To see the complete explanation with code example please visit to "*thecafetechno.com/tutorials/spring/most-frequently-asked-spring-interview-questions-and-answers/"*

What is Dependency Injection in Spring ?

Answer : In spring objects define their associations (dependencies) and do not worry about how to get those dependencies ; now it is the responsibility of Spring to provide the required dependencies for creating objects. Spring creates the dependent objects and injects those object into the bean.

Suppose we have an object Employee and it has a dependency on object Address. So we define a bean corresponding to Employee where it will define its dependency on object Address. When Spring try to create an Object Employee it sees that Employee has a dependency on object Address so first it will create the Address object (dependent object) and then inject this into the Employee Object.

Given below is the xml configuration for injecting addressBean into the employeeBean using setter injection. To see complete example of DI click here.

```
<bean id="employeeBean" class="com.thecafetechno.Employee">
        <property name="name" value="Davinder"/>
        <property name="age" value="24"/>
        <property name="address" ref="addressBean"/>
</bean>
<bean id="addressBean" class="com.thecafetechno.Address">
        <property name="city" value="Hoshiarpur"/>
        <property name="state" value="Punjab"/>
</bean>
```

What is Inversion of Control in spring ?

Answer : IOC means inverting the control of creating the object from our own using new operator to Spring container.

Inversion of Control (also known as dependency injection) says you are not supposed to create application objects but describe how they should be created in a configuration file. To connect components with services just describe which service is needed by which component rather than directly connecting them.

Inversion of Control (IOC) and Dependency Injection are used interchangeably. IOC is achieved through DI. DI is the process of providing the dependencies of an object and IOC is the end result of DI. By DI the responsibility of creating objects is shifted from our application code to Spring container hence the phenomenon is called Inversion of Control (IOC).

What are different types of injection in Spring ?

Answer : Three types of injection in Spring3 Framework

1. Setter Injection
2. Constructor Injection

3. Getter or Method Injection

Benefits of using Spring Framework ?

Answer :

- Spring is relatively light weight container in comparison to other J2EE containers. It does not use much memory and CPU cycles for loading beans, providing services like transaction control , AOP management, JDBC interaction.
- Spring makes it easy to develop J2EE application as it has built-in support for WEB MVC framework.
- Spring helps creating loosely coupled applications by DI.
- Spring has no dependency on any application servers.
- Spring creates objects lazily which helps in developing light weight applications.
- Spring supports aspect oriented programming to manage logging, transaction, security etc.
- Spring makes Database interaction (operations) easier as it has support for data access techniques such as DAO, JDBC, Hibernate, IBATIS, JDO etc.
- With the help of spring application code can be unit tested easily as Spring provides unit testing support classes.
- Spring does not need unique deployment steps.
- Spring configuration is done in standard XML format which easy to write and understand.

Describe layered architecture of Spring Framework ?

Answer : Spring comprises of the following modules.

Core container : Basic functionality of Spring is provided by the core container. BeanFactory is the primary component of core container, an implementation of the Factory design pattern. The BeanFactory applies the Inversion of Control (IOC) pattern so that application's configuration and dependency specification can be separated from the actual application code.

Spring context : Context module is built on the base core container. The Context module inherits features of Beans module. It has support for internationalization , event-propagation, resource-loading. It also supports Java EE features such as EJB, JMX and basic remoting.

Spring AOP : The Spring AOP module provides aspect-oriented programming functionality into the Spring framework. We can apply AOP to any object managed by the Spring framework. Declarative transaction management can be applied into your application by using Spring AOP.

Spring DAO : The Spring DAO makes it easy to work with different database techniques like JDBC, ORM, OXM etc. It provides meaningful exception hierarchy for managing the exceptions and error messages thrown by vendor specific database.

Spring ORM : The Spring ORM module supports different ORM tools like JPA, JDO, Hibernate, and iBatis. We can use all of these O/R-mapping frameworks in combination with other features offered by Spring.

Spring Web module : Spring's Web package provides basic web-oriented integration features, such as multipart file-upload functionality, the initialization of the IoC container using servlet listeners

and a web-oriented application context. When using Spring together with WebWork or Struts, this is the package to integrate with.

Spring MVC framework : Spring MVC module provides a full-featured MVC implementation for building Web applications.

What is BeanFactory ?

BeanFactory (org.springframework.beans.factory.BeanFactory an interface) acts as Spring IoC container that is responsible for containing, configuring and managing the application beans. BeanFactory is the contract defined for Spring managed beans and all the beans managed by Spring implements this interface bydefault. It is the central IoC container interface in Spring Framework.

There are a many implementations of the BeanFactory interface. XmlBeanFactory class is most common implementation of BeanFactory. This implementation helps to define your application objects and dependencies between such objects, in terms of XML format. The XmlBeanFactory requires XML metadata and uses it to generate a fully configured system ready to use.

