SPRING INTERVIEW QUESTIONS

Dear readers, these **Spring Interview Questions** have been designed specially to get you acquainted with the nature of questions you may encounter during your interview for the subject of **Spring**. As per my experience good interviewers hardly plan to ask any particular question during your interview, normally questions start with some basic concept of the subject and later they continue based on further discussion and what you answer:

Q: What is Spring?

A: Spring is an open source development framework for enterprise Java. The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make J2EE development easier to use and promote good programming practice by enabling a POJO-based programming model.

Q: what are benefits of using spring?

A: Following is the list of few of the great benefits of using Spring Framework:

- **Lightweight:** Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 2MB.
- **Inversion of control (IOC):** Loose coupling is achieved in spring using the technique Inversion of Control. The objects give their dependencies instead of creating or looking for dependent objects.
- **Aspect oriented (AOP):** Spring supports Aspect oriented programming and enables cohesive development by separating application business logic from system services.
- **Container:** Spring contains and manages the life cycle and configuration of application objects.
- **MVC Framework:** Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over engineered or less popular web frameworks.
- **Transaction Management:** Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).
- **Exception Handling:** Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.

Q: What are the different modules in Spring framework?

A: Following are the modules of the Spring framework:

- Core module
- Bean module
- Context module
- Expression Language module
- JDBC module
- ORM module
- OXM module
- Java Messaging Service(JMS) module
- Transaction module

- Web module
- Web-Servlet module
- Web-Struts module
- Web-Portlet module

Q: What is Spring configuration file?

A: Spring configuration file is an XML file. This file contains the classes information and describes how these classes are configured and introduced to each other.

Q: What is Dependency Injection?

A: Inversion of Control (IoC) is a general concept, and it can be expressed in many different ways and Dependency Injection is merely one concrete example of Inversion of Control.

This concept says that you do not create your objects but describe how they should be created. You don't directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A container (the IOC container) is then responsible for hooking it all up.

Q: What are the different types of IoC (dependency injection)?

A: Types of IoC are:

- **Constructor-based dependency injection:** Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.
- **Setter-based dependency injection:** Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

Q: Which DI would you suggest Constructor-based or setter-based DI?

A: Since you can mix both, Constructor- and Setter-based DI, it is a good rule of thumb to use constructor arguments for mandatory dependencies and setters for optional dependencies. Note that the use of a *@Required* annotation on a setter can be used to make setters required dependencies.

Q: What are the benefits of IOC?

A: The main benefits of IOC or dependency injection are:

- It minimizes the amount of code in your application.
- It makes your application easy to test as it doesn't require any singletons or JNDI lookup mechanisms in your unit test cases.
- Loose coupling is promoted with minimal effort and least intrusive mechanism.
- IOC containers support eager instantiation and lazy loading of services.

Q: What is AOP?

A: Aspect-oriented programming, or AOP, is a programming technique that allows programmers to modularize crosscutting concerns, or behavior that cuts across the typical divisions of responsibility, such as logging and transaction management. The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules.

Q: What is Spring IoC container?

A: The Spring IoC creates the objects, wire them together, configure them, and manage their complete lifecycle from creation till destruction. The Spring container uses dependency injection (DI) to manage the components that make up an application.

Q: What are types of IoC containers? Explain them.

A: There are two types of IoC containers:

- **Bean Factory container:** This is the simplest container providing basic support for DI .The BeanFactory is usually preferred where the resources are limited like mobile devices or applet based applications
- **Spring ApplicationContext Container:** This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners.

Q: Give an example of BeanFactory implementation.

A: The most commonly used BeanFactory implementation is the **XmlBeanFactory** class. This container reads the configuration metadata from an XML file and uses it to create a fully configured system or application.

Q: What are the common implementations of the ApplicationContext?

A: The three commonly used implementation of 'Application Context' are:

- **FileSystemXmlApplicationContext:** This container loads the definitions of the beans from an XML file. Here you need to provide the full path of the XML bean configuration file to the constructor.
- **ClassPathXmlApplicationContext:** This container loads the definitions of the beans from an XML file. Here you do not need to provide the full path of the XML file but you need to set CLASSPATH properly because this container will look bean configuration XML file in CLASSPATH.
- **WebXmlApplicationContext:** This container loads the XML file with definitions of all beans from within a web application.

Q: What is the difference between Bean Factory and ApplicationContext?

A: Following are some of the differences:

- Application contexts provide a means for resolving text messages, including support for i18n of those messages.
- Application contexts provide a generic way to load file resources, such as images.
- Application contexts can publish events to beans that are registered as listeners.
- Certain operations on the container or beans in the container, which have to be handled in a programmatic fashion with a bean factory, can be handled declaratively in an application context.
- The application context implements MessageSource, an interface used to obtain localized messages, with the actual implementation being pluggable.

