TUTORIAL DE ASPECTJ

Ángel Mateo González
Rubén Darío Guarnizo
Lorraine Rojas Parra
Alejandra Superlano Esquibel

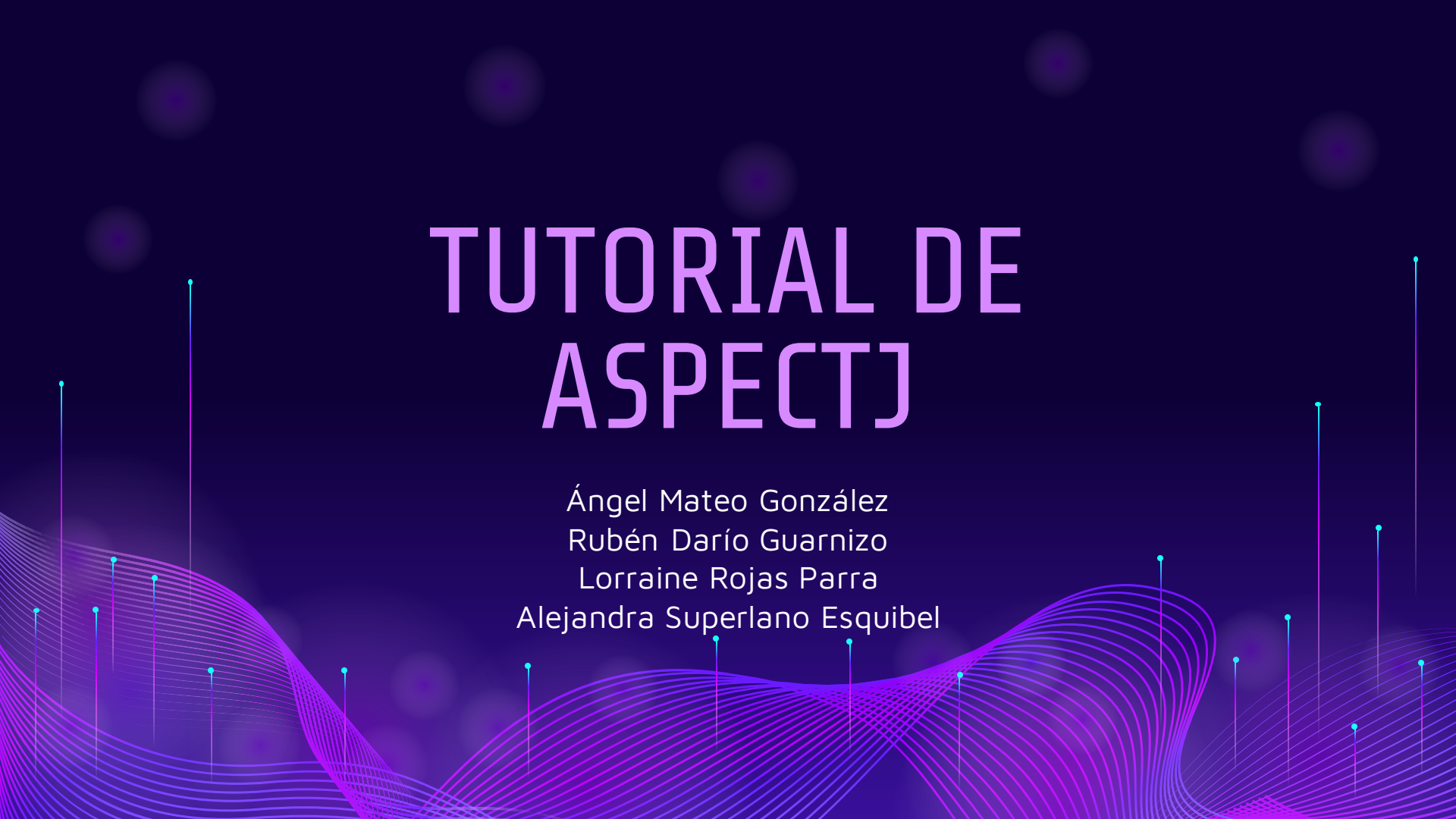
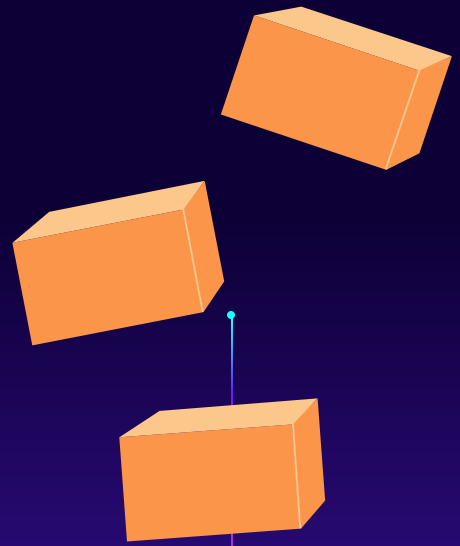


