Programación Orientada a Aspectos – Continuación

Hasta ahora hemos visto 2 tipos de advise
"after" y "before"

***Before*:** este tipo de advice se ejecuta antes de la ejecución del JP capturado. Si se dispara una excepción en el advice, la operación capturada por el JP no se ejecutará.
El advice before se utiliza para llevar a cabo controles de pre-condición, logging, autenticación, etc.

After: este tipo de advice se ejecuta después de la ejecución del JP. Dado que suele ser importante distinguir entre un retorno normal de un JP y aquellos que disparan una excepción AspectJ provee 3 variaciones de este advice:

```
after( ) returning / after( ) returning(<ReturnType  returnObject>)
after( ) throwing / after() throwing (<ExceptionType exceptionObject>)
after( )
```

```
after() returning : call(* Mensaje.*(..)) {
... registrar la terminación con éxito
}
```

El advice será ejecutado después de que se complete con éxito la llamada a cualquier método de la clase Mensaje. Si el método capturado dispara una excepción el advice NO se ejecutará.
Se puede utilizar la forma que contempla el objeto de retorno cuando se desea capturar el objeto que es retornado por el advice.

```
after() throwing : call(* Mensaje.*(..)) {
... registrar la falla
}
```

El advice será ejecutado solamente despues de la llamada a cualquier método de la clase Mensaje, cuando el JP dispara una excepción. Si el método retorna normalmente el advice NO se ejecutará.
Se puede utilizar la forma que contempla el objeto de retorno cuando se desea capturar el objeto excepción. En este caso se puede utilizar el objeto para tomar decisiones en el advice.

Existe un tercer tipo de advise que es: "around".

Around
Este tipo de advice "rodea" la ejecución del JP. Es apropiado para los casos en que se necesita evitar la ejecución, continuarla o incluso ejecutar con un contexto alterado.
Cada advice "around" debe declarar un tipo de retorno, que puede ser void. Generalmente se declara del tipo de resultado del JP que corresponde. Por ejemplo si el método considerado retorna un String, entonces habría que declarr que el advice retorna un String. Si puede retornar distintos tipos dependiendo del objeto receptor, entonces habria que declararlo "Object".

Cuando se utiliza un advice de este tipo es posible modificar la ejecución de código en el JP, puede reemplazarla, o puede saltarla. Tambien puede ejecutar la operación original con un contexto alterado. Para ejecutar el método capturado en el PC se utiliza `proceed()` en el cuerpo del advice. Si no se llama a `proceed()`, el JP será pasado sin actuar sobre él.

Cuando se utiliza proceed() se puede pasar el contexto obtenido por el advice como argumentos a la operación capturada por el PC, o pueden pasarse argumentos diferentes. Lo importante es que el tipo y número de argumentos coincida, y también el tipo de retorno. La lógica del advice requiere que se tenga un tipo de retorno. Cuando el tipo de retorno es un tipo primitivo, se utilizan las clases wrapper.

Ejemplo:

```
String around(Mensaje elMensaje): execution (String *.getMensaje()) &&
                                  target(elMensaje) {     (1)
      if (turno == 0) {
         turno = 1;
         System.out.println ("paso por el around " +
                             elMensaje.toString() + "con get mensaje");
         return proceed(elMensaje);   (2)
      }
      else {
         turno = 0;
         System.out.println ("paso por el around sin éxito " +
                             elMensaje.toString());   (3)
         return ("");
      }
}
```

En este ejemplo se muestra un advice “araound” que toma un parámetro “elMensaje” de tipo Mensaje, y captura la ejecución de un método *getMensaje* de cualquier clase, que devuelva un *String,* y liga al parámetro elMensaje con el objeto target, o sea el objeto sobre el que actua el método getMensaje que fue capturado por el advice (1).
A los efectos de mostrar el funcionamiento de este tipo de advice se utiliza una variable local del aspecto “turno” para controlar si corresponde o no ejecutar el método. Es decir va alternando entre llamadas, una vez “procede” con la ejecución (2) , una vez no procede (3). Notar que el advice retorna algo del tipo de lo que retorna el método ejecutado.

Las implementaciones de advice en ocaciones requieren acceder a los datos del JP, que es la información del contexto. El contexto puede ser pasado a un advice y recuperado en su codigo. Para acceder a esa información se dispone de los PC this( ), target( ) y args( ).
Hay 2 formas de especificar estos PC: usando el tipo de los objetos o usando el identificador de los objetos. En el caso de necesitarlos en un advice se utiliza el identificador del objeto.

En el ejemplo que sigue, en el advice before, se ve un PC anónimo que colecta todos los argumentos asociados con el método.

```
before (Account cuenta, float cantidad):
  call (void Account.credit(float))
  && target (cuenta)
  && args (cantidad) {
    System.out.println("depositando" + cantidad + "en la cuenta" + cuenta);
}
```

El contexto es pasado entre el PC anónimo y el advice.
target( ) colecta los objeots sobre los que se invoca el método
args( ) captura los argumentos del método
this( ) captura el objeto de ejecución corriente

El cuerpo del advice utiliza la información del contexto de la misma forma que se hace en un método. Cuenta y cantidad son los identificadores de objetos.

El mismo advice pero utilizando un PC con nombre queda:

```
pointcut operación (Account cuenta, float cantidad):
                        call (void Account.credit(float))
                        && target (cuenta)
                        && args (cantidad)

before (Account cuenta, float cantidad): operación (cuenta,cantidad)
{
      System.out.println("depositando" + cantidad + "en la cuenta" +
cuenta);
}
```

Al utilizar PC con nombres, ellos son los que deben tomar la información de contexto, y pasarla al advice.

```
after() returning(String nota): execution (String *.getMensaje())
            {     System.out.println("el valor de retorno es " + nota);
            }
```

En este caso el parámetro *nota* que es de tipo String captura el objeto de retorno del método getMensaje