@ASPECTJ BASED AOP WITH SPRING

@AspectJ refers to a style of declaring aspects as regular Java classes annotated with Java 5 annotations. The @AspectJ support is enabled by including the following element inside your XML Schema-based configuration file.

```
<aop:aspectj-autoproxy/>
```

You will also need following AspectJ libraries on the classpath of your application. These libraries are available in the 'lib' directory of an AspectJ installation, otherwise you can download them from the internet.

- aspectjrt.jar
- aspectjweaver.jar
- aspectj.jar
- aopalliance.jar

Declaring an aspect

Aspects classes are like any other normal bean and may have methods and fields just like any other class, except that they will be annotated with @Aspect as follows:

```
package org.xyz;

import org.aspectj.lang.annotation.Aspect;

@Aspect
public class AspectModule {

}
```

They will be configured in XML like any other bean as follows:

```
<bean  >
   <!-- configure properties of aspect here as normal -->
</bean>
```

Declaring a pointcut

A **pointcut** helps in determining the join points *iemethods* of interest to be executed with different advices. While working with @AspectJ based configuration, pointcut declaration has two parts:

- A pointcut expression that determines exactly which method executions we are interested in.
- A pointcut signature comprising a name and any number of parameters. The actual body of the method is irrelevant and in fact should be empty.

The following example defines a pointcut named 'businessService' that will match the execution of every method available in the classes under the package com.xyz.myapp.service:

```
import org.aspectj.lang.annotation.Pointcut;

@Pointcut("execution(* com.xyz.myapp.service.*.*(..))") // expression
private void businessService() {}  // signature
```

The following example defines a pointcut named 'getname' that will match the execution of getName method available in Student class under the package com.tutorialspoint:

```
import org.aspectj.lang.annotation.Pointcut;

@Pointcut("execution(* com.tutorialspoint.Student.getName(..))")
private void getname() {}
```

Declaring advices

You can declare any of the five advices using @{ADVICE-NAME} annotations as given below. This assumes that you already have defined a pointcut signature method businessService:

```
@Before("businessService()")
public void doBeforeTask(){
 ...
}

@After("businessService()")
public void doAfterTask(){
 ...
}

@AfterReturning(pointcut = "businessService()", returning="retVal")
public void doAfterReturnningTask(Object retVal){
  // you can intercept retVal here.
  ...
}

@AfterThrowing(pointcut = "businessService()", throwing="ex")
public void doAfterThrowingTask(Exception ex){
  // you can intercept thrown exception here.
  ...
}

@Around("businessService()")
public void doAroundTask(){
 ...
}
```

You can define you pointcut inline for any of the advices. Below is an example to define inline pointcut for before advice:

```
@Before("execution(* com.xyz.myapp.service.*.*(..))")
public doBeforeTask(){
 ...
}
```

@AspectJ Based AOP Example

To understand above mentioned concepts related to @AspectJ based AOP, let us write an example which will implement few of the advices. To write our example with few advices, let us have working Eclipse IDE in place and follow the following steps to create a Spring application:

Step	Description
1	Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project.
2	Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter.
3	Add Spring AOP specific libraries **aspectjrt.jar, aspectjweaver.jar** and **aspectj.jar** in the project.
4	Create Java classes **Logging**, *Student* and *MainApp* under the *com.tutorialspoint* package.

5	Create Beans configuration file *Beans.xml* under the **src** folder.
6	The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below.

Here is the content of **Logging.java** file. This is actually a sample of aspect module which defines methods to be called at various points.

```
package com.tutorialspoint;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Around;

@Aspect
public class Logging {

   /** Following is the definition for a pointcut to select
    *  all the methods available. So advice will be called
    *  for all the methods.
    */
   @Pointcut("execution(* com.tutorialspoint.*.*(..))")
   private void selectAll(){}

   /**
    * This is the method which I would like to execute
    * before a selected method execution.
    */
   @Before("selectAll()")
   public void beforeAdvice(){
      System.out.println("Going to setup student profile.");
   }

   /**
    * This is the method which I would like to execute
    * after a selected method execution.
    */
   @After("selectAll()")
   public void afterAdvice(){
      System.out.println("Student profile has been setup.");
   }

   /**
    * This is the method which I would like to execute
    * when any method returns.
    */
   @AfterReturning(pointcut = "selectAll()", returning="retVal")
   public void afterReturningAdvice(Object retVal){
      System.out.println("Returning:" + retVal.toString() );
   }

   /**
    * This is the method which I would like to execute
    * if there is an exception raised by any method.
    */
   @AfterThrowing(pointcut = "selectAll()", throwing = "ex")
   public void AfterThrowingAdvice(IllegalArgumentException ex){
      System.out.println("There has been an exception: " + ex.toString());
   }

}
```

Following is the content of the **Student.java** file:

```
package com.tutorialspoint;

public class Student {
   private Integer age;
   private String name;

   public void setAge(Integer age) {
      this.age = age;
   }
   public Integer getAge() {
   System.out.println("Age : " + age );
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      System.out.println("Name : " + name );
      return name;
   }
   public void printThrowException(){
      System.out.println("Exception raised");
      throw new IllegalArgumentException();
   }
}
```

Following is the content of the **MainApp.java** file:

```
package com.tutorialspoint;

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

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      Student student = (Student) context.getBean("student");

      student.getName();
      student.getAge();

      student.printThrowException();
   }
}
```

Following is the configuration file **Beans.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"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

    <aop:aspectj-autoproxy/>

   <!-- Definition for student bean -->
   <bean  >
      <property name="name"  value="Zara" />
      <property name="age"  value="11"/>
   </bean>
```

```
   <!-- Definition for logging aspect -->
   <bean  />

</beans>
```

Once you are done with creating source and bean configuration files, let us run the application. If everything is fine with your application, this will print the following message:

```
Going to setup student profile.
Name : Zara
Student profile has been setup.
Returning:Zara
Going to setup student profile.
Age : 11
Student profile has been setup.
Returning:11
Going to setup student profile.
Exception raised
Student profile has been setup.
There has been an exception: java.lang.IllegalArgumentException
.....
other exception content
```