TABLA DE CONTENIDO



01

DEFINICIÓN

AspectJ

¿Qué es?

Lenguaje de programación orientado a aspectos

Historia

- Desarrollado a finales de la década de 1990
- Publicado en 2001

Conociendo a PARC

Empresa de investigación y desarrollo

Usos

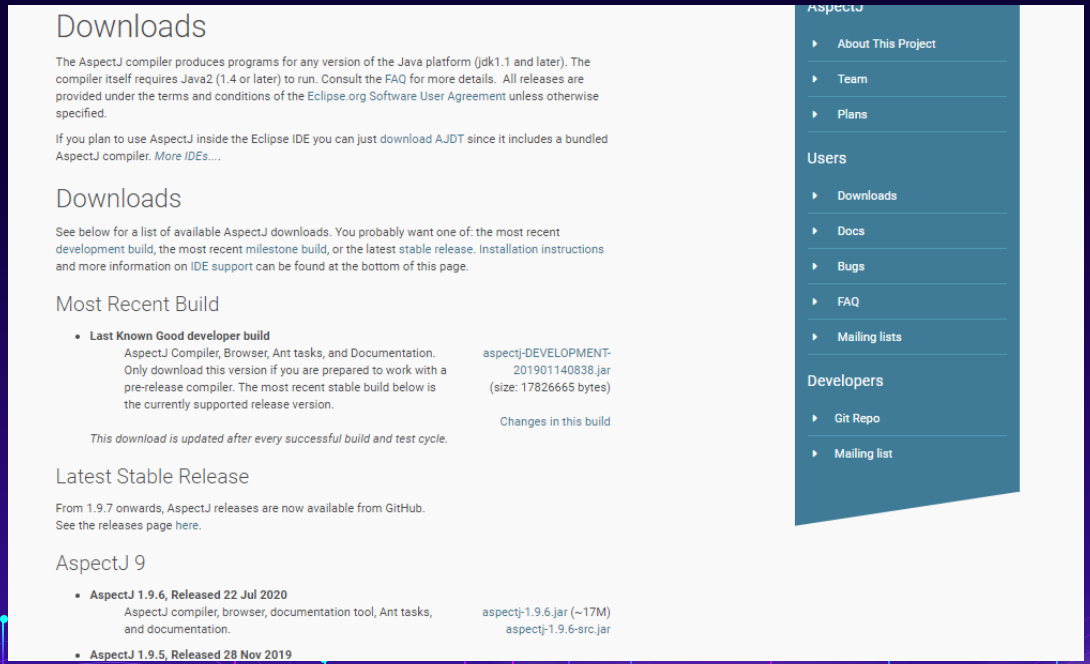
- Frameworks: Spring Roo, Magma, Contract4j
- Monitoreo: perf4j, glassbox
- Servidores: SpringSource

02

PRIMEROS PASOS

PROCESO DE INSTALACIÓN

1. Descargar el compilador desde el enlace https://www.eclipse.org/aspectj/downloads.php



Downloads

The AspectJ compiler produces programs for any version of the Java platform (jdk1.1 and later). The compiler itself requires Java2 (1.4 or later) to run. Consult the FAQ for more details. All releases are provided under the terms and conditions of the Eclipse.org Software User Agreement unless otherwise specified.

If you plan to use AspectJ inside the Eclipse IDE you can just download AJDT since it includes a bundled AspectJ compiler. *More IDEs....*

Downloads

See below for a list of available AspectJ downloads. You probably want one of: the most recent development build, the most recent milestone build, or the latest stable release. Installation instructions and more information on IDE support can be found at the bottom of this page.

