Spring AOP example

spring AOP example from scratch

Here's a **complete Spring AOP example from scratch**, using **Spring Boot** and **AspectJ** for method-level logging. The example covers:

- Maven dependencies
- A simple service with a method
- An aspect to log before and after method execution
- Full project structure

✅ 1. Create a Spring Boot project

You can use Spring Initializr with:

- Project: Maven
- Dependencies:
  - Spring Web
  - Spring AOP

✅ 2. Maven `pom.xml`

xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.example</groupId>
 <artifactId>spring-aop-demo</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <name>spring-aop-demo</name>
 <description>Demo project for Spring AOP</description>

 <parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.1.5</version>
```

```
  </parent>

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

    <!-- Spring AOP -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-aop</artifactId>
    </dependency>

    <!-- Optional for logging -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-logging</artifactId>
    </dependency>

    <!-- Test -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

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

✅ 3. Directory Structure

```
src
└── main
    └── java
        └── com
            └── example
                └── aopdemo
                    ├── SpringAopDemoApplication.java
                    ├── service
                    │   └── GreetingService.java
                    └── aspect
                        └── LoggingAspect.java
```

✅ 4. Main Application

```java
package com.example.aopdemo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringAopDemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringAopDemoApplication.class, args);
    }
}
```

✅ 5. Service Class

```java
package com.example.aopdemo.service;

import org.springframework.stereotype.Service;
```

```java
@Service
public class GreetingService {
    public String greet(String name) {
        return "Hello, " + name;
    }
}
```

✅ 6. Aspect Class

java

```java
package com.example.aopdemo.aspect;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.*;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LoggingAspect {

    @Before("execution(* com.example.aopdemo.service.*.*(..))")
    public void beforeMethod(JoinPoint joinPoint) {
        System.out.println("➡ [Before] Method: " + joinPoint.getSignature().getName());
    }

    @After("execution(* com.example.aopdemo.service.*.*(..))")
    public void afterMethod(JoinPoint joinPoint) {
        System.out.println("⬅ [After] Method: " + joinPoint.getSignature().getName());
    }

    @AfterReturning(pointcut = "execution(* com.example.aopdemo.service.*.*(..))", returning
= "result")
    public void afterReturning(JoinPoint joinPoint, Object result) {
        System.out.println("✅ [Return] Method: " + joinPoint.getSignature().getName() + "
returned: " + result);
    }

    @AfterThrowing(pointcut = "execution(* com.example.aopdemo.service.*.*(..))", throwing
= "ex")
    public void afterThrowing(JoinPoint joinPoint, Throwable ex) {
```

```
        System.out.println("❌ [Exception] Method: " + joinPoint.getSignature().getName() + "
threw: " + ex.getMessage());
    }
}
```

✅ 7. Create a Controller (optional)

java

```
package com.example.aopdemo;

import com.example.aopdemo.service.GreetingService;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api")
public class GreetingController {

    private final GreetingService greetingService;

    public GreetingController(GreetingService greetingService) {
        this.greetingService = greetingService;
    }

    @GetMapping("/greet/{name}")
    public String greet(@PathVariable String name) {
        return greetingService.greet(name);
    }
}
```

✅ 8. Run & Test

Start the app:

bash

```
mvn spring-boot:run
```

Then call:

bash

http://localhost:8080/api/greet/Viswanath

You should see the AOP logs in the console:

mathematica

➡ [Before] Method: greet
⬅ [After] Method: greet
✅ [Return] Method: greet returned: Hello, Viswanath

✅ Summary of AOP Annotations

Annotation	Description
`@Aspect`	Declares the class as an aspect
`@Before`	Runs before the method
`@After`	Runs after the method (even if exception)
`@AfterReturning`	Runs if method returns normally
`@AfterThrowing`	Runs if method throws exception
`@Around`	Wraps method completely (advanced usage)

Would you like an example using `@Around` for execution time as well?

here @GetMapping("/greet/{name}")
public String greet(@PathVariable String name) {
return greetingService.greet(name);
}

where or how LoggingAspect calling

Great question! Let's clarify how Spring AOP (Aspect-Oriented Programming) actually **invokes the** `LoggingAspect` without explicitly calling it in your controller or service.

🧠 How `LoggingAspect` Is Triggered

You wrote this in your controller:

```java
@GetMapping("/greet/{name}")
public String greet(@PathVariable String name) {
    return greetingService.greet(name);
}
```

You're directly calling:

```java
greetingService.greet(name);
```

And you might wonder: **"How is `LoggingAspect` getting triggered automatically?"**

✅ Answer: Spring uses dynamic proxies behind the scenes

Here's what's happening under the hood:

1. **At runtime**, Spring scans your `@Aspect` annotated class (`LoggingAspect`) because it's marked as a Spring `@Component`.
2. When it sees this line in your aspect:

   ```java
   @Before("execution(* com.example.aopdemo.service.*.*(..))")
   ```

   It knows: "I need to intercept all methods in `com.example.aopdemo.service`."

3. Spring **wraps** your `GreetingService` with a **proxy object** using **JDK or CGLIB proxying**.
4. When you call `greetingService.greet(name)` in your controller, you're **not calling the real object directly** — you're calling the **Spring-generated proxy**, which:
   - Intercepts the method call
   - Passes it to the aspect (`LoggingAspect`)
   - Then forwards it to the actual method (`greet`)
   - Then applies `@After`, `@AfterReturning`, etc.

