**SPRING QUESTIONS**

1.WHY DO WE USE SPRING FRAMEWORK?

It is an open-source application framework that is used for building Java applications and projects..

1.It is open source, lightweight and easy to learn.

2.build secure web applications.

3.Spring improves coding efficiency and reduces overall application development time because it is lightweight -- efficient at utilizing system resources -- and has a lot of support.

4. Spring removes tedious configuration work so that developers can focus on writing business logic.

2.Dependency

One object depends on other object for functioning properly is called dependency.

3.Dependency Injection

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application. Dependency Injection makes our programming code loosely coupled.

Dependency Injection is a fundamental aspect of the Spring framework, through which the Spring container “injects” objects into other objects or “dependencies”.This allows for loose coupling of components and moves the responsibility of managing components onto the container.

Spring framework provides two ways to inject dependency

* By Constructor
* By Setter method

4.Inversion of Control

Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework. We most often use it in the context of object-oriented programming.

Spring IoC (Inversion of Control) Container is the core of [Spring Framework](https://www.geeksforgeeks.org/introduction-to-spring-framework/). It creates the objects, configures and assembles their dependencies, manages their entire life cycle.

5.Ioc container

The Container uses Dependency Injection(DI) to manage the components that make up the application. It gets the information about the objects from a configuration file(XML) or Java Code or Java Annotations and Java POJO class. These objects are called Beans. Since the Controlling of Java objects and their lifecycle is not done by the developers, hence the name Inversion Of Control.  
There are 2 types of IoC containers:

* [BeanFactory](https://www.geeksforgeeks.org/spring-beanfactory/)
* [ApplicationContext](https://www.geeksforgeeks.org/spring-applicationcontext/)

That means if you want to use an IoC container in spring whether we need to use a BeanFactory or ApplicationContext.

6.Bean Factory

The BeanFactory is the most basic version of IoC containers

BeanFactory**loads beans on-demand**.

Thus, BeanFactory is lightweight as compared to ApplicationContext.

7.Application context

ApplicationContext extends the features of BeanFactory.

ApplicationContext**loads all beans at startup**

8.Abstract Application context

Abstract implementation of the [ApplicationContext](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/ApplicationContext.html" \o "interface in org.springframework.context) interface. Doesn't mandate the type of storage used for configuration; simply implements common context functionality. Uses the Template Method design pattern, requiring concrete subclasses to implement abstract methods.

9.Configuration metadata

The configuration metadata is represented in XML, Java annotations, or Java code. It allows you to express the objects that compose your application and the rich interdependencies between such objects. Several implementations of the ApplicationContext interface are supplied out-of-the-box with Spring.

Configuration metadata can be provided to Spring container in following ways:

* XML-Based configuration
* Annotation-Based configuration
* Java-based configuration:

10.bean

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.

11.Constructor injection

inject the dependency by constructor. The **<constructor-arg>** subelement of **<bean>** is used for constructor injection.

<constructor-arg value="10" type="int"></constructor-arg>

12.Setter injection

 inject the dependency by setter method also. The **<property>** subelement of **<bean>** is used for setter injection. Here we are going to inject

<bean id="obj" class="com.springdemo.Employee">

<property name="id">

<value>20</value>

</property>

</bean>

13.Autowiring in Spring

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

Autowiring can't be used to inject primitive and string values. It works with reference

Modes of autowiring

* + byType
  + byName
  + constructor
  + default

14.BEAN SCOPES

1.singleton

This scopes the bean definition to a single instance per Spring IoC container (default).

2.prototype

This scopes a single bean definition to have any number of object instances.

3. request

This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext

4. session

This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

5. global-session

This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext

15.Lifecycle of bean

Bean instantiated

Dependencies injected

Bean destroyed

16.Limitations of auto wiring

1. Explicit dependencies in property and constructor-arg settings always override autowiring.
2. You cannot autowire so-called simple properties such as primitives, Strings, and Classes (and arrays of such simple properties). This limitation in autowiring is by-design.

17.Spring annotations:

1. @Configuration: Used to indicate that a class declares one or more @Bean methods. These classes are processed by the Spring container to generate bean definitions and service requests for those beans at runtime.
2. @Bean: Indicates that a method produces a bean to be managed by the Spring container. This is one of the most used and important spring annotation. [@Bean](https://www.digitalocean.com/community/users/bean) annotation also can be used with parameters like name, initMethod and destroyMethod.
   * name – allows you give name for bean
   * initMethod – allows you to choose method which will be invoked on context register
   * destroyMethod – allows you to choose method which will be invoked on context shutdown
3. @PreDestroy and @PostConstruct are alternative way for bean initMethod and destroyMethod. It can be used when the bean class is defined by us. For example;
4. @ComponentScan: Configures component scanning directives for use with [@Configuration](https://www.digitalocean.com/community/users/configuration) classes. Here we can specify the base packages to scan for spring components.
5. @Component: Indicates that an annotated class is a “component”. Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.
6. @PropertySource: provides a simple declarative mechanism for adding a property source to Spring’s Environment. There is a similar annotation for adding an array of property source files i.e @PropertySources.
7. @Service: Indicates that an annotated class is a “Service”. This annotation serves as a specialization of [@Component](https://www.digitalocean.com/community/users/component), allowing for implementation classes to be autodetected through classpath scanning.
8. @Repository: Indicates that an annotated class is a “Repository”. This annotation serves as a specialization of [@Component](https://www.digitalocean.com/community/users/component) and advisable to use with [DAO](https://www.digitalocean.com/community/tutorials/dao-design-pattern) classes.
9. @Autowired: [Spring @Autowired annotation](https://www.digitalocean.com/community/tutorials/spring-autowired-annotation) is used for automatic injection of beans. Spring [@Qualifier](https://www.digitalocean.com/community/users/qualifier) annotation is used in conjunction with Autowired to avoid confusion when we have two of more bean configured for same type.