Most Recent Build

- **Last Known Good developer build**
  AspectJ Compiler, Browser, Ant tasks, and Documentation. Only download this version if you are prepared to work with a pre-release compiler. The most recent stable build below is the currently supported release version.
  aspectj-DEVELOPMENT-20190114083838.jar (size: 17826665 bytes)
  Changes in this build

This download is updated after every successful build and test cycle.

Latest Stable Release

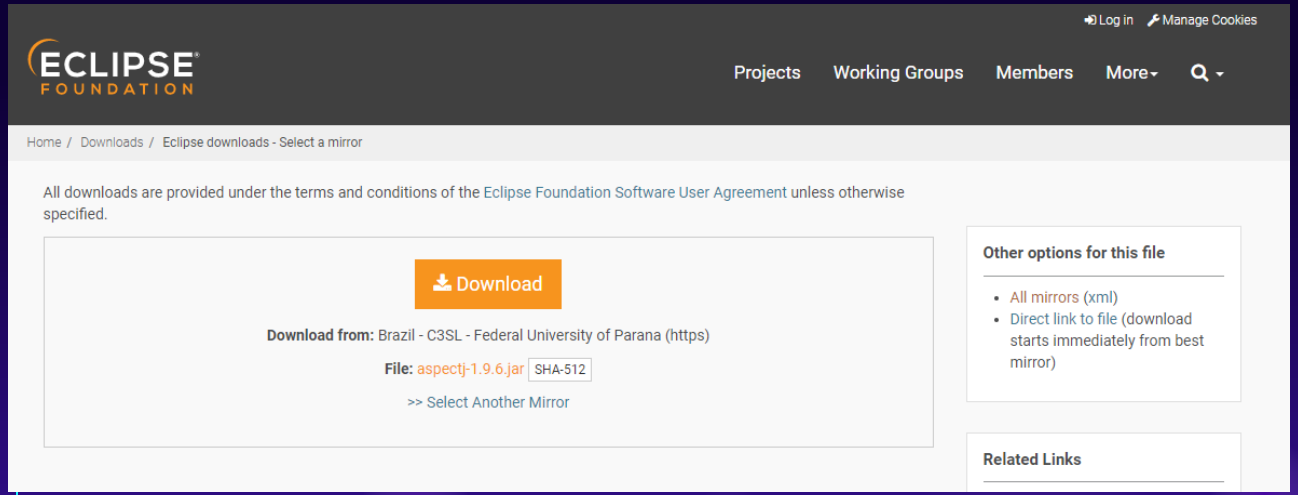
From 1.9.7 onwards, AspectJ releases are now available from GitHub. See the releases page here.

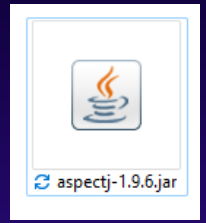
AspectJ 9

- **AspectJ 1.9.6, Released 22 Jul 2020**
  AspectJ compiler, browser, documentation tool, Ant tasks, and documentation.
  aspectj-1.9.6.jar (~17M)
  aspectj-1.9.6-src.jar
- **AspectJ 1.9.5, Released 28 Nov 2019**

PROCESO DE INSTALACIÓN

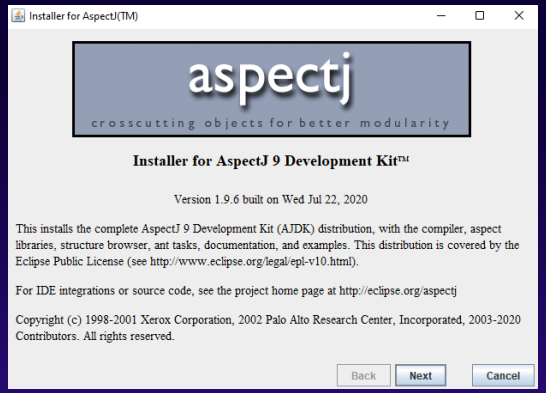
2 Una vez seleccionada la versión a descargar, se abre una nueva página en donde se debe dar click en el botón de ***Descargar***.

