Q=Difference between @Autowired and @Required in spring?

Ans=@Autowired annotation is used when you want to autowire a bean. @Autowired is not limited to setter. It can be used with a constructor and a field as well. If you use @Autowired annotation on a field, that field will be autowired with bean having matching data type.

@Required checks if a particular property has been set or not. If a field has been annotated with @Required annotation and that field is not set, you will get org.springframework.beans.factory.BeanInitializationException.

Edit: As pointed by 'kryger': field annotated with @Autowired is effectively also @Required (unless you explicitly set its parameter required to false). eg:

@Autowired(required=false)

private ObjectType objectType;

For a field that has been annotated @Autowired, if bean with matching data type in not available, org.springframework.beans.factory.BeanCreationException is thrown.

Q=what is JSR-250 annotations in spring?

## Ans=@PostConstruct

## @PreDestroy

## @Resource

# **Q=**Difference Between @Resource, @Autowired and @Inject in Spring Injection?

Ans=

1. **@Resource** – Defined in the javax.annotation package and part of Java
2. **@Inject** – Defined in the javax.inject package and part of Java
3. **@Autowired** – Defined in the package org.springframework.bean.factory and part of Spring framework.

## Q=Difference between @Autowired vs @Inject Annotation?

## Ans=

1) The first and most important difference between @Autowired and @Inject annotation is that the @Inject annotation is only available from Spring 3.0 onwards, so if you want to use annotation-driven dependency injection in Spring 2.5 then you have to use the @Autowired annotation.  
  
2) The second difference between these two annotations is that unlike Spring's @Autowired,  the @Inject does require the **'required'** attribute.

# **3) The third most common difference between @Autowired and @Inject annotation is that former is Spring specific while later is the standard for**[**Dependency Injection**](http://javarevisited.blogspot.com/2015/06/difference-between-dependency-injection.html)**, specified in JSR-330. In general, I recommend the use of JSR 330 annotation for DI, the @Inject annotation is as capable as Spring's @Autowired and if you want you can also mix and match this with Spring's @Value and@Lazy annotations. 4) The @Autowired annotation was added on Spring 2.5 and used for annotation driven dependency injection. It works in conjunction with @Component annotation and <context:component-scan /> to streamline development cycle. From Spring 3.0, Spring offers support for JSR-330 dependency injection annotations e.g. @Inject, @Named, and @Singleton. It also added more Spring specific annotations e.g. @Primary, @Lazy, and @DependsOn annotation.  You can read**[**Spring in Action 4th Edition**](https://www.amazon.com/Spring-Action-Covers-4/dp/161729120X?tag=javamysqlanta-20)**to learn more about Spring specific annotations.** **5) The @Inject annotation is good from the portability point of view. Since @Autowired is specific to Spring framework, if you ever decided to move to**[**Google Guice**](https://github.com/google/guice)**or any other dependency injection framework then you need to re-implement your dependency injection logic, even though your application remains same. All bean creation logic needs to be changed to match with Google Guice's implementation.** Q=What is [stereotype annotations in spring?](https://stackoverflow.com/questions/16051656/advantages-of-using-spring-stereotypes)

Ans =@Controller,@Component,@Service,@Repository

Q=AOP

Ans= <aop:aspectj-autoproxy/>

When class implements any interface.:-

< aop:aspectj-autoproxy proxy-target-class=true/>

1. When multiple places poincut is using then we can write

@Poincut(“execution(\* com.fis.practice .OASISAOP.\*(..))”)

Public void selectAll(){

}

@After(value=” selectAll()”)

Public void selectAll(JointPoint jp){

}

@Before(value=” selectAll()”)

Public void selectAll(JointPoint jp){

}

Q:Bean scope of spring:

