Spring Boot - AOP Before Advice

Last Updated : 23 Jul, 2025

Aspect-oriented programming(AOP) as the name suggests uses aspects in programming. It can be defined as the breaking of code into different modules, also known as modularisation, where the aspect is the key unit of modularity. Aspects enable the implementation of crosscutting concerns such as transaction, logging not central to business logic without cluttering the code core to its functionality. It does so by adding additional behavior that is the advice to the existing code. For example- Security is a crosscutting concern, in many methods in an application security rules can be applied, therefore repeating the code at every method, defining the functionality in a common class and control were to apply that functionality in the whole application. In this article, we will be covering a working example of **Before** Advice. Before Advice is executed before a join point execution. It is denoted by @Before annotation.

Prerequisite: Aspect Oriented Programming and AOP in Spring Framework

Steps to Implement AOP Before Advice in Spring Boot Application

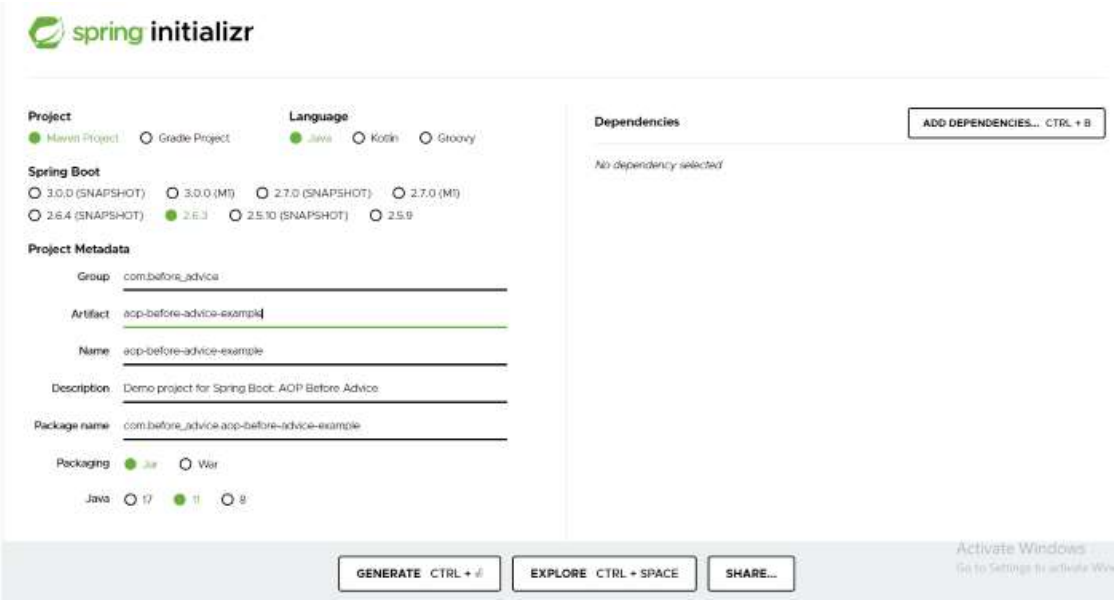
Step 1: Open **Spring Initializr** https://start.spring.io/

Step 2: Provide the **Group** name: com.beforeadvice

Step 3: Provide the **Artifact** Id: aop-before-advice-example

Step 4: Add the **Spring Web** dependency.

Step 5: Click on the **Generate** button. A zip file will be downloaded into the system. Extract it.



Step 6: Import the folder in the IDE by using the following steps

File -> Import -> Existing Maven Projects -> Next -> Browse -> Look for the folder **aop-before-advice-example** -> Finish. When the project is imported, it will install the dependencies. Once it is done, follow the next steps.

