**1. Dependency Configuration**

xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</dependency>

* This **Spring Boot Starter Logging** dependency is included in the project.
* It provides **SLF4J (Simple Logging Facade for Java)** and **Logback**, which are used for logging application events.

**2. Package Declaration**

java

package com.example.demo.aspect;

* This declares that our **LoggingAspect** class belongs to the com.example.demo.aspect package.

**3. Import Statements**

java

import com.example.demo.entities.User;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.aspectj.lang.annotation.AfterReturning;

import org.aspectj.lang.annotation.Pointcut;

import org.springframework.stereotype.Component;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.util.List;

* **com.example.demo.entities.User;** → Imports the User entity to use in logging responses.
* **AOP Annotations (@Aspect, @Before, @AfterReturning, @Pointcut)**
  + These annotations belong to the **Spring AOP (Aspect-Oriented Programming)** module.
* **@Component** → Registers this class as a Spring Bean.
* **Logger (SLF4J)**
  + LoggerFactory.getLogger(LoggingAspect.class) initializes a logger to log messages.
* **List Import**
  + java.util.List; is imported to log the list of users.

**4. Declaring the Aspect Class**

java

@Aspect

@Component

public class LoggingAspect {

* **@Aspect** → Declares this class as an **Aspect**, meaning it contains cross-cutting concerns (e.g., logging).
* **@Component** → Registers it as a Spring Bean so Spring can manage it.

**5. Initializing the Logger**

java

private static final Logger logger = LoggerFactory.getLogger(LoggingAspect.class);

* This creates a **logger instance** to print log messages to the console or a log file.

**Logging for createUser Method**

**5.1 Defining a Pointcut**

java

@Pointcut("execution(\* com.example.demo.Service.UserService.createUser(..))")

public void createUserMethod() {}

* Defines a **pointcut** for the method createUser(..) inside UserService.
* The execution(..) expression means:
  + \* → Any return type.
  + com.example.demo.Service.UserService.createUser(..) → Matches the method in UserService.
  + (..) → Matches any number of parameters.

**5.2 Before Advice**

java

@Before("createUserMethod()")

public void logBeforeCreateUser() {

logger.info("Creating a new user...");

}

* Runs **before** createUser method executes.
* Logs "Creating a new user..." message.

**5.3 After Returning Advice**

java

@AfterReturning(pointcut = "createUserMethod()", returning = "result")

public void logAfterCreateUser(Object result) {

logger.info("Successfully created user with ID: {}", ((User) result).getId());

}

* Runs **after** createUser executes **successfully**.
* Logs "Successfully created user with ID: {userId}".

**Logging for getUserById Method**

**6.1 Defining a Pointcut**

java

@Pointcut("execution(\* com.example.demo.Service.UserService.getUserById(..))")

public void getUserByIdMethod() {}

* Matches getUserById(..) method in UserService.

**6.2 Before Advice**

java

@Before("getUserByIdMethod()")

public void logBeforeGetUserById() {

logger.info("Fetching user by ID...");

}

* Logs "Fetching user by ID..." **before** execution.

**6.3 After Returning Advice**

java

@AfterReturning(pointcut = "getUserByIdMethod()", returning = "result")

public void logAfterGetUserById(Object result) {

if (result != null) {

logger.info("Successfully fetched user with ID: {}", ((User) result).getId());

} else {

logger.info("No user found with the given ID.");

}

}

* Logs "Successfully fetched user with ID: {userId}" if a user is found.
* If null, logs "No user found with the given ID.".

**Logging for getAllUsers Method**

**7.1 Defining a Pointcut**

java

@Pointcut("execution(\* com.example.demo.Service.UserService.getAllUsers(..))")

public void getAllUsersMethod() {}

* Matches getAllUsers() method.

**7.2 Before Advice**

java

@Before("getAllUsersMethod()")

public void logBeforeGetAllUsers() {

logger.info("Fetching all users...");

}

* Logs "Fetching all users..." **before** execution.

**7.3 After Returning Advice**

java

@AfterReturning(pointcut = "getAllUsersMethod()", returning = "result")

public void logAfterGetAllUsers(Object result) {

logger.info("Successfully fetched all users, count: {}", ((List<?>) result).size());

}

* Logs "Successfully fetched all users, count: {size}".

**Logging for updateUser Method**

**8.1 Defining a Pointcut**

java

@Pointcut("execution(\* com.example.demo.Service.UserService.updateUser(..))")

public void updateUserMethod() {}

* Matches updateUser(..) method.

**8.2 Before Advice**

java

@Before("updateUserMethod()")

public void logBeforeUpdateUser() {

logger.info("Updating user...");

}

* Logs "Updating user..." **before** execution.

**8.3 After Returning Advice**

java

@AfterReturning(pointcut = "updateUserMethod()", returning = "result")

public void logAfterUpdateUser(Object result) {

logger.info("Successfully updated user with ID: {}", ((User) result).getId());

}

* Logs "Successfully updated user with ID: {userId}".

**Logging for deleteUser Method**

**9.1 Defining a Pointcut**

java

@Pointcut("execution(\* com.example.demo.Service.UserService.deleteUser(..))")

public void deleteUserMethod() {}

* Matches deleteUser(..) method.

**9.2 Before Advice**

java

@Before("deleteUserMethod()")

public void logBeforeDeleteUser() {

logger.info("Deleting user...");

}

* Logs "Deleting user..." **before** execution.

**9.3 After Returning Advice**

java

@AfterReturning(pointcut = "deleteUserMethod()")

public void logAfterDeleteUser() {

logger.info("Successfully deleted a user.");

}

* Logs "Successfully deleted a user.".

**Summary**

| **Aspect Feature** | **Description** |
| --- | --- |
| @Aspect | Declares the class as an aspect for cross-cutting concerns. |
| @Component | Registers it as a Spring Bean. |
| @Pointcut | Defines where logging should apply. |
| @Before | Logs messages before method execution. |
| @AfterReturning | Logs messages after successful execution. |
| LoggerFactory.getLogger(..) | Used for logging messages. |