Q: What are Spring beans?

A: The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container, for example, in the form of XML <bean/> definitions.

Q: What does a bean definition contain?

A: The bean definition contains the information called configuration metadata which is needed for the container to know the followings:

- How to create a bean
- Bean's lifecycle details
- Bean's dependencies

Q: How do you provide configuration metadata to the Spring Container?

A: There are following three important methods to provide configuration metadata to the Spring Container:

- XML based configuration file.
- Annotation-based configuration
- Java-based configuration

Q: How do add a bean in spring application?

A: Check the following example:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <bean  >
       <property name="message" value="Hello World!"/>
   </bean>

</beans>
```

Q: How do you define a bean scope?

A: When defining a <bean> in Spring, you have the option of declaring a scope for that bean. For example, to force Spring to produce a new bean instance each time one is needed, you should declare the bean's scope attribute to be **prototype**. Similar way if you want Spring to return the same bean instance each time one is needed, you should declare the bean's scope attribute to be **singleton.**

Q: What bean scopes does Spring support? Explain them.

A: The Spring Framework supports following five scopes, three of which are available only if you use a web-aware ApplicationContext.

- **singleton:** This scopes the bean definition to a single instance per Spring IoC container.
- **prototype:** This scopes a single bean definition to have any number of object instances.
- **request:** This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
- **session:** This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
- **global-session:** This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Q: What is default scope of bean in Spring framework?

A: The default scope of bean is Singleton for Spring framework.

Q: Are Singleton beans thread safe in Spring Framework?

A: No, singleton beans are not thread-safe in Spring framework.

Q: Explain Bean lifecycle in Spring framework?

A: Following is sequence of a bean lifecycle in Spring:

- **Instantiate** - First the spring container finds the bean's definition from the XML file and instantiates the bean..
- **Populate properties** - Using the dependency injection, spring populates all of the properties as specified in the bean definition..

- **Set Bean Name** - If the bean implements BeanNameAware interface, spring passes the bean's id to setBeanName() method.
- **Set Bean factory** - If Bean implements BeanFactoryAware interface, spring passes the beanfactory to setBeanFactory() method.
- **Pre Initialization** - Also called postprocess of bean. If there are any bean BeanPostProcessors associated with the bean, Spring calls postProcesserBeforeInitialization() method.
- **Initialize beans** - If the bean implements Intializing Bean,its afterPropertySet() method is called. If the bean has init method declaration, the specified initialization method is called.
- **Post Initialization** - If there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization() methods will be called.
- **Ready to use** - Now the bean is ready to use by the application.
- **Destroy** - If the bean implements DisposableBean , it will call the destroy() method .

Q: What are inner beans in Spring?

A: A <bean/> element inside the <property/> or <constructor-arg /> elements defines a so-called inner bean. An inner bean definition does not require a defined id or name; the container ignores these values. It also ignores the scope flag. Inner beans are always anonymous and they are always scoped as prototypes.

Q: How can you inject Java Collection in Spring?

A: Spring offers four types of collection configuration elements which are as follows:

- **<list>**: This helps in wiring i.e. injecting a list of values, allowing duplicates.
- **<set>**: This helps in wiring a set of values but without any duplicates.
- **<map>**: This can be used to inject a collection of name-value pairs where name and value can be of any type.
- **<props>**: This can be used to inject a collection of name-value pairs where the name and value are both Strings.

Q: What is bean auto wiring?

A: The Spring container is able to autowire relationships between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for your bean by inspecting the contents of the BeanFactory without using <constructor-arg > and <property> elements.

Q: What are different Modes of auto wiring?

A: The autowiring functionality has five modes which can be used to instruct Spring container to use autowiring for dependency injection:

- **no**: This is default setting which means no autowiring and you should use explicit bean reference for wiring. You have nothing to do special for this wiring. This is what you already have seen in Dependency Injection chapter.
- **byName**: Autowiring by property name. Spring container looks at the properties of the beans on which autowire attribute is set to byName in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file.
- **byType**: Autowiring by property datatype. Spring container looks at the properties of the beans on which autowire attribute is set to byType in the XML configuration file. It then tries to match and wire a property if its type matches with exactly one of the beans name in configuration file. If more than one such beans exist, a fatal exception is thrown.
- **constructor**: Similar to byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised.

- **autodetect**: Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by byType.

Q: What are the limitations with autowiring?

A: Limitations of autowiring are:

- **Overriding possibility**: You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring.
- **Primitive data types**: You cannot autowire so-called simple properties such as primitives, Strings, and Classes.
