

Programación orientada a aspectos

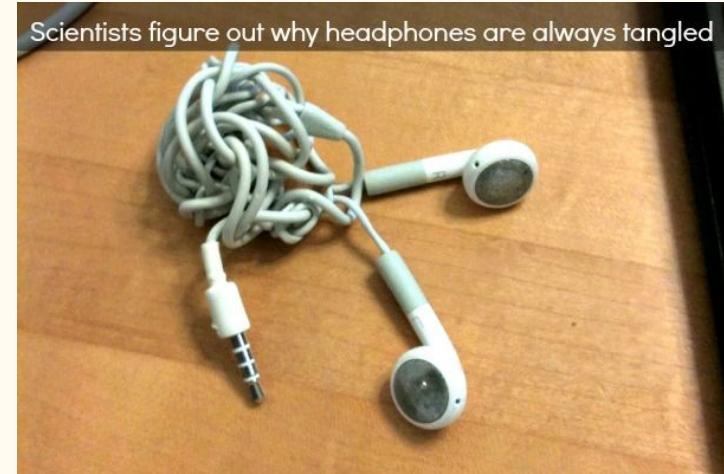
Juan Sebastián Ensuncho



Filosofía del paradigma

¿Qué utilidad tiene respecto a otros paradigmas?

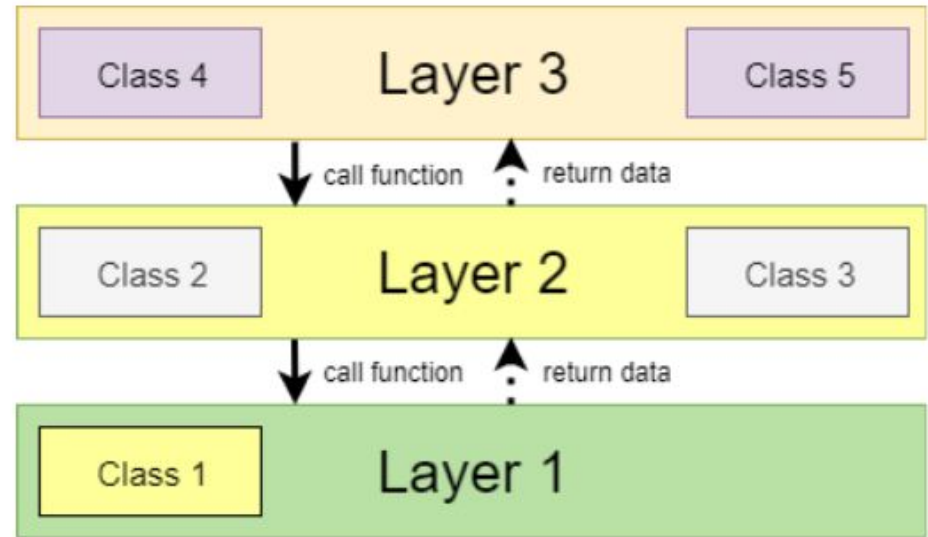
- Permiten abordar problemas que otros paradigmas no logran solucionar de forma adecuada.
- Evita que los sistemas tengan porciones de código “enmarañado”.



Filosofía del paradigma

Por ejemplo:

Imagine un sistema con una arquitectura por capas.