What is XmlBeanFactory ?

Answer : There are a many implementations of the BeanFactory interface. XmlBeanFactory class is most common implementation of BeanFactory. This implementation helps to define your application objects and dependencies between such objects, in terms of XML format. The XmlBeanFactory requires XML metadata and uses it to generate a fully configured system ready to use. XmlBeanFactory extends DefaultListableBeanFactory and defines two constructors.

What is ApplicationContext ?

Answer : ApplicationContext (org.springframework.context.ApplicationContext) is an interface which makes Spring a framework. It is derivd from BeanFactory, so has all the functionality of BeanFactory and adds lots more. Like BeanFactory it is responsible for containing, configuring and managing the application bean. In addtion to this it provides following framework oriented functionalities

- MessageSource, provide support for internationalization (I18N) messages .
- Access to resources, such as URLs and files.
- Event propagation for beans which implements the ApplicationListener interface.
- Loading of multiple (hierarchical) contexts
- Enterprise services like EJB integration ,JNDI,remoting and scheduling

What are the common implementations of the Application Context ?

There are three most commonly used implementation of 'Application Context' as

- ClassPathXmlApplicationContext : It loads the context definition from an XML file present in the classpath, treating context definitions as classpath resources. Given below is the code for loading context from classpath. File 'beans.xml' should be present in classpath of application.
  ApplicationContext appContext = new ClassPathXmlApplicationContext("beans.xml");
- FileSystemXmlApplicationContext : It loads the context definition from an XML file present in the filesystem. Need to give absolute path of the file 'beans.xml' as given below :

ApplicationContext appContext = new FileSystemXmlApplicationContext("F:/ java_program/SpringProject/src/com/thecafetechno/beans.xml");

- XmlWebApplicationContext : It loads the context definition from an XML file present within a web application.

What is Bean wiring ?

The process of linking (associating) beans with each other within a Spring container is called bean wiring. All the Spring managed beans of the application needs to be defined in an xml file. A bean may have dependency on the other bean. So these two beans are to be linked with each other and the process of linking beans is called bean wiring.

Given below are two beans and addressBean is wired (linked) with employeeBean.

```
<bean id="employeeBean" class="com.thecafetechno.Employee">
        <property name="name" value="Davinder"/>
        <property name="age" value="24"/>
        <property name="address" ref="addressBean"/>
</bean>
<bean id="addressBean" class="com.thecafetechno.Address">
        <property name="city" value="Hoshiarpur"/>
        <property name="state" value="Punjab"/>
</bean>
```

What is auto wiring ?

The Spring container provides the functionality of connecting beans with each other automatically called autowiring . Rather we inject one bean into the other using ref attribute, Spring can look into the BeanFactory and decide how to inject one bean into the other. So we don't explicitly connect (wire) beans instead spring does this for us.
Autowiring helps to minimize the need to specify properties or constructor arguments, thus saves a lot of typing . Autowiring is specific to individual beans, a bean may or may not use aotuwiring. To enable autowire for a bean, 'autowire' attribute is use inside <bean> tag.
There are five types of autowiring :

- no (default)
- byName
- byType
- constructor
- autodetect

'addressBean' is autowired with employeeBean using 'autowire=byType' mode.

```
        <bean id="employeeBean" class="com.thecafetechno.Employee"
autowire="byType">
                <property name="name" value="Davinder"/>
```

```
        <property name="age" value="24"/>
        <!--
           Because of autowiring third property is not needed to be defined
           <property name="address" ref="addressBean"/>
        -->
    </bean>
    <bean id="addressBean" class="com.thecafetechno.Address">
        <property name="city" value="Hoshiarpur"/>
        <property name="state" value="Punjab"/>
    </bean>
```

What are the different bean scopes in Spring ?

In Spring, creation of beans can be controlled by defining the scope of bean. To define scope attribute 'scope' of <bean> tag is used. There are five types of bean scopes in Spring

- singleton
- prototype
- request
- session
- global session

What is AOP ? Explain with example ?

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

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

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 in a module is concern of the module. Issues in which we are interested defines our concerns.

Cross-cutting concern : Its 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.

What is Aspect in Spring AOP?

Aspect defines the implementation of cross-cutting concern. We implement a cross-cutting functionality by creating an aspect. Aspect encapsulates our cross-cutting concern. There are two ways to define aspects in Spring. One is schema based(regular java class and configuration in spring configuration file) and other in annotation based (regular java class with annotation @Aspect) . An aspect contains pointcut and advice body.

What is Advice in Spring AOP?

Action performed by an aspect at a particular join point is called an Advice. Different types of advice are “around,” “before” ,”after”, “afterreturning” and “afterthrowing” . Advice is linked to pointcut and defines what/how and when to perform the action.

What is pointcut in Spring AOP?