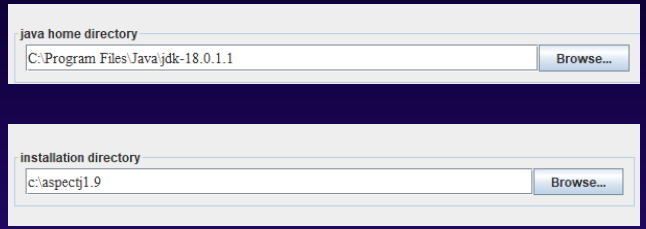


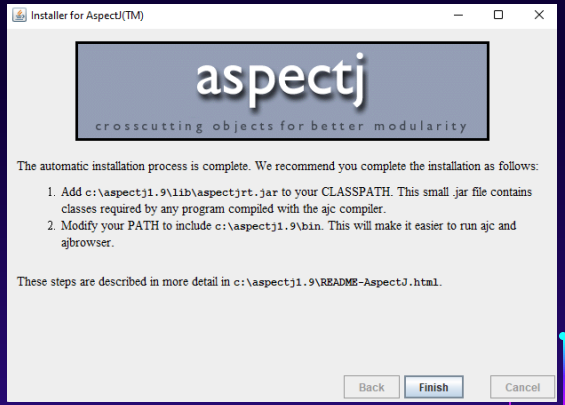


PROCESO DE INSTALACIÓN

3 Al terminar la descarga, se debe hacer doble click en el .jar para iniciar con el proceso de instalación







03

TOUR POR EL LENGUAJE

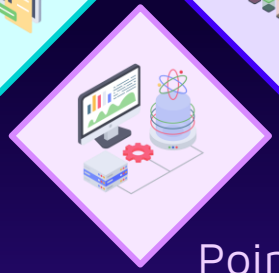
CONCEPTOS BÁSICOS

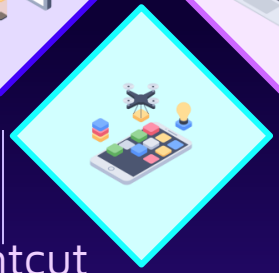
Aspecto

Modularización de una funcionalidad

InterType

Permiten definir los campos y los métodos implementados para múltiples clases.





Joinpoint
(Puntos de enlace)

