Spring AOP Interview Questions

In this post we will look at Spring AOP Interview questions. Examples are provided with explanation.   
  
  
**Q: What is Spring AOP? What is its use  
A:**Suppose we want to log every method entry and exit. This can be achieved by writing log statements in every method at the start and end. But this will require lot of code work. There are various such tasks like Security which need to be applied across all methods or classes. These are known as cross cutting concerns. AOP addresses the problem of cross-cutting concerns, which would be any kind of code that is repeated in different methods and cannot normally be completely refactored into its own module, like with logging or verification.   
  
**Q: What are the different implementations of Spring AOP?  
A:**The different implementations of Spring AOP are-

* AspectJ
* Spring AOP
* JBoss AOP

**Q: Explain different AOP terminologies??  
A:**The different AOP terminologies are

* **Joinpoint:** A Joinpoint is a candidate point in the Program Execution of the application where an aspect can be plugged in. This point could be a method being called, an exception being thrown, or even a field being modified. These are the points where your aspect€™s code can be inserted into the normal flow of your application to add new behavior.
* **Advice:** This is an object which includes API invocations to the system wide concerns representing the action to perform at a Joinpoint specified by a point.
* Pointcut: A Pointcut defines at what Joinpoint, the associated Advice should be applied. Advice can be applied at any Joinpoint supported by the AOP framework. Of course, you done€™t want to apply all of your aspects at all of the possible Joinpoint. Pointcut allow you to specify where you want your advice to be applied. Often you specify these Pointcut using explicit class and method names or through regular expressions that define matching class and method name patterns. Some AOP frameworks allow you to create dynamic Pointcut that determine whether to apply advice based on runtime decisions, such as the value of method parameters.
* **Aspect:** The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Aspects enable the modularization of concerns such as transaction management that cut across multiple types and objects.
* **Weaving:** In Spring AOP makes it possible to modularize and separate logging, transaction like services and apply them declaratively to the components Hence programmer can focus on specific concerns. Aspects are wired into objects in the spring XML file in the way as JavaBean. This process is known as 'Weaving'.

**Q: What are the different types of Spring Advice?  
A:**The different types of Spring Advice are-

* **Before advice:** Advice that executes before a join point.
* **After returning advice:** Advice to be executed after a join point completes normally.
* **After throwing advice:** Advice to be executed if a method exits by throwing an exception.
* **After advice:** Advice to be executed regardless of the means by which a join point exits.
* **Around advice:** Advice that surrounds a join point such as a method invocation.

**Q: Define Run-time AOP vs Compile-time AOP?  
A:**The different types of AOPs depending on when they are loaded are-

* **Source code weaving:** Aspect code is injected as source code statements into your application source code. This is some kind of preprocessor approach. No AOP framework in the Java world uses this approach nowadays, but there used to be some in the early days of AOP.
* **Compile-time weaving:** Aspect code is woven into your application by a special compiler.
* **Binary weaving:** Aspect code is woven into existing class files after compilation rather than during compilation.
* **Load-time weaving (LTW):** A weaving agent/library is loaded early when your VM/container is started. It gets a configuration file with rules describing which aspects should be woven into which classes.
* **Proxy-based LTW:** This special LTW form is used by Spring AOP while AspectJ does the previous 3 forms listed above. It works by creating dynamic proxies (i.e. subclasses or interface implementations) for aspect targets.