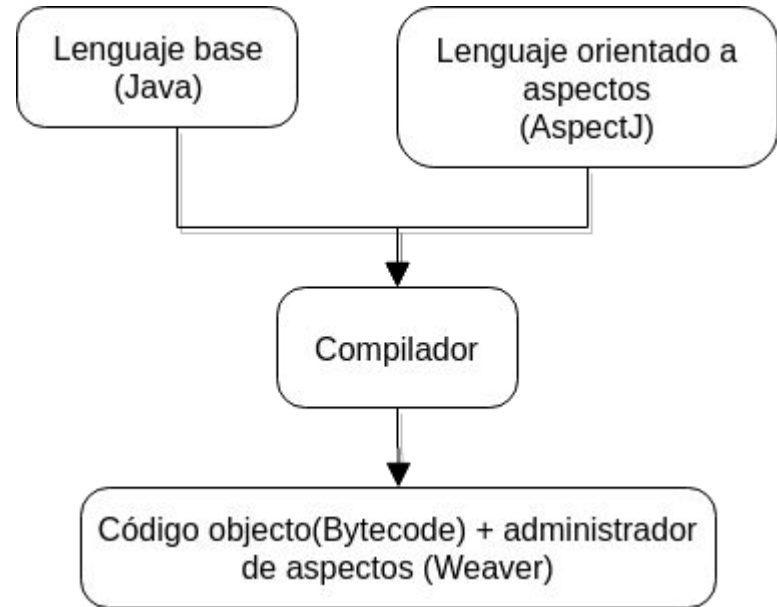
```
aspect Logger {
    pointcut method() : execution(* *(..));
    before() : method() {
        System.out.println("Entering " +
            thisJoinPoint.getSignature().toString());}
    after() : method() {
        System.out.println("Leaving " +
            thisJoinPoint.getSignature().toString());}
}
public class Foo {
    public void bar() {
        System.out.println("Executing Foo.bar()");}
    public void baz() {
        System.out.println("Executing Foo.bar()");}
}
```



```
public class Foo {
    public void bar() {
        System.out.println("Entering Foo.bar()");
        System.out.println("Executing Foo.bar()");
        System.out.println("Leaving Foo.bar()");
    }
    public void baz() {
        System.out.println("Entering Foo.baz()");
        System.out.println("Executing Foo.baz()");
        System.out.println("Leaving Foo.baz()");
    }
}
```

## weaving dinámico

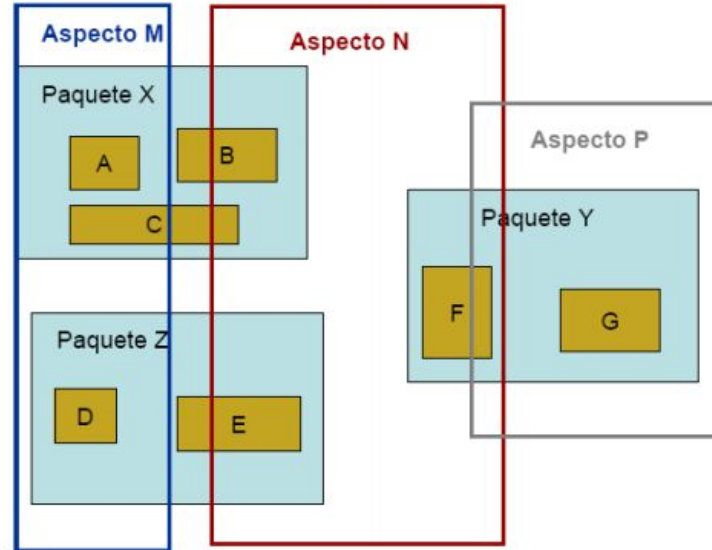
En tiempo de ejecución se identifican los pointcuts y se hace la inclusión y ejecución de los advices cuando sea necesario. Esto se puede hacer mediante reflexión.





# Desarrollo orientado a aspectos - OAD

Diseñar un sistema basado en aspectos requiere entender qué se debe incluir en el lenguaje base, que se debe incluir dentro de los lenguajes de aspectos y qué debe compartirse entre ambos lenguajes.