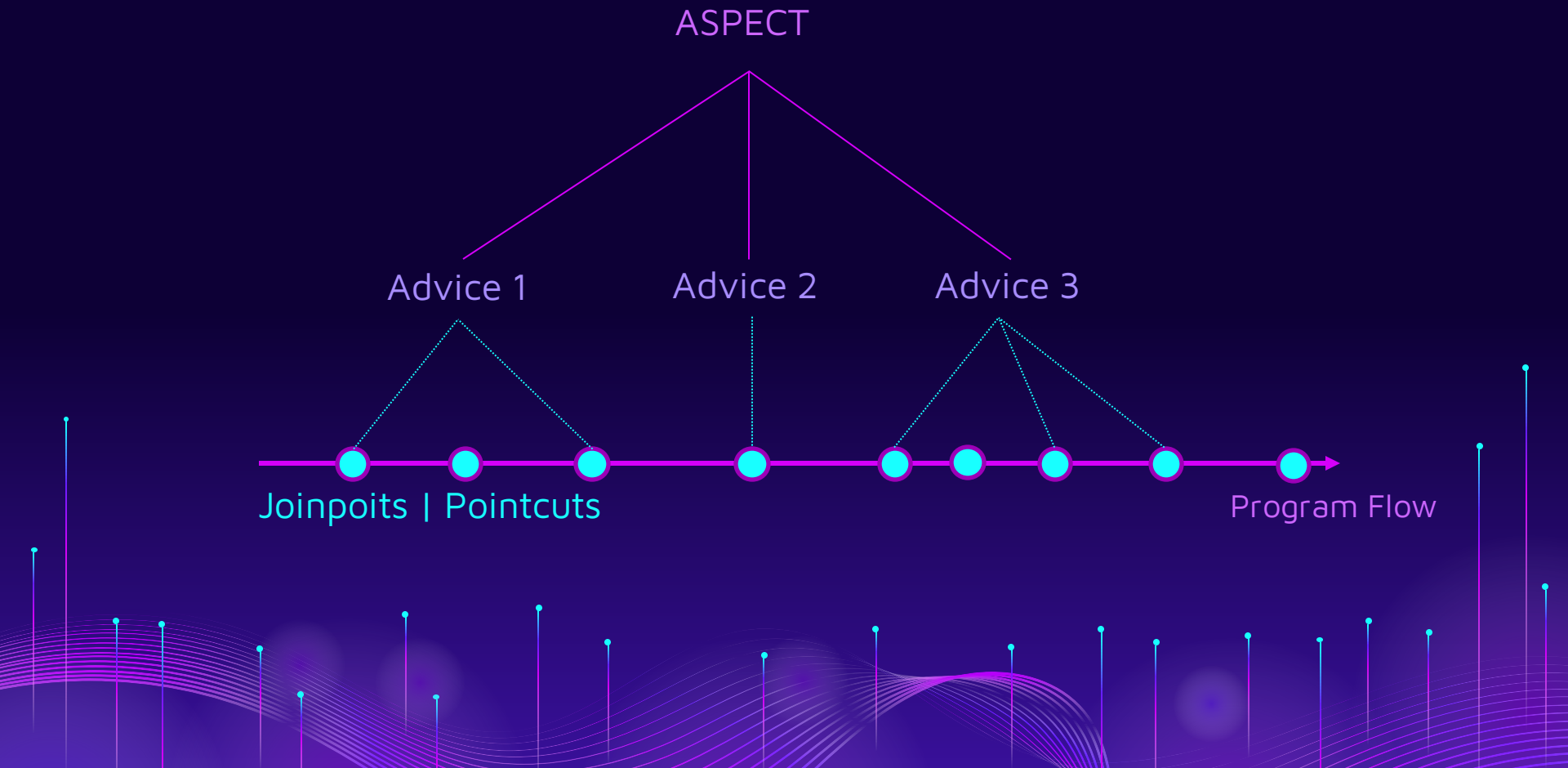
Punto durante la ejecución de un script

Pointcut
(Puntos de corte)

Grupo de joinpoints concatenados lógicamente

Aviso
(Advice)

Inyectan un nuevo comportamiento en los joinpoints


ASPECT
Advice 1
Advice 2
Advice 3
Joinpoits | Pointcuts
Program Flow

ASPECTOS

SINTÁXIS DE UN ASPECTO

```
[privileged] aspect id [extends Type] [implements Typelist] [PerClause] {Body}
```

Privilegios

Permite acceder a recursos privados o protegidos de otros tipos

PerClause

Definen cómo se instancia y asocia el aspecto

- issingleton()
- perthis(pointcut)
- pertarget(pointcut)
- percflow(pointcut)
- percflowbelow(pointcut)

Herencia

Puede extender de una clase, u otro aspecto abstracto

Condiciones

- No se puede crear una instancia del mismo con el método *new*.
- Una clase no puede extender un aspecto.
- Un aspecto no puede extender un aspecto que no sea abstracto

EJEMPLO

```
public class Client
  {
      public static void main(String[] args) {
          Client c = new Client();
      }
  }

  aspect Watchcall {
      pointcut myConstructor(): execution(new(..));

      before(): myConstructor() {
          System.err.println("Entering Constructor");
      }
  }
```

Declaración sencilla de un aspecto

```
class C {
      private int i = 0;
      void foo() { }
  }
  privileged aspect A {
      private int C.i = 999;
      before(C c): call(void C.foo()) target(c) {
      System.out.println(c.i);
      }
  }
```

Declaración de un aspecto privilegiado

JOINPOINTS

En AspectJ todo gira alrededor de los **puntos de enlace**

SINTÁXIS DE UN JOINPOINT

Join Point

Elementos comodines

*	..	+

1010
1010

Operadores binarios

!	||	&&

EJEMPLO

public MyClass.new(..)	Todos los constructores de la clase MyClass que tomen cualquier número de argumentos.
* *.*(..) throws SQLException	Cualquier método que lance SQLException.
public MyClass.new(Integer)	Constructores públicos de la clase MyClass que tomen un único argumento de tipo Integer.
public void Stack.*()	Todos los métodos en de la clase Stack que tengan acceso público devuelvan void y no tengan argumentos.
public void Stack.clean()	El método clean() de la clase Stack que tenga acceso público, devuelva void y no tenga argumentos.
*Manager	Tipos con un nombre que termine en Manager.