Step 7: Add the dependency for **spring AOP** in **pom.xml**

```
<project xmlns="https://maven.apache.org/POM/4.0.0"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="https://maven.apache.org/POM/4.0.0
                              https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.beforeadvice</groupId>
 <artifactId>aop-before-advice-example</artifactId>
 <version>0.0.1-SNAPSHOT</version>

 <packaging>jar</packaging>

 <name>aop-before-advice-example</name>
 <description>Demo project for Spring Boot</description>

  <parent>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-parent</artifactId>
     <version>2.2.2.RELEASE</version>
     <relativePath /> <!-- lookup parent from repository -->
  </parent>

  <properties>
     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
     <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
```

```
     <java.version>1.8</java.version>
  </properties>

    <dependencies>
       <!-- dependency for spring web -->
       <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
       </dependency>

       <!-- added dependency for spring aop -->
       <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-aop</artifactId>
        </dependency>
    </dependencies>

<build>
 <plugins>
  <plugin>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-maven-plugin</artifactId>
  </plugin>
 </plugins>
</build>

</project>
```

Save the changes and it will download the jars. Once it is done, move ahead with the next steps.

> ***Note**: If the jars are not added properly, you may get some errors.*

Step 8: Create a package with the name com.beforeadvice.model. and add a Student model class to it.

Student class:

```
package com.beforeadvice.model;

public class Student {
    private String firstName;
    private String secondName;

    public Student() {}

    public String getFirstName() { return firstName; }
```

```
    public void setFirstName(String firstName)
    {
        this.firstName = firstName;
    }

    public String getSecondName() { return secondName; }

    public void setSecondName(String secondName)
    {
        this.secondName = secondName;
    }
}
```

Step 9: Create a package with the name com.beforeadvice.service. and add a Student Service class to it. Add a method to add students with given name arguments.

StudentService class:

```
package com.beforeadvice.service;

import com.beforeadvice.model.Student;
import org.springframework.stereotype.Service;

@Service
public class StudentService {

    public Student addStudent(String fname, String sname)
    {
        System.out.println(
            "Add student service method called");
        Student stud = new Student();
        stud.setFirstName(fname);
        stud.setSecondName(sname);
        return stud;
    }
}
```

Step 10: Create a package with the name com.beforeadvice.controller. and add a Student Controller class to it. Add a method to handle Get requests and call Student Service from it.

StudentController class:

```
package com.beforeadvice.controller;

import com.beforeadvice.model.Student;
```

```
import com.beforeadvice.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class StudentController {

    @Autowired private StudentService studentService;

    @GetMapping(value = "/add")
    public Student addStudent(
        @RequestParam("firstName") String firstName,
        @RequestParam("secondName") String secondName)
    {
        return studentService.addStudent(firstName,
                                          secondName);
    }
}
```

Step 11: Create a package with the name com.beforeadvice.aspect. and add a Student Service Aspect class to it. Here we will add our Advice method and PointCut expression.

StudentServiceAspect class

```
package com.beforeadvice.aspect;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class StudentServiceAspect {

    // the pointcut expression specifying execution of any
    // method in com.beforeadvice.service.StudentService
    // class of any return type with 0 or more number of
    // arguments
    @Pointcut("execution(* com.beforeadvice.service.StudentService.*(..)) ")
    private void anyStudentService() {} // the pointcut signature

    @Before("anyStudentService() && args(fname, sname)")
    public void beforeAdvice(JoinPoint joinPoint,
                             String fname, String sname)
    {
        System.out.println(
            "Before method:" + joinPoint.getSignature()+ "\n Adding Student
with first name- "
```

```
                + fname + ", second name- " + sname);
    }
}
```

Step 12: We are done with the code structure. Now to run the application, start the application as "run as boot application". Open the browser and hit the following URL to make a get request call: http://localhost:{portNumber}/add?firstName={fname}&secondName={sname}

For a demo, we are hitting URL with fname as Harry and sname as Potter


```
{"firstName":"Harry","secondName":"Potter"}
```

As seen in the output, the advice method is called before the service method is executed. If any exception occurs in the advice method, then the service method will not be executed.