Filosofía del paradigma

```
if (status == 1) {
  //let vehicle_status, vehicle_data = await (await Driver
  let vehicle = await getHandler("Driver_Vehicle").getVehi
  if (vehicle.status != 1) {
    logger.error("api.js: " + vehicle.error)
    return res.status(500).json({ status: -1, error: "Er
  }
  logger.info("api.js: returned Driver id succesfully")
  return res.status(200).json({ status: 1, db_driver_id: d
} else if (status == 0) {
  logger.info("api.js: could not find Driver")
  return res.status(400).json({ status: 0, error: "Correo
}
else if (status == -1 || status == -2) {
  logger.error("api.js: " + data)
  return res.status(500).json({ status: -1, error: "Error
}
```

```
if(bcrypt.compareSync(Driver_password, driver.data.Driver_password)
return {status: 0, data: 'Las contraseñas no coinciden'}};
}
logger.info("DriverHandler: succesfull call to getDriverByEmail")
return {status: 1, data: driver.data};
} else if(status===-1)
{
  logger.error("DriverHandler: error from getDriverBeEmail"+ error)
  return {status: -2, data: driver.error};
}
logger.info("DriverHandler: Driver is not valid");
return {status: 0, data: false}; ;
```

```
if(cargo_info.length==0)
{
  logger.info("CargoController: No cargo found with that id")
  return {status:0, data:" No cargo information found with that id"}
}
else{
  logger.info("CargoController: Cargo info found")
  return {status: 1, data: cargo_info[0].dataValues}
}
```

Filosofía del paradigma

Estos elementos que quedan esparcidos son conocidos como **aspectos**.

Los aspectos son difíciles de encapsular porque son transversales a toda la funcionalidad del sistema

Conceptos claves

Concern: Requerimiento específico del sistema.

Cada concern o “preocupación” del sistema puede ser:

- **Componente o concern central:**
 - Fácilmente encapsulado
 - Asociado a funcionalidades del sistema

Conceptos claves

Concern: Requerimiento específico del sistema.

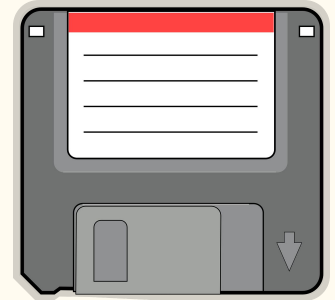
Cada concern o “preocupación” del sistema puede ser:

- **Aspecto o cross cutting concern:**
 - No se puede encapsular de forma limpia.
 - Asociado al desempeño del sistema.

Conceptos claves

Ejemplos de aspecto:

- Sistema de registros (*logger*).
- Uso de memoria.
- Persistencia de datos.
- Seguridad.
- Manejo de errores.



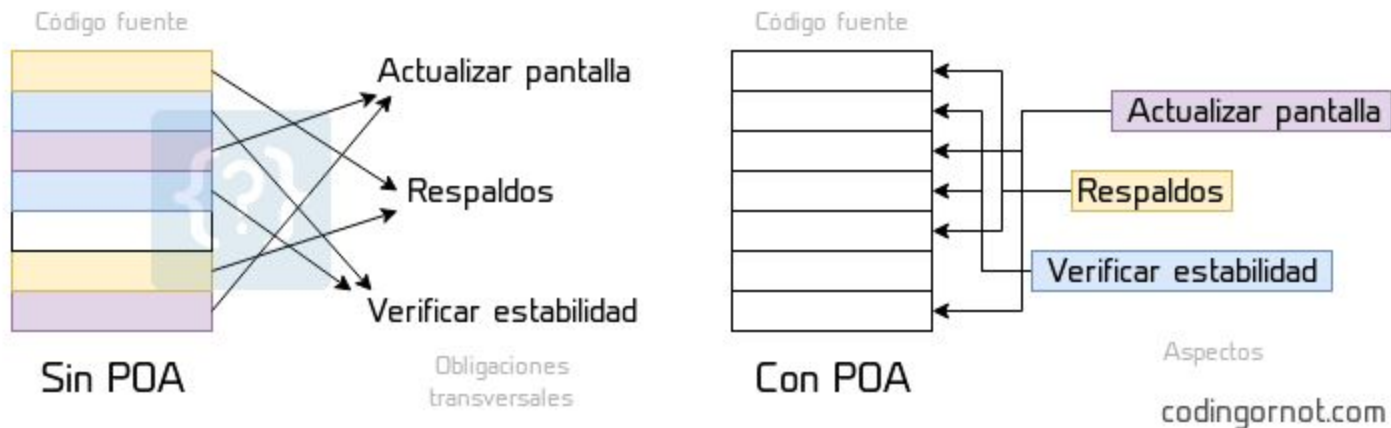
Conceptos clave

Separation of *concerns*: Se deben separar los aspectos de los componentes.



