\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SPRING Interview Question\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 1:  What is Spring Framework?**

Spring is one of the most widely used Java EE frameworks. Spring framework core concepts are “**Dependency Injection**” and “**Aspect Oriented Programming**”.

Spring framework can be used in normal java applications also to achieve loose coupling between different components by implementing dependency injection and we can perform cross cutting tasks such as logging and authentication using spring support for aspect oriented programming.

**Question 2:  What are some of the important features and advantages of Spring Framework?**  
Spring Framework is built on top of two design concepts – Dependency Injection and Aspect Oriented Programming.

Some of the features of spring framework are:

* Lightweight and very little overhead of using framework for our development.
* Dependency Injection or Inversion of Control to write components that are independent of each other, spring container takes care of wiring them together to achieve our work.
* Spring IoC container manages Spring Bean life cycle and project specific configurations such as JNDI lookup.
* Spring MVC framework can be used to create web applications as well as restful web services capable of returning XML as well as JSON response.
* Support for transaction management, JDBC operations, File uploading, Exception Handling etc with very little configurations, either by using annotations or by spring bean configuration file.

**Question 3: What do you understand by Dependency Injection?**

Dependency Injection design pattern allows us to remove the hard-coded dependencies and make our application loosely coupled, extendable and maintainable. We can implement dependency injection pattern to move the dependency resolution from compile-time to runtime.

Some of the benefits of using Dependency Injection are: Separation of Concerns, Boilerplate Code reduction, Configurable components and easy unit testing.

**Question 4:  How do we implement DI in Spring Framework?**

We can use Spring XML based as well as Annotation based configuration to implement DI in spring applications.

**Question 5:  Name some of the important Spring Modules?**

Some of the important Spring Framework modules are:

**Spring Context** – for dependency injection.  
**Spring AOP** – for aspect oriented programming.  
**Spring DAO** – for database operations using DAO pattern  
**Spring JDBC** – for JDBC and DataSource support.  
**Spring ORM** – for ORM tools support such as Hibernate  
**Spring Web Module** – for creating web applications.  
**Spring MVC** – Model-View-Controller implementation for creating web applications, web services etc.

**Question 6:  What do you understand by Aspect Oriented Programming?**

Enterprise applications have some common cross-cutting concerns that is applicable for different types of Objects and application modules, such as logging, transaction management, data validation, authentication etc. In Object Oriented Programming, modularity of application is achieved by Classes whereas in AOP application modularity is achieved by Aspects and they are configured to cut across different classes methods.

AOP takes out the direct dependency of cross-cutting tasks from classes that is not possible in normal object oriented programming. For example, we can have a separate class for logging but again the classes will have to call these methods for logging the data.

**Question 7: What is Aspect, Advice, Pointcut, JointPoint and Advice Arguments in AOP?**

**Aspect:** Aspect is a class that implements cross-cutting concerns, such as transaction management. Aspects can be a normal class configured and then configured in Spring Bean configuration file or we can use Spring AspectJ support to declare a class as Aspect using @Aspect annotation.

**Advice:** Advice is the action taken for a particular join point. In terms of programming, they are methods that gets executed when a specific join point with matching pointcut is reached in the application

**Pointcut:** Pointcut are regular expressions that is matched with join points to determine whether advice needs to be executed or not. Pointcut uses different kinds of expressions that are matched with the join points. Spring framework uses the AspectJ pointcut expression language for determining the join points where advice methods will be applied.

**Join Point:** A join point is the specific point in the application such as method execution, exception handling, changing object variable values etc. In Spring AOP a join points is always the execution of a method.

**Advice Arguments:** We can pass arguments in the advice methods. We can use args() expression in the pointcut to be applied to any method that matches the argument pattern. If we use this, then we need to use the same name in the advice method from where argument type is determined.

**Question 8: What is the difference between Spring AOP and AspectJ AOP?**

AspectJ is the industry-standard implementation for Aspect Oriented Programming whereas Spring implements AOP for some cases. Main differences between Spring AOP and AspectJ are:

Spring AOP is simpler to use than AspectJ because we don’t need to worry about the weaving process.  
Spring AOP supports AspectJ annotations, so if you are familiar with AspectJ then working with Spring AOP is easier.  
Spring AOP supports only proxy-based AOP, so it can be applied only to method execution join points. AspectJ support all kinds of pointcuts.  
One of the shortcoming of Spring AOP is that it can be applied only to the beans created through Spring Context.

