IoC (DI)

1. What is IOC or inversion of control?  
     
   As the name implies Inversion of control means now we have inverted the control of creating the object from our own using new operator to container or framework. Now it’s the responsibility of container to create object as required. We maintain one xml file where we configure our components, services, all the classes and their property. We just need to mention which service is needed by which component and container will create the object for us. This concept is known as dependency injection because all object dependency (resources) is injected into it by framework.  
     
   Example:  
    <bean id="createNewStock" class="springexample.stockMarket.CreateNewStockAccont">  
    <property name="newBid"/>  
    </bean>  
     
   In this example CreateNewStockAccont class contain getter and setter for newBid and container will instantiate newBid and set the value automatically when it is used. This whole process is also called wiring in Spring and by using annotation it can be done automatically by Spring, refereed as auto-wiring of bean in Spring.
2. Explain Bean-LifeCycle (Difficult question).  
     
   Spring framework is based on IOC so we call it as IOC container also So Spring beans reside inside the IOC container. Spring beans are nothing but Plain old java object (POJO).  
     
   Following steps explain their life cycle inside container.  
   1. Container will look the bean definition inside configuration file (e.g. bean.xml).  
   2 using reflection container will create the object and if any property is defined inside the bean definition then it will also be set.  
   3. If the bean implements the BeanNameAware interface, the factory calls setBeanName() passing the bean’s ID.  
   4. If the bean implements the BeanFactoryAware interface, the factory calls setBeanFactory(), passing an instance of itself.  
   5. If there are any BeanPostProcessors associated with the bean, their post- ProcessBeforeInitialization() methods will be called before the properties for the Bean are set.  
   6. If an init() method is specified for the bean, it will be called.  
   7. If the Bean class implements the DisposableBean interface, then the method destroy() will be called when the Application no longer needs the bean reference.  
   8. If the Bean definition in the Configuration file contains a 'destroy-method' attribute, then the corresponding method definition in the Bean class will be called.
3. What is difference between singleton and prototype bean?  
     
   Ans: This is another popular spring interview questions and an important concept to understand. Basically a bean has scopes which defines their existence on the application  
   Singleton: means single bean definition to a single object instance per Spring IOC container.  
   Prototype: means a single bean definition to any number of object instances.  
   Whatever beans we defined in spring framework are singleton beans. There is an attribute in bean tag named ‘singleton’ if specified true then bean becomes singleton and if set to false then the bean becomes a prototype bean. By default it is set to true. So, all the beans in spring framework are by default singleton beans.  
     
    <bean id="createNewStock" class="springexample.stockMarket.CreateNewStockAccont" singleton=”false”>  
    <property name="newBid"/>  
    </bean>
4. What are the different types of bean injections?  
   There are two types of bean injections.  
   1. By setter  
   2. By constructor

AOP

1. What is AOP?  
     
   Answer : The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules. AOP is a programming technique that allows developer to modularize crosscutting concerns, that cuts across the typical divisions of responsibility, such as logging and transaction management. Spring AOP, aspects are implemented using regular classes or regular classes annotated with the @Aspect annotation
2. Example Use of AOP?  
   1. Logging  
   2. Transaction
3. How many advice Spring support?  
   Spring aspects can work with five kinds of advice:   
   *Before*—The advice functionality takes place before the advised method is invoked.   
   *After*—The advice functionality takes place after the advised method completes, regardless of the outcome.   
   *After-returning*—The advice functionality takes place after the advised method successfully completes.   
   *After-throwing*—The advice functionality takes place after the advised method throws an exception.   
   *Around*—The advice wraps the advised method, providing some functionality before and after the advised method is invoked.
4. What kind of joint point Spring support?  
   SPRING ONLY SUPPORTS METHOD JOIN POINTS, not support field and constructor join points.