Conceptos clave

Ejemplo de funcionamiento de la programación orientada a aspectos (POA)



Conceptos clave

Hay una serie de pasos que involucran usar este paradigma

1. **Descomposición en aspectos:** Identificar componentes y aspectos.
2. **Implementación de *concerns*:**
 - a. Implementación de componentes.
 - b. Implementación de aspectos.

Conceptos clave

Hay una serie de pasos que involucran usar este paradigma

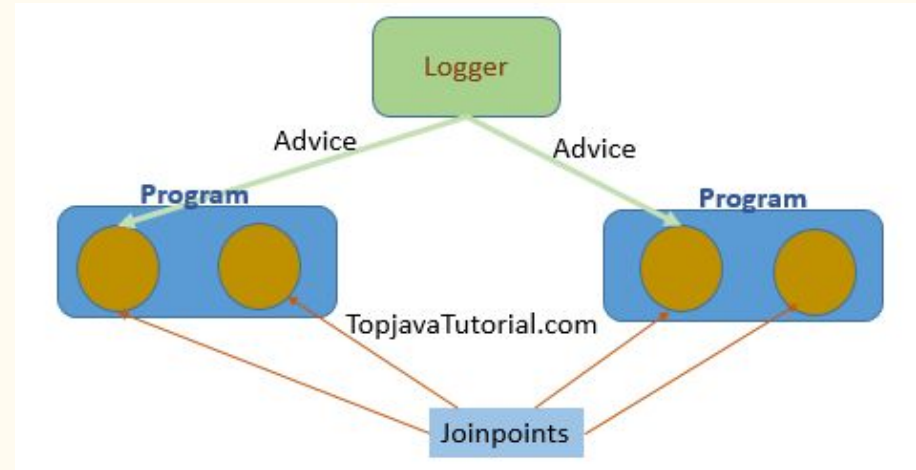
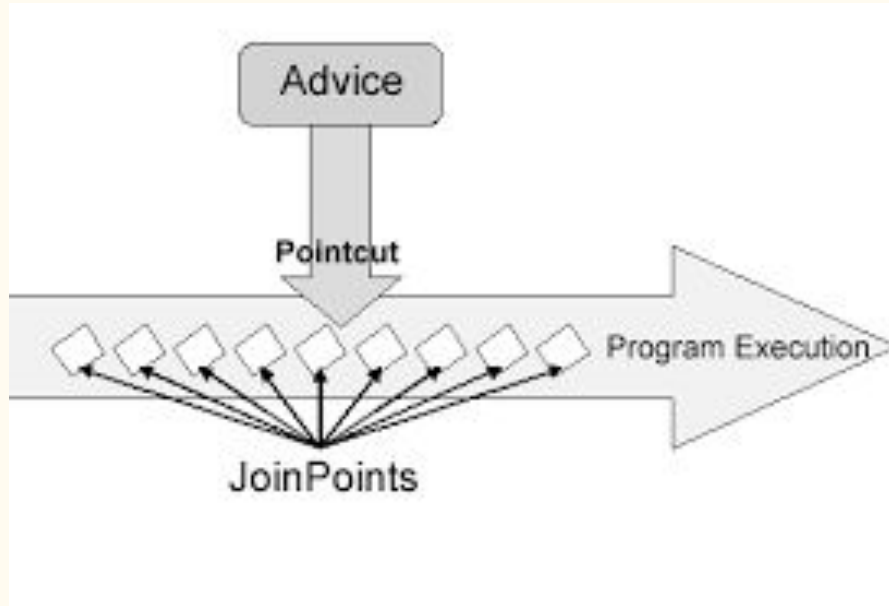
1. **Descomposición en aspectos:** Identificar componentes y aspectos.
2. **Implementación de *concerns*:** Conceptos a tener en cuenta:

Advice: Código que involucra los aspectos.

