

To use the aop namespace tags described in this section, you need to import the spring-aop schema as described below:

```
<?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 ">

   <!-- bean definition & AOP specific configuration -->

</beans>
```

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

An **aspect** is declared using the **<aop:aspect>** element, and the backing bean is referenced using the **ref** attribute as follows:

```
<aop:config>
   <aop:aspect >
   ...
   </aop:aspect>
</aop:config>

<bean  >
...
</bean>
```

Here "aBean" will be configured and dependency injected just like any other Spring bean as you have seen in previous chapters.

Declaring a pointcut

A **pointcut** helps in determining the join points *iemethods* of interest to be executed with different advices. While working with XML Schema based configuration, pointcut will be defined as follows:

```
<aop:config>
   <aop:aspect >

   <aop:pointcut
      expression="execution(* com.xyz.myapp.service.*.*(..))"/>
   ...
   </aop:aspect>
</aop:config>

<bean  >
```

```
...
</bean>
```

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

```
<aop:config>
   <aop:aspect >

   <aop:pointcut
      expression="execution(* com.tutorialspoint.Student.getName(..))"/>
   ...
   </aop:aspect>
</aop:config>

<bean  >
...
</bean>
```

Declaring advices

You can declare any of the five advices inside an <aop:aspect> using the <aop:{ADVICE NAME}> element as given below:

```
<aop:config>
   <aop:aspect >
      <aop:pointcut
         expression="execution(* com.xyz.myapp.service.*.*(..))"/>

      <!-- a before advice definition -->
      <aop:before pointcut-ref="businessService"
         method="doRequiredTask"/>

      <!-- an after advice definition -->
      <aop:after pointcut-ref="businessService"
         method="doRequiredTask"/>

      <!-- an after-returning advice definition -->
      <!--The doRequiredTask method must have parameter named retVal -->
      <aop:after-returning pointcut-ref="businessService"
         returning="retVal"
         method="doRequiredTask"/>

      <!-- an after-throwing advice definition -->
      <!--The doRequiredTask method must have parameter named ex -->
      <aop:after-throwing pointcut-ref="businessService"
         throwing="ex"
         method="doRequiredTask"/>

      <!-- an around advice definition -->
      <aop:around pointcut-ref="businessService"
         method="doRequiredTask"/>
   ...
   </aop:aspect>
</aop:config>

<bean  >
...
</bean>
```

You can use same **doRequiredTask** or different methods for different advices. These methods will be defined as a part of aspect module.

XML Schema Based AOP Example

To understand above mentioned concepts related to XML Schema 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;

public class Logging {

   /**
    * This is the method which I would like to execute
    * before a selected method execution.
    */
   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.
    */
   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.
    */
   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.
    */
   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:config>
      <aop:aspect >
         <aop:pointcut
         expression="execution(* com.tutorialspoint.*.*(..))"/>
         <aop:before pointcut-ref="selectAll" method="beforeAdvice"/>
         <aop:after pointcut-ref="selectAll" method="afterAdvice"/>
         <aop:after-returning pointcut-ref="selectAll"
                              returning="retVal"
                              method="afterReturningAdvice"/>
```

```
         <aop:after-throwing pointcut-ref="selectAll"
                             throwing="ex"
                             method="AfterThrowingAdvice"/>
      </aop:aspect>
   </aop:config>

   <!-- 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
```

Let me explain that above defined <aop:pointcut> selects all the methods defined under the package com.tutorialspoint. Let us suppose, you want to execute your advice before or after a particular method, you can define your pointcut to narrow down your execution by replacing stars * in pointcut definition with actual class and method names. Below is a modified XML configuration file to show the concept:

```
<?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:config>
   <aop:aspect >
      <aop:pointcut
      expression="execution(* com.tutorialspoint.Student.getName(..))"/>
      <aop:before pointcut-ref="selectAll" method="beforeAdvice"/>
      <aop:after pointcut-ref="selectAll" method="afterAdvice"/>
   </aop:aspect>
   </aop:config>

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

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

```
</beans>
```

If you will execute sample application with these configuration changes, this will print the following message:

```
Going to setup student profile.
Name : Zara
Student profile has been setup.
Age : 11
Exception raised
.....
other exception content
```