Spring AOP

Stands for Aspect-Oriented Programming

"The modularization of cross-cutting concerns" – functionality required through multiple layers of the application, and loosely coupling from other code/injecting where necessary.

In OOP, the unit of modularity is objects; in AOP, it's aspects. Aspects are classes decorated with the @Aspect annotation.

Common uses of AOP: logging, security, caching, etc…

@Aspect is from AspectJ; Spring supports integration with this as it is an AOP framework.

Join-Point – point in application execution at which advice (injected code) is applied. Usually method execution.

Point-Cup – set of join-points that a particular advice should be applied at.

Advice is associated with point cuts & run at matching join-points.

Advice – Code to inject (methods in aspect)

@Before – Executes before the JP.

@After – Executes after the JP, regardless if an exception was thrown or not.

@AfterThrowing – Executes after the JP, only if an exception was thrown.

@AfterReturning - Executes after the JP, only if no exception(s) were thrown.

@Around – Executes both before and after the JP. This is the most powerful advice, as it exposes the ProceedingJoinPoint aka the currently executing JP.

Target – The object being advised. This uses a Spring AOP proxy object, set up by framework to intercept calls to target & delegate appropriate advice.

Point-cut expressions

Specifies where to apply the advice.

Example: `execution(* com.revature.*.*(..))`

"exection" is the point-cut type (there are 5, this is the most common)

The first asterisk is the return type (any in this case)

Next is the package, and method name(s); all methods in com.revature in this case

Finally is the parameter list in the enclosed parenthesis