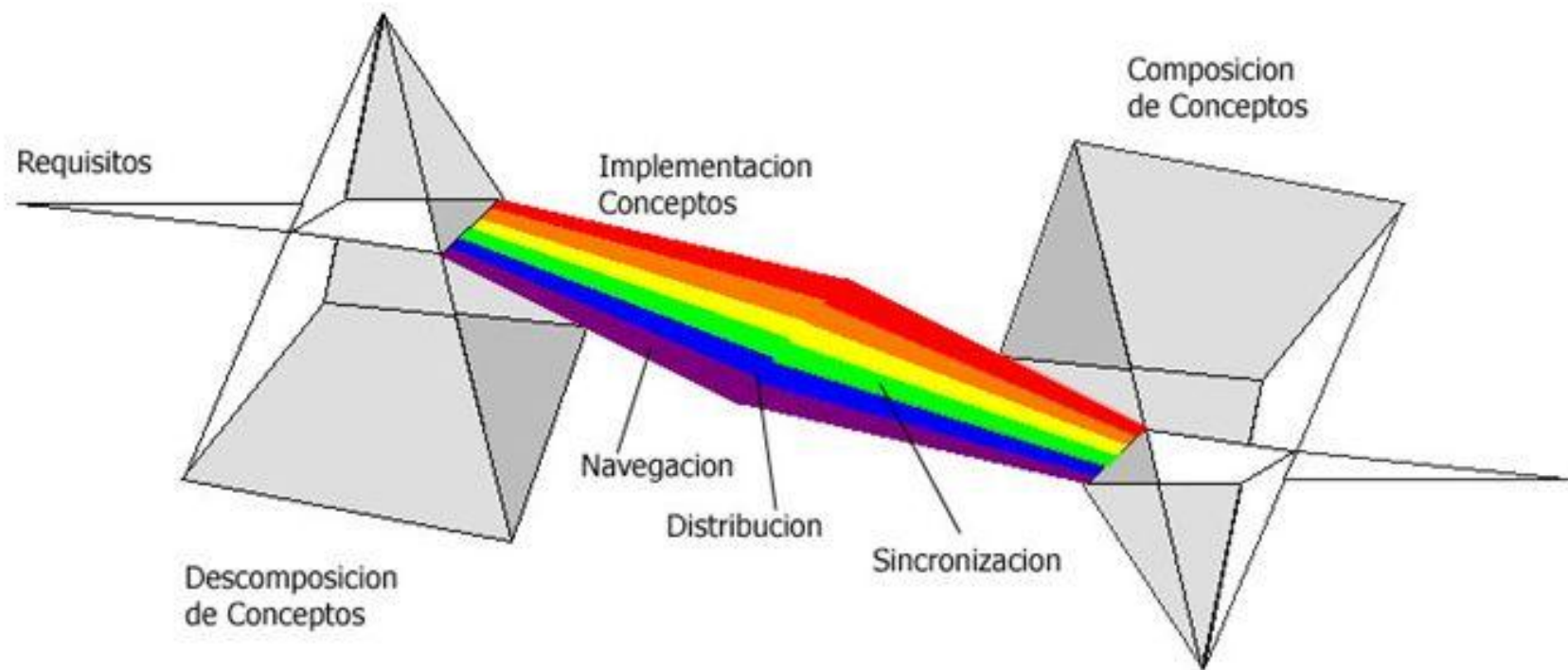




## Desarrollo orientado a aspectos - OASD

En general el desarrollo de una aplicación basada en aspectos consiste de tres pasos:

- Descomposición de aspectos: es descomponer los requerimientos para distinguir aquellos que son aspectos y que no.
- Implementación de requerimientos: implementar cada requerimiento por separado.
- Recomposición: dar las reglas de recomposición que permitan combinar el sistema completo.






# Aplicaciones - Frameworks

**Spring AOP:** El famoso framework para aplicaciones spring en su núcleo tiene un framework para AOP. Este framework es implementado en java puro con las anotaciones de java `@Aspect` o basado en esquema con un XML.

```
@Aspect
public class myAspect {

}
```

```
<aop:config>
    <aop:aspect id="myAspect" ref="aBean">
        ...
    </aop:aspect>
</aop:config>
```



```
// the pointcut expression
@Pointcut("execution(* transfer(..)")
// the pointcut signature
private void businessService() {}

@Before("businessService()")
public void doAccessCheck() {
    // ...
}
```

```
<aop:config>
    <aop:pointcut id="businessService"
expression="execution(* transfer(..)"/>
</aop:config>
```

```
<aop:before
    pointcut-ref="businessService"
    method="doAccessCheck"/>
```



# Aplicaciones - Frameworks

**JBoss AOP:** En el servidor de aplicaciones JBoss AS se incluye en su núcleo el paquete JBoss AOP que por lo general se usa para el desarrollo de servicios de seguridad y manejo de transacciones.

JBoss con anotaciones

```
@Aspect (scope = Scope.PER_VM)
public class FooAspect {
    @Bind (pointcut="execution(* Foo->fooMethod())")
    public Object trace (Invocation invocation) throws
        Throwable {
        try {
            System.out.println("Enter the joinpoint");
            return invocation.invokeNext ();
        } finally {
            System.out.println("Leave the joinpoint");
        }
    }
}
```



## JBoss sin anotaciones

```
public class FooAspect {
    public Object trace (Invocation invocation)
        throws Throwable {
        try {
            System.out.println("Enter the joinpoint");
            return invocation.invokeNext ();
        } finally {
            System.out.println("Leave the joinpoint");
        }
    }
}
```

```
<aop>
  <aspect class="FooAspect"
    scope="PER_VM"/>
  <bind
    pointcut="execution(public
      String Foo->fooMethod(..))">
    <advice name="trace"
      aspect="FooAspect"/>
  </bind>
</aop>
```



## Aplicaciones - Frameworks

**Oracle TopLink:** Es un paquete de mapeo objeto-relacional (ORM) para java. Logra un alto nivel de transparencia y persistencia usando Spring AOP.