Join point: Lugar en el programa donde se aplica el aspecto.

Pointcut: Especifican las reglas de recomposición de los aspectos.

Conceptos clave



Conceptos clave

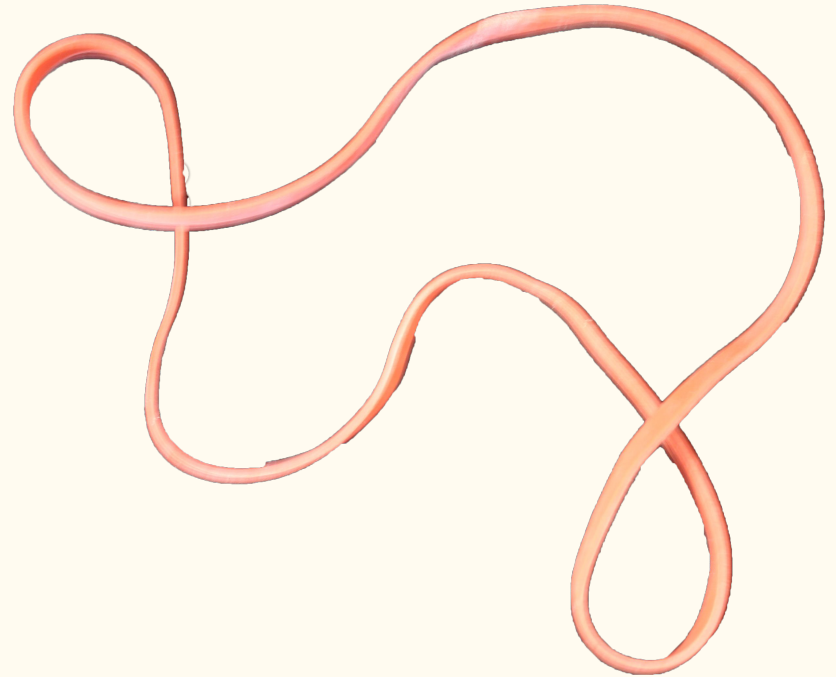
Hay una serie de pasos que involucran usar este paradigma

1. **Descomposición en aspectos:** Identificar componentes y aspectos.
2. **Implementación de *concerns*:**
3. **Recomposición de aspectos:**
 - a. ***Weaving*:** Proceso de recomposición.
 - b. ***Weaver*:** Herramienta que combina componentes y aspectos y produce un código ejecutable.

Ventajas

Flexibilidad y reusabilidad

Los programadores pueden usar módulos de aspecto donde sea necesario en una aplicación sin necesidad de reescribir el código



Ventajas

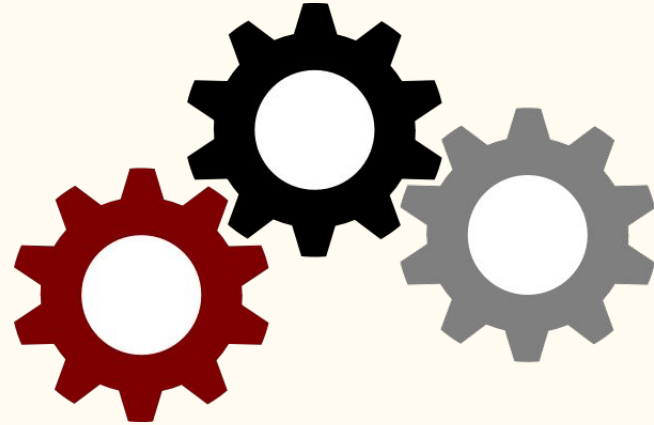
Modularidad

Los programadores pueden usar módulos de aspecto donde sea necesario en una aplicación sin necesidad de reescribir el código.