**Question 9: What is Spring IoC Container?**

Inversion of Control (IoC) is the mechanism to achieve loose-coupling between Objects dependencies. To achieve loose coupling and dynamic binding of the objects at runtime, the objects define their dependencies that are being injected by other assembler objects. Spring IoC container is the program that injects dependencies into an object and make it ready for our use.

Spring Framework IoC container classes are part of org.springframework.beans and org.springframework.context packages and provides us different ways to decouple the object dependencies.

Some of the useful ApplicationContext implementations that we use are;

**AnnotationConfigApplicationContext:** For standalone java applications using annotations based configuration

**ClassPathXmlApplicationContext:** For standalone java applications using XML based configuration.

**FileSystemXmlApplicationContext:** Similar to ClassPathXmlApplicationContext except that the xml configuration file can be loaded from anywhere in the file system.

AnnotationConfigWebApplicationContext and XmlWebApplicationContext for web applications.

**Question 10: What is a Spring Bean?**

Any normal java class that is initialized by Spring IoC container is called Spring Bean. We use Spring ApplicationContext to get the Spring Bean instance.

Spring IoC container manages the life cycle of Spring Bean, bean scopes and injecting any required dependencies in the bean.

**Question 11: What is the importance of Spring bean configuration file?**

We use Spring Bean configuration file to define all the beans that will be initialized by Spring Context. When we create the instance of Spring ApplicationContext, it reads the spring bean xml file and initialize all of them. Once the context is initialized, we can use it to get different bean instances.

Apart from Spring Bean configuration, this file also contains spring MVC interceptors, view resolvers and other elements to support annotations based configurations.

**Question 12: What are different scopes of Spring Bean?**

There are five scopes defined for Spring Beans.

**singleton:** Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure spring bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues because it’s not thread-safe.  
**prototype:** A new instance will be created every time the bean is requested.  
**request:** This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.  
**session:** A new bean will be created for each HTTP session by the container.  
**global-session:** This is used to create global session beans for Portlet applications.  
Spring Framework is extendable and we can create our own scopes too, however most of the times we are good with the scopes provided by the framework.

To set spring bean scopes we can use “scope” attribute in bean element or @Scope annotation for annotation based configurations.

**Question 13: What is Spring Bean life cycle?**

Spring Beans are initialized by Spring Container and all the dependencies are also injected. When context is destroyed, it also destroys all the initialized beans. This works well in most of the cases but sometimes we want to initialize other resources or do some validation before making our beans ready to use. Spring framework provides support for post-initialization and pre-destroy methods in spring beans.

We can do this by two ways – by implementing InitializingBean and DisposableBean interfaces or using init-method and destroy-method attribute in spring bean configurations.

**Question 14:  How to get ServletContext and ServletConfig object in a Spring Bean?**

There are two ways to get Container specific objects in the spring bean.

1. Implementing Spring \*Aware interfaces, for these ServletContextAware and ServletConfigAware interfaces, for complete example of these aware interfaces.  
2. Using @Autowired annotation with bean variable of type ServletContext and ServletConfig. They will work only in servlet container specific environment only though.

@Autowired

ServletContext servletContext;

**Question 15: What is Bean wiring and @Autowired annotation?**

The process of injection spring bean dependencies while initializing it called Spring Bean Wiring.

Usually it’s best practice to do the explicit wiring of all the bean dependencies, but spring framework also supports autowiring. We can use @Autowired annotation with fields or methods for autowiring byType. For this annotation to work, we also need to enable annotation based configuration in spring bean configuration file. This can be done by context:annotation-config element.

**Question 16: What are different types of Spring Bean autowiring?**

There are four types of autowiring in Spring framework.

autowire **byName**  
autowire **byType**  
autowire **by constructor**  
autowiring by @Autowired and @Qualifier annotations

Prior to Spring 3.1, autowire by autodetect was also supported that was similar to autowire by constructor or byType.

**Question 17: Does Spring Bean provide thread safety?**

The default scope of Spring bean is singleton, so there will be only one instance per context. That means that all the having a class level variable that any thread can update will lead to inconsistent data. Hence in default mode spring beans are not thread-safe.

However we can change spring bean scope to request, prototype or session to achieve thread-safety at the cost of performance. It’s a design decision and based on the project requirements.

**Question 18: What is a Controller in Spring MVC?**

Just like MVC design pattern, Controller is the class that takes care of all the client requests and send them to the configured resources to handle it. In Spring MVC, org.springframework.web.servlet.DispatcherServlet is the front controller class that initializes the context based on the spring beans configurations.