.

**Siemens Soarian:** Es un sistema de manejo de información médica. Este usa AspectJ para las funciones transversales de rastreo, auditoría y monitoreo del desempeño

.





## Aplicaciones - Frameworks

**WEAVR:** Es una herramienta creada por motorola que extiende el estándar de UML 2.0 con aspectos.

- .

**Glassbox:** Es un agente de troubleshooting para aplicaciones de java. El inspector supervisa el proceso en la JVM usando AspectJ.

- .



# Escenarios apropiados del paradigma

- Manejo de transacciones
- Sincronización
- Manejo de Memoria
- Control de Acceso o Seguridad
- Logging
- Manejo de Excepciones



# Lenguajes de propósito general

- Aspect C y AspectC++
- AspectJ
- Aspect (PERL)
- AOP.io (PHP)
- AOP con SpringFramework
- Aspyct AOP (Python)

# AspectC++



Es la extensión orientada a aspectos de c y c++. Este funciona compilando fuente a fuente, lo cual traduce el código de aspectc++ en un código compatible con c++.

# AspectJ

Es la extensión orientada a aspectos de java. Este esta disponible en Eclipse Foundation. Es el lenguaje de POA más usado debido a que es particularmente fácil de aprender y utilizar.

# Aspect (Perl)



El Perl Aspect Module intenta seguir de cerca a AspectJ. Pero dada la naturaleza dinámica de Perl, varias de las características de AspectJ son inútiles en Perl.

# Aspyct (Python)

Python no requiere de una extensión de lenguaje para hacer POA. Pero igual si existen alguno proyectos que intentan aportar algunas cosas.



## Lenguajes de dominio específico - DSALs

- COOL(COOrdination Language): trata los aspectos de sincronismo entre hilos concurrentes. (Java)
- RIDL (Remote Interaction and Data transfers aspect Language)
- MALAJ (Multi Aspect LAnguage for Java)
- KALA: Modelos transaccionales avanzados
- DIE: Un lenguaje de aspectos de dominio específico para eventos de IDE
- HYPERJ
- AspectG(ANTLR)
- AspectMatlab



# Ventajas

- Tiene un código más entendible y corto.
- Es más fácil asimilar los conceptos, pues están separados y dependen muy poco entre ellos.
- Facilidad en arreglar y modificar el código.
- Los grandes cambios en la definición de una sección tiene un impacto mínimo en las demás.
- Es más fácil reutilizar código.
- Puede mezclarse con cualquier otro paradigma de programación.



## Desventajas

- Diseñar puntos de enlace entre aspectos es más complicado.
- Pueden surgir problemas entre el lenguaje base y el funcional.
- Es más probable que existan problemas de herencia entre aspectos.
- Surgen nuevos errores propios del paradigma.
- Como es una tecnología nueva, se está cambiando y actualizando periódicamente.
- El control de flujo se difumina un poco haciéndolo igual o peor que la sentencia GOTO.





# Referencias

- [https://en.wikipedia.org/wiki/Cross-cutting\\_concern](https://en.wikipedia.org/wiki/Cross-cutting_concern)
- [https://en.wikipedia.org/wiki/Aspect-oriented\\_programming](https://en.wikipedia.org/wiki/Aspect-oriented_programming)
- [https://es.wikipedia.org/wiki/Programaci%C3%B3n\\_orientada\\_a\\_aspectos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_aspectos)
- [https://fr.wikipedia.org/wiki/Programmation\\_orient%C3%A9e\\_aspect](https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect)
- [http://www.epidataconsulting.com/tikiwiki/tiki-read\\_article.php?articleId=65](http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=65)
- <http://iie.fing.edu.uy/~josej/docs/Programacion%20Orientada%20Aspectos%20-%20Jose%20Joskowicz.pdf>
- <https://docs.spring.io/spring/docs/2.0.x/reference/aop.html>



# Referencias

- <http://www.aspectc.org/doc/ac-languageref.pdf>
- <http://ferestrepoca.github.io/paradigmas-de-programacion/poa/tutoriales/aspectJ/index.html>
- <https://pleiad.cl/papers/2014/fabryAI-csur2014.pdf>
- [http://www.jucs.org/jucs\\_20\\_2/die\\_a\\_domain\\_specific/jucs\\_20\\_02\\_0135\\_0168\\_fabry.pdf](http://www.jucs.org/jucs_20_2/die_a_domain_specific/jucs_20_02_0135_0168_fabry.pdf)
- <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.653.3921&rep=rep1&type=pdf>
- <https://revistas.unal.edu.co/index.php/avances/article/download/10024/10552>
- <http://slideplayer.es/slide/10344266/>
- <http://eil.utoronto.ca/wp-content/static/profiles/rune/node7.html>