AOP Interview Questions and Answers for Experienced Developers

What is AOP and why is it used?

AOP (Aspect-Oriented Programming) is a programming paradigm that separates cross-cutting concerns (like logging, security, transactions) from the main business logic. It allows better modularization.

Example: Instead of adding logging in every method, define one logging aspect.

What is a cross-cutting concern?

A functionality that spans multiple points in an application e.g., logging, security, exception handling, and transactions.

What are the main components of AOP?

- Aspect: Module with cross-cutting concerns.
- Join Point: Point during execution (e.g., method call).
- Advice: Code to execute at join point.
- Pointcut: Expression to select join points.
- Weaving: Linking aspects with other code.

Difference between Spring AOP and AspectJ?

Spring AOP uses proxies and supports method execution join points only. AspectJ supports field access, constructors, and uses compile-time weaving. Spring AOP is runtime and limited in join points.

Types of advice in Spring AOP?

- Before: Runs before join point.
- After returning: After successful completion.
- After throwing: On exception.
- After (finally): After method (success or failure).
- Around: Wraps method execution; allows control over execution.

What is a Join Point and Pointcut in Spring AOP?

Join Point: Where advice can be applied (only method execution in Spring AOP).
Pointcut: Expression to filter join points, e.g., execution(* com.example.service.*.*(..))

What is weaving in AOP?

Weaving is the process of linking aspects with target objects to create an advised object. In Spring AOP, weaving is done at runtime using proxies.

How is Spring AOP implemented internally?

Spring AOP uses dynamic proxies: JDK proxies if the target implements an interface; CGLIB proxies if its a class.

How to create an aspect in Spring Boot?

```
@Aspect
@Component
public class LoggingAspect {
  @Before("execution(* com.example.service.*.*(..))")
  public void logBefore(JoinPoint joinPoint) {
    System.out.println("Before method: " + joinPoint.getSignature());
  }
}
```

What is the order of execution for multiple aspects?

Use @Order annotation to define precedence. Lower values have higher precedence.

Can you apply AOP to private methods in Spring?

No, Spring AOP only intercepts public methods in Spring-managed beans.

Can AOP be applied to fields or constructors?

Not in Spring AOP only AspectJ supports this with compile-time weaving.

How would you handle logging across services using AOP?

Use a common aspect with @Before and @AfterReturning on service layer packages.

How do you exclude methods or classes from being advised?

Using pointcut expressions with exclusions, e.g., !execution(* com.example.util..*(..))

How do you access method arguments and return values in advice?

Use ProceedingJoinPoint in @Around advice to capture args and result.

Cross Questions

- Why can't Spring AOP handle field-level join points?
Because it uses proxies and only supports method execution.
- What is the performance impact of AOP?
Minimal, but overuse can cause bottlenecks.
- Can AOP be used in exception handling?
Yes, use @AfterThrowing.
- What happens if an aspect throws an exception?
It propagates unless caught in advice.
- How can we test AOP in unit tests?
Use @SpringBootTest or log assertions.