🔁 Illustration of the flow

```
text

Client (Browser) ---> Controller
                        |
                        V
     greetingService.greet("Viswanath")  [This is a proxy call]
                        |
    -------------------------------
    |   AOP Intercepts via Aspect |
    |   LoggingAspect triggers    |
    -------------------------------
                        |
                        V
      Real greet() method executes
```

✅ So to sum up:

- You **do not call** `LoggingAspect` **explicitly**.
- Spring **injects proxies** for beans and **calls your aspect methods** based on matching pointcuts.

🧪 Want to prove it?

Add a print inside your `GreetingService`:

```java
public String greet(String name) {
    System.out.println("🔧 Inside greet()");
    return "Hello, " + name;
}
```

Then observe console logs:

```text
➡️ [Before] Method: greet
🔧 Inside greet()
⬅️ [After] Method: greet
✅ [Return] Method: greet returned: Hello, Viswanath
```

This confirms: **Aspect logic runs automatically before and after your method** without manual invocation.

Would you like me to show an example using `@Around` to log execution time as well?

exaplin eactch every point Key Concepts of Spring AOP:

Aspect:

Advice:

Join Point:

Pointcut:

Target Object:

Weaving:

above explain with code

Absolutely! Let's break down **key Spring AOP concepts** with clear explanations and code examples so you can deeply understand each part:

🧠 1. Aspect

🔹 Definition:

An **aspect** is a module/class that encapsulates cross-cutting concerns like logging, security, transaction, etc.

> Think of it as a class that contains **advices** (actions to take) and **pointcuts** (when to take them).

✅ Code:

```java
@Aspect
@Component
public class LoggingAspect {
    @Before("execution(* com.example.service.*.*(..))")
    public void logBeforeMethod() {
        System.out.println("📍 Logging BEFORE method execution.");
    }
}
```

🧠 2. Advice

🔹 Definition:

An **advice** is the actual action taken **at a join point** (e.g., log, validate, secure).

> Advice types: `@Before`, `@After`, `@AfterReturning`, `@AfterThrowing`, `@Around`

✅ Code:

```java
// BEFORE advice
@Before("execution(* com.example.service.*.*(..))")
public void logBefore() {
    System.out.println("💡 This is a BEFORE advice");
}

// AFTER advice
```

```
@After("execution(* com.example.service.*.*(..))")
public void logAfter() {
    System.out.println("💡 This is an AFTER advice");
}
```

Each of these methods is an **advice** – they run at specific points in method execution.

🧠 3. Join Point

🔹 Definition:

A **join point** is a point in program execution where an advice can be applied.

> In Spring AOP, join points are **only method executions** (not field access or constructor calls).

✅ Code:

```
java

@Before("execution(* com.example.service.GreetingService.greet(..))")
public void log(JoinPoint joinPoint) {
    System.out.println("🎯 JoinPoint Signature: " + joinPoint.getSignature().getName());
}
```

This advice executes at the **join point**: the execution of `greet()` method in `GreetingService`.

🧠 4. Pointcut

🔹 Definition:

A **pointcut** is an expression that matches **join points** and tells Spring **when to apply advices**.

> You define the **conditions** under which advices run.

✅ Code:

```
java
```

```
@Pointcut("execution(* com.example.service.GreetingService.*(..))")
public void allGreetingMethods() {}

@Before("allGreetingMethods()")
public void beforeAdvice() {
    System.out.println("📌 Pointcut matched. Running before advice.");
}
```

This pointcut targets **all methods** in `GreetingService`.

🧠 5. Target Object

🔹 Definition:

The **target object** is the **actual object** whose method is being advised (proxied).

> When you call `greetingService.greet()`, Spring injects a **proxy** of the real object and applies advices.

✅ Example:

```
java

@RestController
public class GreetingController {
    @Autowired
    private GreetingService greetingService;

    @GetMapping("/greet/{name}")
    public String greet(@PathVariable String name) {
        return greetingService.greet(name); // Target object is GreetingService
    }
}
```

Here, `greetingService` is the **target object** (proxied by Spring AOP).

🧠 6. Weaving

🔹 Definition:

Weaving is the process of **linking aspects with the target object** at the appropriate join points.

> In Spring, weaving is done at **runtime** using **proxy-based AOP** (JDK dynamic proxy or CGLIB).

✅ Example:

No code needed — Spring Boot + `@Aspect` + `@Component` + AOP starter does the weaving automatically at runtime.

💡 Summary Table:

Concept	Description	Code Snippet/Example
Aspect	Module with cross-cutting logic	`@Aspect`, `@Component` class with advices
Advice	Action taken at join point	`@Before`, `@After`, etc. methods
Join Point	Point in app where advice is applied (method exec in Spring)	`JoinPoint joinPoint` in advice
Pointcut	Expression that matches join points	`@Pointcut("execution(..)")` and use in advice
Target	Real bean whose method is being advised	`GreetingService` bean injected and proxied by Spring
Weaving	Linking aspect with app objects	Happens automatically at runtime in Spring using proxies

---

Would you like a **diagram** to visualize these concepts or want to see an **@Around advice** for timing methods?