Ventajas

Mantenibilidad

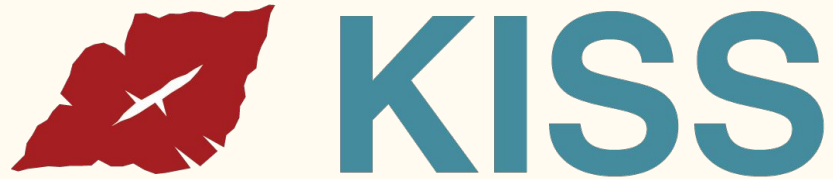
Modulariza las “preocupaciones transversales” mejorando la mantenibilidad y la comprensibilidad del código.



Ventajas

Simplicidad

Los aspectos eliminan muchas líneas de código disperso que de otra manera los programadores tendrían que dedicar un tiempo considerable a escribir, rastrear, mantener y cambiar.

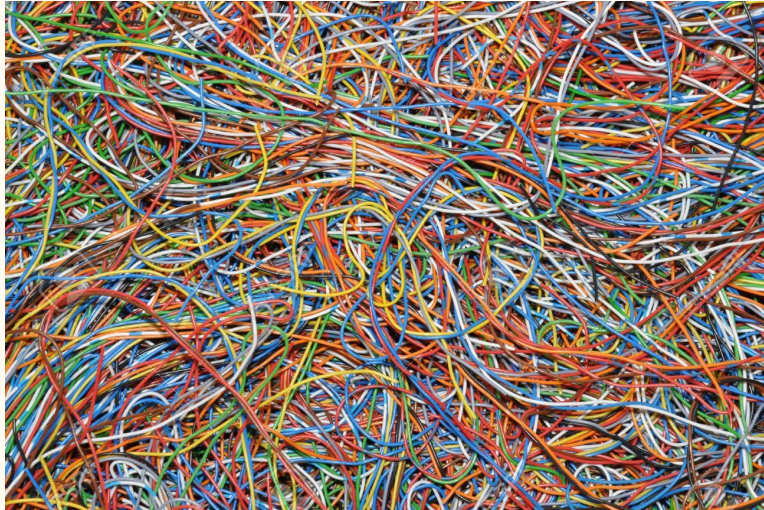


Keep. It. Simple. Stupid.

Ventajas

Scattered and tangled code

Ayuda a corregir problemas de “código disperso” y “código enmarañado” .



Desventajas

Disciplina de código

Requiere mucha disciplina de codificación, especialmente en términos de nombres.



Desventajas

Debugging

Una pequeña dificultad es la depuración del código de una aplicación basada en AOP.



Desventajas

Es muy joven

Debido a que es un paradigma relativamente nuevo, aplicaciones desarrolladas mediante este paradigma necesitan actualizarse constantemente, lo que puede generar problemas de compatibilidad entre versiones de la aplicación.

Lenguajes de Programación

AspectJ

Extensión AOP para el lenguaje Java mediante el IDE Eclipse.

La principal herramienta de soporte de AspectJ es un plug-in de Eclipse, AspectJ Development Tools (AJDT).

Una de las características más importantes de la AJDT es una herramienta para la visualización del **crosscutting**, que es útil para depurar una especificación de un **pointcut**.

Lenguajes de Programación

AspectJ

Soportar el manejo de aspectos agregando a la semántica de Java las siguientes cuatro entidades:

Puntos de enlace.

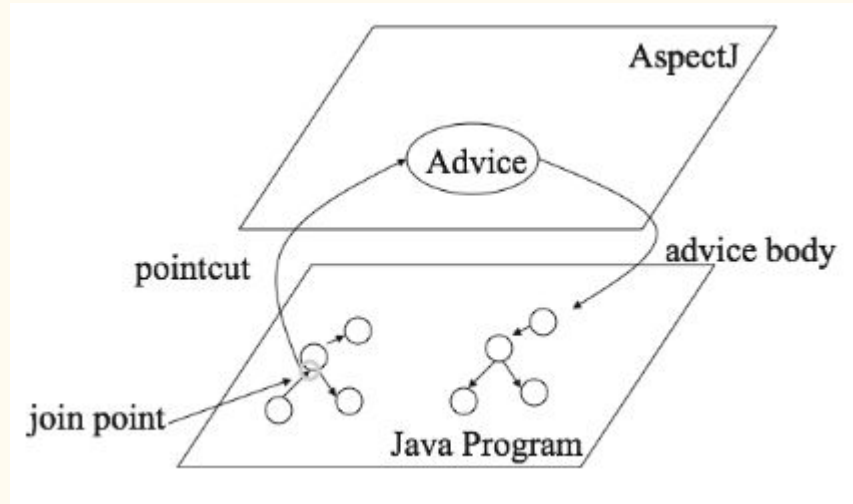
Puntos de corte.

