

## Pointcut args()

- Il pointcut `args()` fornisce i valori degli argomenti passati al metodo catturato dall'altro pointcut specificato

Oggetti passati al metodo invocato

```
pointcut move(int x, int y) :  
  args(x, y) &&  
  call(void Figure.moveBy(int, int));  
  
after (int x, int y) returning : move(x, y) {  
  <x e y sono i valori passati a moveBy(>  
}
```

Ing. E. Tramontana - AOP - 9-Nov-06

1

## Advice

- Implementano le azioni aggiuntive da eseguire ai join point catturati
- Tipi di advice; before, after, around
  - Prima di procedere al join point: `before()`
  - Dopo il ritorno di un valore dal join point: `after() returning`
    - Per catturare il valore di ritorno `after() returning(Type o)`
  - Dopo aver lanciato un throwable al join point: `after() throwing`
  - Al ritorno di un join point in qualunque modo: `after()`
  - All'arrivo al join point, prende il controllo e decide se continuare l'esecuzione: `around()`
    - Per continuare l'esecuzione di ciò che è catturato: `proceed()`
    - Deve avere un valore di ritorno (poiché può sostituire l'esecuzione)

Ing. E. Tramontana - AOP - 9-Nov-06

3

## Pointcut this()

- Per catturare tutti i join point dove `this` è instanceof `Account`: `this(Account)`
  - `Account` è una classe
- Differisce da `within()` poiché
  - Cattura i join point durante l'esecuzione delle istanze di `Account` o di qualche sottoclasse
  - Non cattura l'esecuzione all'interno di classi nested
  - Non cattura metodi static
- Altro uso: `this(anAccount)`
  - L'argomento specificato (`anAccount`) è una variabile
  - Fornisce il riferimento dell'istanza su cui avviene la cattura
    - Si comporta analogamente a `target()`, ed esattamente come `target()` se il pointcut è di tipo `execution`

Ing. E. Tramontana - AOP - 9-Nov-06

2

## Advice

- Formato di un advice before, analogo ad after e around

```
before([param]) : unPointcut([param]) {  
  operazioni  
}
```

Ing. E. Tramontana - AOP - 9-Nov-06

4

## Advice before() e after()

```
pointcut trace() : <qualche join point>
```

```
before() : trace() {  
    System.out.println(" catturato: "+thisJoinPoint.getSignature());  
}
```

```
after() : trace() {  
    System.out.println(" catturato: "+thisJoinPoint.getSignature());  
}
```

## Advice around()

```
public aspect Check {  
    pointcut checkValore(int v) :  
        args(v) &&  
        (call(void Point.setX(int)) ||  
         call(void Point.setY(int)));  
  
    void around(int v) : // dichiarazione advice  
        checkValore(v) { // pointcut  
        if (v < 9) v = 10; // body  
        proceed(v); // esegue metodo catturato  
        // do something  
    }  
}
```

Valore di ritorno per around()

## Advice around()

- Il valore di ritorno non void deve essere passato con return dall'advice

```
int around() : check() {  
    // do something  
    return proceed(v);  
}
```

## Pointcut con wildcard

- Per catturare join point con caratteristiche comuni si usano le wildcard
- Per identificare qualunque metodo println() di PrintStream che ritorna un void e prende un argomento di qualunque tipo  
void java.io.PrintStream.println(\*)
- Per identificare qualunque metodo public di Figure  
public \* Figure.\*(..)
- Per identificare qualunque metodo di Line il cui nome inizia con set  
public \* Line.set\*(..)
- \* indica un numero qualsiasi di caratteri escluso il punto
- .. Indica un num. qualsiasi di caratteri incluso un qualunque num. di punti
- + indica qualunque sottoclasse o sottointerfaccia di un tipo (es. Figure+)

## Esempi di uso di wildcard

- Per indicare qualsiasi tipo che termina per Account  
\*Account
- Per indicare il tipo Date in qualsiasi package sotto il package java  
java.\*.Date
- Per indicare qualsiasi tipo all'interno del package java o di qualche subpackage  
java..\*
- Per indicare tutti i tipi all'interno del package javax che terminano con Model e tutti i loro sottotipi  
javax.\*Model+
- Per catturare tutte le esecuzioni di metodo invocate  
execution (\* \*.\*(..))

## Aspetto per tracing

```
package file;
import java.io.*;
public aspect TraceV1 {
    pointcut trace() :
        execution (* *.*(..)) && // cattura tutto
        !within(TraceV1) && // fuori dall'aspetto stesso
        within(file..*); // dentro il package
    before() : trace() {
        System.out.println("** prima: "+thisJoinPoint.getSignature());
    }
    after() : trace() {
        System.out.println("** dopo: "+thisJoinPoint.getSignature());
    }
}
```

## Pointcut per i campi

- Un campo x della classe Rectangle è identificato con  
public int Rectangle.x
- Tutti i campi di Account sono identificati con  
\* Account.\*
- Per intercettare la lettura di un campo di Account  
get(\* Account.\*)
- Scrittura: set()

## Pointcut basati sul flusso

- I pointcut possono catturare i join point basandosi sul flusso del controllo (ciò che è invocato) dei join point catturati da un altro pointcut
- Per catturare tutti i join point dal momento in cui un certo pointcut specificato è catturato: cflow(pointcut)
- Es. per catturare tutti i join point a partire dalla chiamata a debit() di Account: cflow(call(\* Account.debit(..)))
- Per escludere il join point del pointcut specificato: cflowbelow(pointcut)

## Esempio cflow()

```
public aspect TraceV2 {
    pointcut trace() :
        // cattura tutto, a partire dall'esecuzione di Store.write()
        cflow(execution(void Store.write(..)) &&
            !within(TraceV2); // fuori dall'aspetto stesso

    before() : trace() {
        System.out.println("* "+thisJoinPoint.getSignature());
    }
}
```

## Pointcut basati su struttura

- I pointcut basati sulla struttura lessicale del codice `within()` e `withincode()`, catturano tutti i join point di classe, aspetto o metodo specificato
- Per catturare tutti i join point della classe `Account`:  
`within(Account)`
- Per catturare tutti i join point del metodo:  
`withincode(* Account.debit(..))`