For using around advice in a class it must implement Method Interceptor

AOP provides simplified way to apply with cross-cutting concerns.

Cross cutting concerns : any non-functional requirement that you might have to take care whenever you are building your application.

Examples :

1. Logging
2. Transaction Management
3. Security
4. Auditing
5. Locking
6. Event Handling

We can integrate spring with AspectJ framework or we can use spring AOP.

Join Points:

Well defined point during execution of application where you can insert additional logic for cross-cutting. Example of join point : Method invocation, class and object initialization

AspectJ and Spring AOP supports only one type of join point and that is method invocation.

Advice : The code that is executed at a particular join point

Type of advice :

1. Before advice
2. After advice
   a. After returning
   b. After Throwing
3. Around advice (combination of 1 and 2)

Pointcuts : subset/collection of joinpoints

```
Class A
{
M1()
M2()
}
Class B
{
A1()
A2()
A3()
}
```

Total no of jointpoints : 5 (no of methods)

If I want to insert cross cutting on all the methods which ends with 1 and 2 (condition based) then this will be number of pointcuts.

Aspect : it is combination of advice (What to do that code) and pointcut(where to do) Aspect or pointcutAdvisor is a class which will contain the details of advice and pointcuts.

Weaving : process of actually inserting aspects into application code at appropriate points It can be in three ways in spring (compile time—inserting at compile time, bytecode weaving—inserting while bytecode generation , dynamic proxy based weaving –spring provides a proxy at that time aspect details will be given to insert). Spring supports only dynamic proxy based weaving. you can enter aspects only dynamically in spring

Target(pojo class) : An object whose execution is modified by some AOP process is called Target or advised object.

To use around Advice class has to implement method Interceptor interface.

Concept of Introduction in Spring : Process by which you can modify the structure

Of an object by introducing additional methods, fields etc…..

Once it comes into picture we can pass more objects/fields in berore or after method

Currently this is not fully supported by spring. In release 4.0 it might support.

Resource.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

	<!-- CONFIG my main proxy -->
		<!-- Two important things that need to supply to proxy Target and
ASpect -->
	<bean id="businessLogicBean"
class="org.springframework.aop.framework.ProxyFactoryBean">
	<property name="proxyInterfaces"
value="com.spring.aop.IBusinessLogic"></property><!-- proxy can do weaving only
through interface -->
	<property name="target">
	<ref bean="beanTarget"/>
	</property>

	<property name="interceptorNames"> <!-- List of pointcut advisor/aspects is
supplied here -->
	<list>
	<value>tracingBeforeAdvisor</value>
	<value>tracingAfterAdvisor</value>
	<value>loggingThrowsAdvisor</value>
```

```
	</list>
	</property>

	</bean>
	<!-- bean definition for concrete class /target classCLASS --><!-- bean
definition for target -->
	<bean id="beanTarget" class="com.spring.aop.BusinessLogic"></bean>

	<!-- bean definition for ADVICE -->
	<bean id="theTracingAfterAdvice"
class="com.spring.aop.TracingAfterAdvice"></bean>

	<bean id="theTracingBeforeAdvice"
class="com.spring.aop.TracingBeforeAdvice"></bean>

	<bean id="theLoggingThrowsAdvice"
class="com.spring.aop.LoggingThrowsAdvice"></bean>

	<!-- Advisor pointcut definition for before advice -->

	<bean id="tracingBeforeAdvisor"
class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">

	<property name="advice">
	<ref bean="theTracingBeforeAdvice"/>
	</property>

	<property name="pattern" value=".f*.*"></property>
	<!-- Here pattern .* indicates : .means any character and * means that
character can occur zero or any number of times . Maching this expression we have
only two method/pointcut that is foo and bar. So before advice be applied to both
of them -->
	<!-- the class has a property called pattern using which you can supply
/select pointcut by writing regular expression -->

	</bean>
	<!-- Advisor pointcut definition for after advice -->
	<bean id="tracingAfterAdvisor"
class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">

	<property name="advice">
	<ref bean="theTracingAfterAdvice"/>
	</property>

	<property name="pattern" value=".*"></property>
	<!-- Here pattern .* indicates : .means any character and * means that
character can occur zero or any number of times . Maching this expression we have
only two method/pointcut that is foo and bar. So before advice be applied to both
of them -->
	<!-- the class has a property called pattern using which you can supply
/select pointcut by writing regular expression -->

	</bean>

	<!-- Advisor pointcut definition for throws advice -->
	<bean id="loggingThrowsAdvisor"
class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
```

```
        <property name="advice">
        <ref bean="theLoggingThrowsAdvice"/>
        </property>

        <property name="pattern" value=".*"></property>
        <!-- Here pattern .* indicates : .means any character and * means that
 character can occur zero or any number of times . Maching this expression we have
 only two method/pointcut that is foo and bar. So before advice be applied to both
 of them -->
        <!-- the class has a property called pattern using which you can supply
 /select pointcut by writing regular expression -->

        </bean>


 </beans>
```

SpringAOP Full Example

Arithmetic.java

```
package aspectMethods;



public class Arithmetic {
	public Arithmetic() {
		super();
		System.out.println("Arithmetic constructor");
	}
	public int add(int a, int b)
	{
		return (a+b);
	}
	public int subtract(int a, int b)
	{
		return (a-b);
	}
	public int multiply (int a, int b)
	{
		int result=0;
		if(a==0 && b==0)
		{
			throw new ArithmeticException();
		}
		else
		{
			result=a*b;
		}
		return result;
	}
}
```

MyAdvice.java

```java
package beans;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
//"execution(public * *(..))"-- this means public method,any name,any return type and
any number of parameters
@Aspect    //The class will act as Aspect class here all aspects will be defined.
public class MyAdvice {
	@Before("execution(public * *(..))")
	public void beforeExecution( JoinPoint joinPoint)
	{
		System.out.println("Before  "+joinPoint.getSignature()+" Method call");
	}

	@AfterThrowing("execution(public * *(..))")
	public void checkException(JoinPoint joinPoint)
	{

		System.out.println("Exception is thrown....."+joinPoint);
	}


	@AfterReturning("execution(public * *(..))")
	public void afterReturning( JoinPoint joinPoint)
	{
		System.out.println("After  "+joinPoint.getSignature()+" Method Return");
	}

}
```

Configuration.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
	  http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
	  http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">


	<aop:aspectj-autoproxy /> <!--Enables AOP  -->

	 <bean id="arithmetic" class="beans.Arithmetic"></bean>
```

```java
package main;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import beans.Arithmetic;
```

Output

```
Arithmetic constructor
Before  int beans.Arithmetic.add(int,int) Method call
After  int beans.Arithmetic.add(int,int) Method Return
25
Before  int beans.Arithmetic.subtract(int,int) Method call
After  int beans.Arithmetic.subtract(int,int) Method Return
1
Before  int beans.Arithmetic.multiply(int,int) Method call
Exception is thrown.....execution(int beans.Arithmetic.multiply(int,int))
java.lang.ArithmeticException is caught in main class
```