Advices(Consejos).

Instrucciones y declaraciones

Lenguajes de Programación

AspectJ



Lenguajes de Programación

AspectC++

Basándose en la implementación de AspectJ, se desarrolla AspectC y AspectC++ para los lenguajes C y C++.

Tiene un compilador fuente-a-fuente, que traduce el código fuente de AspectC++ a código C++ compilable.

Lenguajes de Programación

AspectR

Lanzada en 2001, AspectR es una gema AOP para el lenguaje Ruby, creada por Avi Bryant y Robert Feldt.

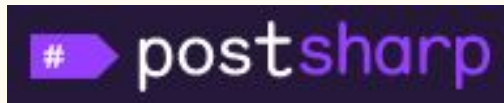
Aquarium

Es un framework que implementa AOP para Ruby. Aquarium proporciona un Lenguaje de Dominio Específico (DSL) con el que se puede expresar el comportamiento "aspectual" del sistema de forma modular

Lenguajes de Programación

PostSharp

Es una aplicación comercial de AOP para .NET con una edición gratuita pero limitada.



AspectJ: Pointcut



```
1 //Cualquier llamado al método setX de los objetos de la clase Point
2 pointcut point_a(): call(void Point.setX(int))
3
4 //Cualquier llamado al método setX OR setY de los objetos de la clase Point
5 pointcut point_b(): call(void Point.setX(int)) || call(void Point.setY(int))
6
7 //Ejecución de cualquier getter (método público //cuyo nombre comience por "get" y que no tenga
  parámetros).
8 pointcut point_c(): call(public * get*())
9
10 pointcut move():
11 call(void FigureElement.setXY(int,int)) ||
12 call(void Point.setX(int)) ||
13 call(void Point.setY(int)) ||
14 call(void Line.setP1(Point)) ||
15 call(void Line.setP2(Point));
```

AspectJ - Aspect



```
1 import org.aspectj.lang.annotation.Aspect;
2 import org.springframework.stereotype.Component;
3
4 @Component
5 @Aspect
6 public class EjemploDeAspecto {
7
8     //aquí vendrían los advices ...
9 }
```

AspectJ - @Before



```
1 import org.aspectj.lang.annotation.Aspect;
2 import org.aspectj.lang.annotation.Before;
3
4 @Aspect
5 public class EjemploBefore {
6
7     @Before("execution(public * get*())")
8     public void controlaPermisos() {
9         // ...
10    }
11
12 }
```

AspectJ - @Around



```
1 import org.aspectj.lang.annotation.Aspect;
2 import org.aspectj.lang.annotation.Around;
3 import org.aspectj.lang.ProceedingJoinPoint;
4
5 @Aspect
6 public class EjemploAround {
7
8     @Around("execution(public * get*())")
9     public Object ejemploAround(ProceedingJoinPoint pjp) throws Throwable {
10         System.out.println("ANTES");
11         Object valorRetorno = pjp.proceed();
12         System.out.println("DESPUES");
13         return valorRetorno;
14     }
15 }
```


AspectJ - @AfterReturning




```
1 import org.aspectj.lang.annotation.Aspect;
2 import org.aspectj.lang.annotation.AfterReturning;
3
4 @Aspect
5 public class EjemploAfterReturning {
6
7     @AfterReturning(
8         pointcut="execution(public * get*())",
9         returning="valor")
10    public void log(Object valor) {
11        // ...
12    }
13 }
```