- **Confusing nature**: Autowiring is less exact than explicit wiring, so if possible prefer using explicit wiring.

Q: Can you inject null and empty string values in Spring?

A: Yes.

Q: What is Annotation-based container configuration?

A: An alternative to XML setups is provided by annotation-based configuration which relies on the bytecode metadata for wiring up components instead of angle-bracket declarations. Instead of using XML to describe a bean wiring, the developer moves the configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

Q: How do you turn on annotation wiring?

A: Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file by configuring <context:annotation-config/>.

Q: What does @Required annotation mean?

A: This annotation simply indicates that the affected bean property must be populated at configuration time, through an explicit property value in a bean definition or through autowiring. The container throws BeanInitializationException if the affected bean property has not been populated.

Q: What does @Autowired annotation mean?

A: This annotation provides more fine-grained control over where and how autowiring should be accomplished. The @Autowired annotation can be used to autowire bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

Q: What does @Qualifier annotation mean?

A: There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property, in such case you can use @Qualifier annotation along with @Autowired to remove the confusion by specifying which exact bean will be wired.

Q: What are the JSR-250 Annotations? Explain them.

A: Spring has JSR-250 based annotations which include @PostConstruct, @PreDestroy and @Resource annotations.

- **@PostConstruct**: This annotation can be used as an alternate of initialization callback.
- **@PreDestroy**: This annotation can be used as an alternate of destruction callback.
- **@Resource** : This annotation can be used on fields or setter methods. The @Resource annotation takes a 'name' attribute which will be interpreted as the bean name to be injected. You can say, it follows by-name autowiring semantics.

Q: What is Spring Java Based Configuration? Give some annotation example.

A: Java based configuration option enables you to write most of your Spring configuration without XML but with

the help of few Java-based annotations.

For example: Annotation **@Configuration** indicates that the class can be used by the Spring IoC container as a source of bean definitions. The **@Bean** annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context.

Q: How is event handling done in Spring?

A: Event handling in the *ApplicationContext* is provided through the *ApplicationEvent* class and *ApplicationListener* interface. So if a bean implements the *ApplicationListener*, then every time an *ApplicationEvent* gets published to the ApplicationContext, that bean is notified.

Q: Describe some of the standard Spring events.

A: Spring provides the following standard events:

- **ContextRefreshedEvent**: This event is published when the ApplicationContext is either initialized or refreshed. This can also be raised using the refresh() method on the ConfigurableApplicationContext interface.
- **ContextStartedEvent**: This event is published when the ApplicationContext is started using the start() method on the ConfigurableApplicationContext interface. You can poll your database or you can re/start any stopped application after receiving this event.
- **ContextStoppedEvent**: This event is published when the ApplicationContext is stopped using the stop() method on the ConfigurableApplicationContext interface. You can do required housekeep work after receiving this event.
- **ContextClosedEvent**: This event is published when the ApplicationContext is closed using the close() method on the ConfigurableApplicationContext interface. A closed context reaches its end of life; it cannot be refreshed or restarted.
- **RequestHandledEvent**: This is a web-specific event telling all beans that an HTTP request has been serviced.

Q: What is Aspect?

A: A module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement. In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the @Aspect annotation (@AspectJ style).

Q: What is the difference between concern and cross-cutting concern in Spring AOP?

A: Concern: Concern is behavior which we want to have in a module of an application. Concern may be defined as a functionality we want to implement. Issues in which we are interested define our concerns.

Cross-cutting concern: It's a concern which is applicable throughout the application and it affects the entire application. e.g. logging , security and data transfer are the concerns which are needed in almost every module of an application, hence are cross-cutting concerns.

Q: What is Join point?

A: This represents a point in your application where you can plug-in AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework.

Q: What is Advice?

A: This is the actual action to be taken either before or after the method execution. This is actual piece of code that is invoked during program execution by Spring AOP framework.

Q: What is Pointcut?

A: This is a set of one or more joinpoints where an advice should be executed. You can specify pointcuts using expressions or patterns as we will see in our AOP examples.

Q: What is Introduction?

A: An introduction allows you to add new methods or attributes to existing classes.

Q: What is Target object?

A: The object being advised by one or more aspects, this object will always be a proxy object. Also referred to as the advised object.

Q: What is Weaving?

A: Weaving is the process of linking aspects with other application types or objects to create an advised object.

Q: What are the different points where weaving can be applied?

A: Weaving can be done at compile time, load time, or at runtime.

Q: What are the types of advice?

A: Spring aspects can work with five kinds of advice mentioned below:

- **before**: Run advice before the a method execution.
