AOP(Apsect Oriented programming) Pointcut declarations ::

In case if we have a question of how to reuse a pointcut expression? It mean how to apply point cut expression on multiple advices in our AOP Aspect.

We can achieve this in a traditional method like give below

1) @Before(“execution(\* it.cogni.com.\*.\*(..))”)

Public void beforeAddAccountAdvice(){

}

2) @Before(“execution(\* it.cogni.com.\*.\*(..))”)

Public void beforeAddMembershipAdvice(){

}

The second way is to create a point cut declaration and apply pointcut declaration to any advices within the application.

Example of declaring a pointcut ::

@Pointcut(“execution(\* it.cogni.com.dao.\*.\*(..))”)

Private void onDaoPackage(){}

Example Code snippet ::

@Aspect

@Component

Public class MyAspectClass{

@Pointcut(“execution(\* it.cogni.com.dao.\*.\*(..))”)

Private void onDaoPackage(){}

@Before(“onDaoPackage()”)

Public void beforeAddAccountAdvice(){

}

@Before(“onDaoPackage()”)

Public void beforeAddMembershipAdvice(){

}

}

The benefits of declaring pointcut at one location and reusing at multiple advices is that , if we need any modification in pointcut then it can be done only at one place and will be applied to multiple advices. And can also share and combine pointcut expressions.

## Now we have to have a look on how to apply multiple pointcut expressions on single advice?

## And as well how to execute an advice only if certain conditions are met

## for example if we want to apply advices on all the package method except the print methods or getter and setter method if available.

## We can combine pointcut expressions by using logic operators like

* AND (&&)
* OR (||)
* NOT (!)

## Using the above logical operators we can make the pointcut expressions to work as if it is written in if conditions, and the expressions will be only evaluated if it returns true.

@Before(“expressionOne() && expressionTwo()”)

@Before(“expressionOne() || expressionTwo()”)

@Before(“expressionOne() && !expressionTwo()”)

**Ordering Aspects ::**

If we want to order the advices then we need to separate the “Aspects” then to use

**@Order** annotation for fine grained control on the execution of advices.

**For Example ::**

@Order(1)

Public class MyFirstAspect {

Have some advices

}

We can order the aspects by providing the numeric values to the annotation under parenthesis. The precedence will be given to the lower values. Here negative numbers are allowed.

Example ::

@Aspect

@Component

@Order(1)

Public class MyFirstAspect{

Have some advice

}

=========================

@Aspect

@Component

@Order(2)

Public class MySecondAspect{

Have some advice code

}

**After returning Advice ::**

This Aspect is executed after the business method is executed while returning the response or output. Her post processing of data is possible before returning the response to the caller.

Most common use cases of “After returning” advice is logging, security, transactions as well Audit logging.

Example ::

@AfterReturning(“execution(\* it.com.dao.StudentDAO.addStudent(..))”)

Public void afterReturningAddStudentMethodAdvice(){

System.out.println(“Executing @AfterReturning advice”);

}

In case if we want to use the business method returned values(after successful execution) in advices, then the advice definition will be as follows

Example ::

@AfterReturning(pointcut=“execution(\* it.com.dao.StudentDAO.addStudent(..))”,

Returning=”result”)

Public void afterReturningAddStudentMethodAdvice(JoinPoint jonpoint,String result){

System.out.println(“Executing @AfterReturning advice”);

}