A Controller class is responsible to handle different kind of client requests based on the request mappings. We can create a controller class by using **@Controller** annotation. Usually it’s used with **@RequestMapping**annotation to define handler methods for specific URI mapping.

**Question 19: What’s the difference between @Component, @Controller, @Repository & @Service annotations in Spring?**

**@Component** is used to indicate that a class is a component. These classes are used for auto detection and configured as bean, when annotation based configurations are used.

**@Controller** is a specific type of component, used in MVC applications and mostly used with RequestMapping annotation.

**@Repository** annotation is used to indicate that a component is used as repository and a mechanism to store/retrieve/search data. We can apply this annotation with DAO pattern implementation classes.

**@Service** is used to indicate that a class is a Service. Usually the business facade classes that provide some services are annotated with this.

We can use any of the above annotations for a class for auto-detection but different types are provided so that you can easily distinguish the purpose of the annotated classes.

**Question 20: What is DispatcherServlet and ContextLoaderListener?**

DispatcherServlet is the front controller in the Spring MVC application and it loads the spring bean configuration file and initialize all the beans that are configured. If annotations are enabled, it also scans the packages and configure any bean annotated with @Component, @Controller, @Repository or @Service annotations.

ContextLoaderListener is the listener to start up and shut down Spring’s root WebApplicationContext. It’s important functions are to tie up the lifecycle of ApplicationContext to the lifecycle of the ServletContext and to automate the creation of ApplicationContext. We can use it to define shared beans that can be used across different spring contexts.

**Question 21: How to create ApplicationContext in a Java Program?**

There are following ways to create spring context in a standalone java program.

**AnnotationConfigApplicationContext:**If we are using Spring in standalone java applications and using annotations for Configuration, then we can use this to initialize the container and get the bean objects.

**ClassPathXmlApplicationContext:** If we have spring bean configuration xml file in standalone application, then we can use this class to load the file and get the container object.

**FileSystemXmlApplicationContext:** This is similar to ClassPathXmlApplicationContext except that the xml configuration file can be loaded from anywhere in the file system.

**Question 22: What are the minimum configurations needed to create Spring MVC application?**

For creating a simple Spring MVC application, we would need to do following tasks.

* Add spring-context and spring-webmvc dependencies in the project.
* Configure DispatcherServlet in the web.xml file to handle requests through spring container.
* Spring bean configuration file to define beans, if using annotations then it has to be configured here. Also we need to configure view resolver for view pages.
* Controller class with request mappings defined to handle the client requests.
* Above steps should be enough to create a simple Spring MVC Hello World application.

**Question 23: How would you relate Spring MVC Framework to MVC architecture?**

As the name suggests Spring MVC is built on top of Model-View-Controller architecture. DispatcherServlet is the Front Controller in the Spring MVC application that takes care of all the incoming requests and delegate it to different controller handler methods.

Model can be any Java Bean in the Spring Framework, just like any other MVC framework Spring provides automatic binding of form data to java beans. We can set model beans as attributes to be used in the view pages.

View Pages can be JSP, static HTMLs etc. and view resolvers are responsible for finding the correct view page. Once the view page is identified, control is given back to the DispatcherServlet controller. DispatcherServlet is responsible for rendering the view and returning the final response to the client.

**Question 24: What are some of the important Spring annotations you have used?**

Some of the Spring annotations that I have used in my project are:

**@Controller** – for controller classes in Spring MVC project.  
**@RequestMapping** – for configuring URI mapping in controller handler methods. This is a very important annotation.  
**@ResponseBody** – for sending Object as response, usually for sending XML or JSON data as response.  
**@PathVariable** – for mapping dynamic values from the URI to handler method arguments.  
**@Autowired** – for autowiring dependencies in spring beans.  
**@Qualifier** – with @Autowired annotation to avoid confusion when multiple instances of bean type is present.  
**@Service** – for service classes.  
**@Scope** – for configuring scope of the spring bean.  
**@Configuration, @ComponentScan and @Bean** – for java based configurations.  
AspectJ annotations for configuring aspects and advices, **@Aspect, @Before, @After, @Around, @Pointcut**etc.

**Question 25: How would you achieve Transaction Management in Spring?**

Spring framework provides transaction management support through Declarative Transaction Management as well as programmatic transaction management. Declarative transaction management is most widely used because it’s easy to use and works in most of the cases.

We use annotate a method with @Transactional annotation for Declarative transaction management. We need to configure transaction manager for the DataSource in the spring bean configuration file.

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource" />

</bean>