- **after**: Run advice after the a method execution regardless of its outcome.
- **after-returning**: Run advice after the a method execution only if method completes successfully.
- **after-throwing**: Run advice after the a method execution only if method exits by throwing an exception.
- **around**: Run advice before and after the advised method is invoked.

Q: What is XML Schema based aspect implementation?

A: Aspects are implemented using regular classes along with XML based configuration.

Q: What is @AspectJ? based aspect implementation?

A: @AspectJ refers to a style of declaring aspects as regular Java classes annotated with Java 5 annotations.

Q: How JDBC can be used more efficiently in spring framework?

A: JDBC can be used more efficiently with the help of a template class provided by spring framework called as JdbcTemplate.

Q: How JdbcTemplate can be used?

A: With use of Spring JDBC framework the burden of resource management and error handling is reduced a lot. So it leaves developers to write the statements and queries to get the data to and from the database. JdbcTemplate provides many convenience methods for doing things such as converting database data into primitives or objects, executing prepared and callable statements, and providing custom database error handling.

Q: What are the types of the transaction management Spring supports?

A: Spring supports two types of transaction management:

- **Programmatic transaction management:** This means that you have managed the transaction with the help of programming. That gives you extreme flexibility, but it is difficult to maintain.
- **Declarative transaction management:** This means you separate transaction management from the business code. You only use annotations or XML based configuration to manage the transactions.

Q: Which of the above transaction management type is preferable?

A: Declarative transaction management is preferable over programmatic transaction management though it is less flexible than programmatic transaction management, which allows you to control transactions through your code.

Q: What is Spring MVC framework?

A: The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

Q: What is a DispatcherServlet?

A: The Spring Web MVC framework is designed around a DispatcherServlet that handles all the HTTP requests and responses.

Q: What is WebApplicationContext ?

A: The *WebApplicationContext* is an extension of the plain *ApplicationContext* that has some extra features necessary for web applications. It differs from a normal *ApplicationContext* in that it is capable of resolving themes, and that it knows which servlet it is associated with.

Q: What are the advantages of Spring MVC over Struts MVC ?

A: Following are some of the advantages of Spring MVC over Struts MVC:

- Spring's MVC is very versatile and flexible based on interfaces but Struts forces Actions and Form object into concrete inheritance.
- Spring provides both interceptors and controllers, thus helps to factor out common behavior to the handling of many requests.
- Spring can be configured with different view technologies like Freemarker, JSP, Tiles, Velocity, XLST etc. and also you can create your own custom view mechanism by implementing Spring View interface.
- In Spring MVC Controllers can be configured using DI (IOC) that makes its testing and integration easy.
- Web tier of Spring MVC is easy to test than Struts web tier, because of the avoidance of forced concrete inheritance and explicit dependence of controllers on the dispatcher servlet.
- Struts force your Controllers to extend a Struts class but Spring doesn't, there are many convenience Controller implementations that you can choose to extend.
- In Struts, Actions are coupled to the view by defining ActionForwards within a ActionMapping or globally. Spring MVC has HandlerMapping interface to support this functionality.
- With Struts, validation is usually performed (implemented) in the validate method of an ActionForm. In Spring MVC, validators are business objects that are NOT dependent on the Servlet API which makes these validators to be reused in your business logic before persisting a domain object to a database.

Q: What is Controller in Spring MVC framework?

A: Controllers provide access to the application behavior that you typically define through a service interface. Controllers interpret user input and transform it into a model that is represented to the user by the view. Spring implements a controller in a very abstract way, which enables you to create a wide variety of controllers.

Q: Explain the *@Controller* annotation.

A: The *@Controller* annotation indicates that a particular class serves the role of a controller. Spring does not require you to extend any controller base class or reference the Servlet API.

Q: Explain *@RequestMapping* annotation.

A: *@RequestMapping* annotation is used to map a URL to either an entire class or a particular handler method.

Q: What are the ways to access Hibernate by using Spring?

A: There are two ways to access hibernate using spring:

- Inversion of Control with a Hibernate Template and Callback.

- Extending HibernateDAOSupport and Applying an AOP Interceptor node.

Q: What are ORM's Spring supports ?

A: Spring supports the following ORM's :

- Hibernate
- iBatis
- JPA (Java Persistence API)
- TopLink
- JDO (Java Data Objects)
- OJB

What is Next ?

Further you can go through your past assignments you have done with the subject and make sure you are able to speak confidently on them. If you are fresher then interviewer does not expect you will answer very complex questions, rather you have to make your basics concepts very strong.

Second it really doesn't matter much if you could not answer few questions but it matters that whatever you answered, you must have answered with confidence. So just feel confident during your interview. We at tutorialspoint wish you best luck to have a good interviewer and all the very best for your future endeavor. Cheers :-)