AspectJ - @AfterThrowing



```
1 @Aspect
2 public class EjemploAfterThrowing {
3
4     @AfterThrowing(
5         pointcut="execution(public * get*())",
6         throwing="daoe")
7     public void logException(DAOException daoe) {
8         // ...
9     }
10 }
```

Ruby - Aquarium



```
1 #Ejecutar advice antes de ejecutar un método (Sin importar si hay errores)
2 before :types ⇒ ...
3 #Ejecutar advice después de ejecutar un método (Sin importar si hay errores)
4 after :types ⇒ ...
5
6 #Ejecutar advices tras ejecutarse de manera correcta (Sin errores)
7 after_returning :types ⇒ ...
8
9 #Ejecutar advices tras producirse un error
10 after_raising :types ⇒ ... # Tras producirse cualquier Exception
11 after_raising ⇒ MyError, :types ⇒ ... # Solo tras producirse "MyError"
12
13 #Ejecutar advice antes y después de ejecutar un método
14 before_and_after :types ⇒, ...
15 before_and_after_returning :types ⇒, ...
```

Ruby - aquarium



```
1 class Foo
2   include Aquarium::DSL
3   ...
4   def critical_operation *args
5     ...
6   end
7 end
8
9 class Foo
10   around :critical_operation do |join_point, object, *args|
11     p "Entering: critical_operation"
12     result = join_point.proceed
13     p "Leaving: critical_operation"
14     result
15   end
16 end
```

Ruby - aquarium



```
1 class Account
2   attr_reader :balance
3   def credit amount
4     raise "... " unless amount ≥ 0
5     @balance += amount
6   end
7
8   def debit amount
9     raise "... " unless amount < @balance
10    @balance -= amount
11  end
12 end
```

Ruby - aquarium



```
1 require 'aquarium'
2 class Account
3   include Aquarium
4   before :calls_to => [:credit, :debit] do |join_point, object, *args|
5     object.balance = read_from_database ...
6   end
7
8   after_returning_from :calls_to=>[:credit, :debit] do |join_point, object, *args|
9     update_in_database (object.balance,...)
10  end
11
12  before :calls_to => [:credit, :debit] do |jp, *args|
13    raise "..." unless user_authorized
14  end
15
16 end
17
```

Ruby - Aquarium



```
1 class Account
2   attr_reader :balance
3
4   def debit amount
5     raise "... " unless amount < @balance
6     @balance -= amount
7   end
8
9 end
```

Ruby - Aquarium

```
1 class Account
2   attr_reader :balance
3
4   def debit amount
5     @balance -= amount
6   end
7
8 end
```

```
1 attr_accessor :max_retiro
2 before :calls_to=> :debit, :in_type=> :Account do |jp, account, *args|
3   if args[0] > max_retiro
4     raise "Exception"
5   end
6   if (account.balance - args[0]) < account.balance
7     raise "Exception"
8   end
9 end
```


Aplicaciones



Capacidades y objetivos

Spring AOP

- No pretende ser una solución completa del paradigma orientado a aspectos

AspectJ

- AspectJ es la tecnología original de POA, tiene como objetivo proporcionar una solución completa.

Weaving

Spring AOP

- Usa Runtime weaving durante la ejecución usando proxy dinámico JDK o proxy CGLIB

AspectJ

Hace uso de 3 tipos de Weaving:

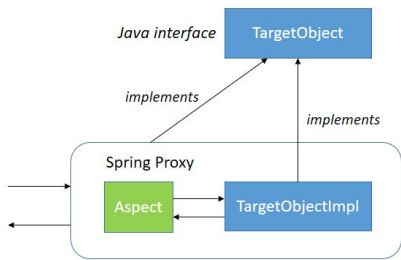
- Compile-Time Weaving
- Post-Compile Weaving
- Load-Time Weaving