Patrones de puntos de enlace

El significado de los caracteres comodines depende si se aplican a tipos, métodos o atributos

INTER TYPES

INTER TYPES

01

Permiten que un aspecto declare miembros que son parte de otros tipos (Clases o aspectos)

```
private boolean Server.disabled = false;
```

POINTCUTS

POINTCUT

Constructores

Especifican los puntos de enlace y se captura información referente al contexto

Entrelazado

Combinar los comportamientos definidos en los aspectos con el código original

Avisos

Se asocian para ejecutar joinpoints

Declaración

Anónimos o con un nombre

SINTÁXIS DE POINTCUTS CON NOMBRE

01

Los puntos de corte que se definen con un nombre pueden después ser referenciados desde múltiples partes del código

```
pointcut nombredelpointcut(argumentos): definición
```

Definición

Nombre

```
pointcut operacionesPublicas(): call(public *.*(..));

before() : operacionesPublicas(){
//acciones a realizar antes de las llamadas a las funciones públicas
}
```

SINTÁXIS DE POINTCUTS ANÓNIMO

02

Se definen en el lugar donde se usan y no pueden reusarse

```
[especificacion-del-advice]: definicion-del-pointcut
```

```
pointcut operacionesPublicas(): call(public *.*(..));

before() : operacionesPublicas(){
//acciones a realizar antes de las llamadas a las funciones públicas
}
```

```
before() : call(public *.*(..)){
//acciones a realizar antes de las llamadas a las funciones
públicas
}
```

SINTÁXIS DE LOS POINTCUTS

```
<Handler>(<Return type> <Package>.<Method Names>(<Parameters>))
```

Handler

- Execution
- Call
- Handler
- Get | Set
- Target
- Whitin
- Cflow(after | before)

Return

- Int
- Boolean
- String
- Object
- *: Any

Clase | Paquete | Metodos

- [a-zA-Z0-9] Selector de nombre de paquete, clase o nombre metodo

Parámetros

- Type
- , Type
- .. : Any
- Type

COMBINAR POINTCUTS

```
POINTCUT<MOD>ANOTHEPOINTCUT<MOD>...
```

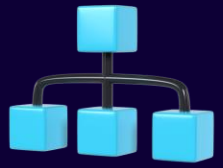
Boolean MOD

- &&: and
- || : or
- ! : not

Podemos incluir varios pointcuts para seleccionar según una expresión booleana.

Los modificadores son expresiones booleanas

```
call(* *(..)) && (within(Line) || within(Point))
```



TIPOS DE POINTCUTS

Relacionados con métodos

Relacionados con campos (atributos)

Relacionados con el manejo de excepciones

Relacionados con la ejecución de los advices

Basados en el estado

Basados en el control de flujo

Basados en condicionales

Basados en la localización

EJEMPLOS DE POINTCUTS

1

```
//Selecciona Todos los JoinPoint y dispara al ejecutar
execution(* *(..))
```

2

```
// Se ejecuta cuando las funciones en la clase Employers arrojan una excepción ArrayIndexOutOfBound.
within(Employers) && handle(ArrayIndexOutOfBounds)
```

3

```
//Seleciona los metodos que contengan <set> en su nombre, y ejecuta al llamar.
call(* set(..))
```

4

```
//Metodos con nombre new de cualquier Clase y dos parametro int.
call(*.new(int, int))
```

5

```
//Seleciona todos metodos que retornen int
execution(int *())
```

6

```
//Cualquier Metodo publicos no estatico.
execution(public !static * *(..))
```

EJEMPLOS DE POINTCUTS COMPLEJOS

1

```
//Metodos en instancias Point, en
metodos setX o setY, con params int, y
retornen void.
pointcut setter(Point p): target(p) &&
         (call(void setX(int))||
          call(void setY(int)));
```

2

```
// Llamado cuando se supere el tamaño del buffer.
pointcut overflow(): if(Buffer.size() > MaxAllowed)
```

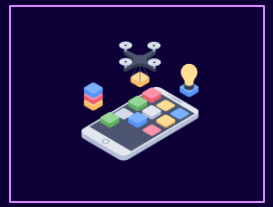
3

```
// Cuando el primer y ultimo param sea String e int
pointcut first_last(): args(String,..,int) ||
args(int,..,String)
```

Por una chocolatina...