Ans=

| **Scope** | **Description** |
| --- | --- |
| [singleton](https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-factory-scopes-singleton) | (Default) Scopes a single bean definition to a single object instance per  Spring IoC container. |
| [prototype](https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-factory-scopes-prototype) | Scopes a single bean definition to any number of object instances. |
| [request](https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-factory-scopes-request) | Scopes a single bean definition to the lifecycle of a single HTTP request;  that is, each HTTP request has its own instance of a bean  created off the back of a single bean definition.  Only valid in the context of a web-aware Spring ApplicationContext. |
| [session](https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-factory-scopes-session) | Scopes a single bean definition to the lifecycle of an HTTP Session.  Only valid in the context of a web-aware Spring ApplicationContext. |
| [globalSession](https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-factory-scopes-global-session) | Scopes a single bean definition to the lifecycle of a global HTTP Session.  Typically only valid when used in a Portlet context.  Only valid in the context of a web-aware Spring ApplicationContext. |
| [application](https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-factory-scopes-application) | Scopes a single bean definition to the lifecycle of a ServletContext.  Only valid in the context of a web-aware Spring ApplicationContext. |
| [websocket](https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#websocket-stomp-websocket-scope) | Scopes a single bean definition to the lifecycle of a WebSocket.  Only valid in the context of a web-aware Spring ApplicationContext. |

Q=Spring bean inheritance?

Ans=for inharit one bean to another bean we neet to configure parent=”imported bean name”.

<bean id=”person” class=”com.fis.Person”>

<property name=”id” value=”101”/>

<property name=”name” value=”raj”/>

<property name = “age” value=”26”/>

</bean>

<bean id=”employee” class=”com.fis.Employee” **parent=”person”>**

<property name=”panno” value=”PQEWER23”/>

</bean>

We can also create inheritance template :-

<bean id=”person” abstract=”true”>

<property name=”id” value=”101”/>

<property name=”name” value=”raj”/>

<property name = “age” value=”26”/>

</bean>

<bean id=”employee” class=”com.fis.Employee” **parent=”person”>**

<property name=”panno” value=”PQEWER23”/>

</bean>

Q= how to execute beans sequencelly ?

Ans=using @dependsOn(“bean names”)

Q=how to resolve cyclic dependency?

Ans=using setter method injection.

Q= how to write call back methods global and local for every bean initialization time and destroy time?

Ans= for callback methods we can use:

1. Implements initializingBean and DesposableBean
2. In xml we can configure init-method=”method name” and destroy-method=method name”
3. We can configure globally in whole class with beans tag default-init-method=’method name” and default-destroy-method =”method name”.
4. Using @PostConstruct and @PreDestoy annotation from JSR 250 (Java specification request 250).For enable thease annotation we need to create a bean of CommonAnnotationBeanPostProcessor.

Q=Spring offers how many types of collection?

Ans=

5 types

1. Array
2. List
3. Set
4. Map
5. Props

Q=use of @Configuration?

Ans=**@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. The simplest possible @Configuration class would be as follows −

package com.tutorialspoint;

import org.springframework.context.annotation.\*;

@Configuration

public class HelloWorldConfig {

@Bean

public HelloWorld helloWorld(){

return new HelloWorld();

}

}

The above code will be equivalent to the following XML configuration −

<beans>

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld" />

</beans>

Once your configuration classes are defined, you can load and provide them to Spring container using *AnnotationConfigApplicationContext* as follows −

public static void main(String[] args) {

ApplicationContext ctx = new AnnotationConfigApplicationContext(HelloWorldConfig.class);

HelloWorld helloWorld = ctx.getBean(HelloWorld.class);

helloWorld.setMessage("Hello World!");

helloWorld.getMessage();

}

You can load various configuration classes as follows −

public static void main(String[] args) {

AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();

ctx.register(AppConfig.class, OtherConfig.class);

ctx.register(AdditionalConfig.class);

ctx.refresh();

MyService myService = ctx.getBean(MyService.class);

myService.doStuff();

}

Q=Diffrenece between bean Factory and application context design pattern?

Ans=

**BeanFactory**-Does not support the Annotation based dependency Injection.  
 **ApplicationContext**--Support Annotation based dependency Injection.-@Autowired, @PreDestroy  
 **BeanFactory**-Does not Support  
**ApplicationContext**- Application contexts can publish events to beans that are registered as listeners  
**BeanFactory**-Does not support way to access Message Bundle(internationalization (I18N)   
**ApplicationContext**-Support internationalization (I18N) messages.  
**BeanFactory**-Doesn’t support.  
**ApplicationContext**-Support  many enterprise services such JNDI access, EJB integration, remoting.  
**BeanFactory-**By default its support Lazy loading  
**ApplicationContext**-- its By default support Aggresive loading.