Estructura Interna y Aplicación

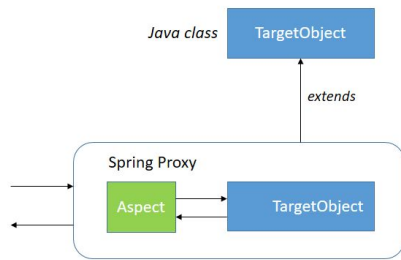
Spring AOP

Spring AOP Process

JDK Proxy (interface based)



CGLib Proxy (class based)



AspectJ



Joinpoints

Spring AOP

- Como está basado en patrones de proxy, necesita subclasificar la clase Java y aplicar aspectos transversales de acuerdo sea acorde.
- NO se puede aplicar a clases finales o a métodos estáticos y finales.

AspectJ

- No requiere subclasificar el objeto de destino
- Debido a que el weaving lo hace en tiempo de compilación puede acceder al sector de código requerido antes que se inicie el tiempo de ejecución

Simplicidad

Spring AOP

- Spring AOP es más simple porque no introduce ningún compilador o weaver adicional en nuestro proceso de compilación.

AspectJ

- Para usar AspectJ, estamos obligados a presentar el compilador de AspectJ (ajc) y volver a empaquetar todas nuestras bibliotecas (a menos que cambiemos el weaving posterior a la compilación o al tiempo de carga).

Simplicidad

Puntos de Corte:

Spring

Referencias de imágenes

- [1] <https://wp-assets.futurism.com/2014/06/Headphones-Tangled.jpg>
- [2] <https://medium.com/java-vault/layered-architecture-b2f4ebe8d587>
- [3] <https://pixabay.com/>
- [4] <https://graphicriver.net/item/tennis-balls-in-plastic-cans-realistic-vector/24624188>
- [5] <http://codingornot.com/wp-content/uploads/2017/05/poa-aop-programaci%C3%B3n-orientada-a-aspectos-aspect-oriented-programming-ejemplo-qu%C3%A9-es-paradigma.png>
- [6] <https://medium.com/@zengcode/spring-aop-aspectj-142289800429>
- [7] <https://www.topjavatutorial.com/frameworks/spring/spring-aop/aspect-oriented-programming-concepts/attachment/aop-concepts/>
- [8] <https://www.clker.com/cliparts/E/v/h/Z/f/3/3-gears-hi.png>
- [9] <http://codingpedagogy.net/images/logo.png>

Referencias de imágenes

[10]<https://previews.123rf.com/images/citadelle/citadelle1201/citadelle120100001/11823276-mara%C3%B1a-de-cables-de-colores-un-a-red-mundial-de-internet.jpg>

[11]https://imgs.developpaper.com/imgs/4032517280-725166783e56f04e_articlex.png

Referencias

- [1] G. Kiczales et al., "Aspect-oriented programming", 1997. Available: <https://www.cs.ubc.ca/~gregor/papers/kiczales-ECOOP1997-AOP.pdf>. [Accessed 7 June 2020].
- [2] R. Laddad, AspectJ in action. Greenwich, CT: Manning, 2004.
- [3] Aspect-Oriented Programming. Retrieved from <https://www.peterindia.net/AOP.html>
- [4] O. Spinczyk, D. Lohmann, M. Urban. AspectC++: an AOP Extension for C++ 2005
- [5] J. Sung. ASPECTUAL CONCEPTS. Northeastern University. 2002

Referencias

[6]<https://profesores.virtual.uniandes.edu.co/~miso4204/dokuwiki/lib/exe/fetch.php?media=temas:aspectj.pdf>

[7]http://www.jtech.ua.es/j2ee/publico/spring-2012-13/apendice_AOP-apuntes.html

[8]<https://sites.google.com/site/programacionhm/conceptos/aop>

[9]https://deanwampler.github.io/polyglotprogramming/papers/Aquarium_RubyAOP.pdf

[10] <https://www.baeldung.com/spring-aop-vs-aspectj>