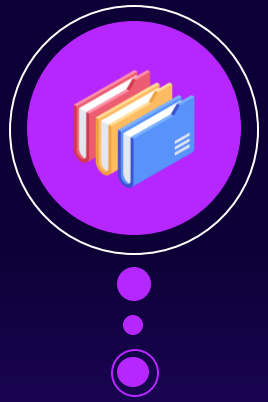
Pregunta

```
pointcut whatis(Point p1,Point p2) :target(p1) &&
                                    args(p2)&&
                        call(boolean equals(Object));
```



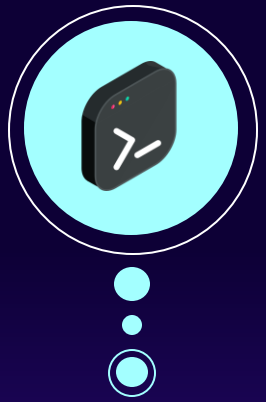
AVISOS

AVISOS



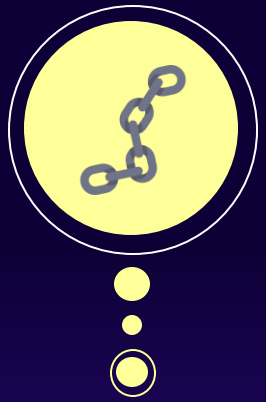
Tipos

Antes, después o "alrededor"



Lógica

Similar al cuerpo de un método



Entrelazado

En conjunto con los puntos de corte

SINTÁXIS DE UN AVISO

Definición de un PdC

- Declarar puntos de enlace
- Referencia a uno previamente definido

```
tipo_de_aviso (...) excepcion_o_retorno : punto_de_corte {
    // cuerpo
}
```

Tipo

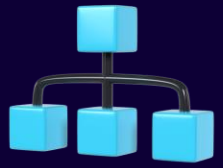
- before
- after
- around

Información contextual

- Vacío
- Declarar objeto
- Argumentos

Resultado

- Vacío
- Lanzar una excepción
- Retornar un resultado

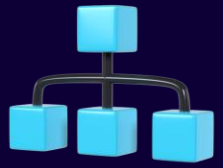


TIPOS DE AVISOS

1 before

```
pointcut connectionOperation(Connection connection):
    call(* Connection.*(..) throws SQLException)

before(Connection connection): connectionOperation (connection){
    checkConnectionValidity(connection);
}
```



TIPOS DE AVISOS

2 after

- returning

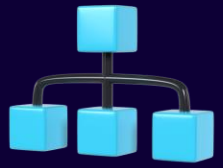
```
after () returning (Object x) : connectionOperation (connection) { ... }
```

- throwing

```
after () throwing (tipo_excepcion exc)  : connectionOperation (connection) { ... }
```

- default

```
after () : connectionOperation (connection) { ... }
```



TIPOS DE AVISOS

3 around

```
Object around(): memoryOperations() {
  try {
    proceed();
  } catch(Exception e) {
    /*Manejar la excepción e */
  }
}
```

Por una galleta...

¿Con cuál tipo de aviso se modifica la ejecución del código?

04

VENTAJAS Y DESVENTAJAS

VENTAJAS

1. AspectJ permite modelar referencias transversales
2. Es un lenguaje conciso y explícito diseñado con el fin de ofrecer las ventajas de la modularidad.
3. AspectJ está siendo usado en la actualidad con mucho éxito en proyectos reales de todos los tamaños
4. Ayuda a crear un sistema mantenible
5. La lectura de código es mas sencilla

DESVENTAJAS

1. Sin una herramienta apropiada, el programador no comprenderá la funcionalidad del sistema completo
2. Los aspectos pueden ser mal utilizados. Un programador puede implementar una falsa funcionalidad.
3. El paradigma puede emplearse mal quitando y/o delegando responsabilidades a ciertas clases
4. Puede introducir nuevos errores y fallas de seguridad si no se implementa adecuadamente

05

PARTICULARIDADES

ENTRELAZADO

Generación de programas → Salida → Java + POA *weaving process*

Se ve afectado el comportamiento global o una parte de la aplicación

DINÁMICO

Reglas de entrelazado

CROSSCUTTING

ESTÁTICO

Causado por modificaciones en la estructura del programa

COMPILADOR

El compilador **ajc** de AspectJ genera byte-codes compatibles con cualquier máquina virtual de Java

```
org.aspectj.tools.ajc.Main
```

GENERACIÓN DE DOCUMENTACIÓN

GRACIAS POR SU ATENCIÓN