A pointcut is an expression/predicate which points to a join point. Pointcut defines where to hook the advice. e.g. @Pointcut(“execution(* com.thecafetechno.Drive.drive(..))”) this pointcut is pointing to the execution of method ‘drive’ of class com.thecafetechno.Drive . Pointcut is associated with an advice.

What is Around advice ? Explain with an example ?

This advice is combination of before and after advices. It lets you do something before target method (join point) and some other things after the target method (join point). Around advice surrounds the join point. Around advice method takes an argument of type 'org.aspectj.lang.ProceedingJoinPoint'. In around advice, method 'proceed()' is called on ProceedingJoinPoint which invokes our target method.

```
// Around advice
    @Around("pointcutForDrive()")
    public void putSeatBelt(ProceedingJoinPoint pjp) throws Throwable {
        System.out.println("Around Advice : Driver puts seat belt");
        pjp.proceed(); //invokes the target method
        System.out.println("Around Advice : Driver reached destination
safely");
    }
```

What is join point ? Explain with an example?

Join point represents execution of a method where we want to hook the advice into.

Consider execution of a program and during this some methods are called say meth1(), meth2(), meth3(). Now we need to implement some cross-cutting functionality before or after execution of meth2(), this meth2() act as a join point.

A join point is referred by a pointcut.

Consider the pointcut @Pointcut("execution(public void xyz.meth2())") which refers to join point 'meth2()'

How to instantiates the IOC container ?

Common Implementation of ApplicationContext:

- classPathXmlApplicationContext
- fileSystemXmlApplicationContext
- xmlWebApplicationContext

How to customize bean initialization process ? [controlling bean initialization]

The Spring offers several callback interfaces to modify the behaviour of your bean. By implementing InitializingBean interface Spring Container will call afterPropertiesSet() method to allow the bean to perform certain initialization tasks after all properties on the bean have been injected by the container .

This can be achieved by following two ways

(a) Implemet InitializingBean interface and provide implementation of afterPropertiesSet() method.

```
<bean id="testBeanId" class="example.TestBean "/>

public class TestBean implements InitializingBean {

   public void afterPropertiesSet() {
      // do  initialization task
   }
}
```

(b) Usually the above described way of customizing the bean initialization process is avoided as it forces our bean class to implement an interface (InitializingBean) thus couples the client code to Spring Container. Same thing can be achieved by using 'init-method' attribute of 'bean' tag in Spring configuration file as given below.

```
<bean id="testBeanId" class="example.TestBean " init-method="init"/>
public class TestBean {
      public void init() {
      // do  initialization task
   }
}
```

How to customize bean destruction process ? [controlling bean destruction]

The Spring offers several callback interfaces to modify the behaviour of your bean. By implementing DisposableBean interface Spring Container will call destroy() method just before the container is destroyed.

This can be achieved by following two ways

(a) Implemet DisposableBean interface and provide implementation of destroy() method.

```
<bean id="testBeanId" class="example.TestBean "/>

public class TestBean implements DisposableBean {
  public void destroy() {
      // do  cleanup task
   }
}
```

(b) Usually the above described way of customizing the bean initialization process is avoided as it forces our bean class to implement an interface (DisposableBean) thus couples the client code to Spring Container. Same thing can be achieved by using 'init-method' attribute of 'bean' tag in Spring configuration file as given below.

<bean id="testBeanId" class="example.TestBean " destroy-method="cleanup"/>
public class TestBean {
    public void cleanup() {
    // do cleanup task
  }

}

How to implement inheritance in Bean definition ?

Inheritance can be implemented by specifying the 'parent' attribute in bean and passing id of (parent) bean that you want to inherit .

In the example below Employee bean inherits properties(name, age, sex) of Person bean, no need to specify these properties in child bean (Employee).

```
<?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 id="personBean" class="com.thecafetechno.Person">
              <property name="name" value="Davinder"/>
              <property name="age" value="24"/>
              <property name="sex" value="male"/>
            </bean>
            <bean id="employeeBean" class="com.thecafetechno.Employee"
parent="personBean">
              <property name="company" value="thecafetechno.com"/>
              <property name="address" ref="addressBean"/>
            </bean>
            <bean id="addressBean" class="com.thecafetechno.Address">
              <property name="city" value="Hoshiarpur"/>
              <property name="state" value="Punjab"/>
            </bean>
</beans>
```

How can you override beans default lifecycle methods?

The bean tag has two more important attributes with which you can define your own custom initialization and destroy methods.

Here I have shown a small demonstration. Two new methods fooSetup and fooTeardown are to be added to your Foo class.

<beans>

<bean id="bar" init-method="fooSetup" destroy="fooTeardown"/>

</beans>

Inner Bean in Spring ?

A bean which is defined (embedded) inside the property tag of other bean is called an inner bean. Inner beans can not be used (referred) by other beans except the enclosing one.