### **Q10. How to define the scope of a bean?**

To set Spring Bean’s scope, we can use @Scope annotation or “scope” attribute in XML configuration files. There are five supported scopes:

* **singleton**
* **prototype**
* **request**
* **session**
* **global-session**

### **Q12. What does the Spring bean lifecycle look like?**

First, a Spring bean needs to be instantiated, based on Java or XML bean definition. It may also be required to perform some initialization to get it into a usable state. After that, when the bean is no longer required, it will be removed from the IoC container.

The whole cycle with all initialization methods is shown on the image ([source](http://www.dineshonjava.com/2012/07/bean-lifecycle-and-callbacks.html)):

## <http://www.baeldung.com/wp-content/uploads/2017/06/Spring-Bean-Life-Cycle.jpg>

### **Q17. Name some of the Design Patterns used in the Spring Framework?**

* **Singleton Pattern:** Singleton-scoped beans
* **Factory Pattern:** Bean Factory classes
* **Prototype Pattern:** Prototype-scoped beans
* **Adapter Pattern:** Spring Web and Spring MVC
* **Proxy Pattern:** Spring Aspect Oriented Programming support
* **Template Method Pattern:** JdbcTemplate, HibernateTemplate, etc.
* **Front Controller:** Spring MVC DispatcherServlet
* **Data Access Object:** Spring DAO support
* **Model View Controller:**Spring MVC

# Q=**How to load multiple Spring bean configuration file?**

Ans=ApplicationContext context =

new ClassPathXmlApplicationContext(new String[] {"Spring-Common.xml",

"Spring-Connection.xml","Spring-ModuleA.xml"});

The above ways are lack of organizing and error prone, the better way should be organized all your Spring bean configuration files into a single XML file. For example, create a Spring-All-Module.xml file, and import the entire Spring bean files like this :

File : Spring-All-Module.xml

<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-2.5.xsd">

<import resource="common/Spring-Common.xml"/>

<import resource="connection/Spring-Connection.xml"/>

<import resource="moduleA/Spring-ModuleA.xml"/>

</beans>

Now you can load a single xml file like this :

ApplicationContext context =

new ClassPathXmlApplicationContext(Spring-All-Module.xml);

In web application (Spring mvc)

The framework will, on initialization of a DispatcherServlet, look for a file named [servlet-name]-servlet.xml in the WEB-INF directory of your web application and create the beans defined there (overriding the definitions of any beans defined with the same name in the global scope).

In intrafest-servlet.xml file you can use [import](http://static.springframework.org/spring/docs/2.5.x/reference/beans.html#beans-factory-xml-import) to compose your XML configuration.

<beans>

<bean id="bean1" class="..."/>

<bean id="bean2" class="..."/>

<import resource="foo-services.xml"/>

<import resource="foo-persistence.xml"/>

</beans>

There are a couple of ways to do this.

**1. web.xml contextConfigLocation**

Your first option is to load them all into your Web application context via the ContextConfigLocation element. You’re already going to have your primary applicationContext here, assuming you’re writing a web application. All you need to do is put some white space between the declaration of the next context.

<context-param>

<param-name> contextConfigLocation </param-name>

<param-value>

applicationContext1.xml

applicationContext2.xml

</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

The above uses carriage returns. Alternatively, yo could just put in a space.

<context-param>

<param-name> contextConfigLocation </param-name>

<param-value> applicationContext1.xml applicationContext2.xml </param-value>

</context-param>

<listener>

<listener-class> org.springframework.web.context.ContextLoaderListener </listener-class>

</listener>

**2. applicationContext.xml import resource**

Your other option is to just add your primary applicationContext.xml to the web.xml and then use import statements in that primary context.

In applicationContext.xml you might have…

<!-- hibernate configuration and mappings -->

<import resource="applicationContext-hibernate.xml"/>

<!-- ldap -->

<import resource="applicationContext-ldap.xml"/>

<!-- aspects -->

<import resource="applicationContext-aspects.xml"/>

**Which strategy should you use?**

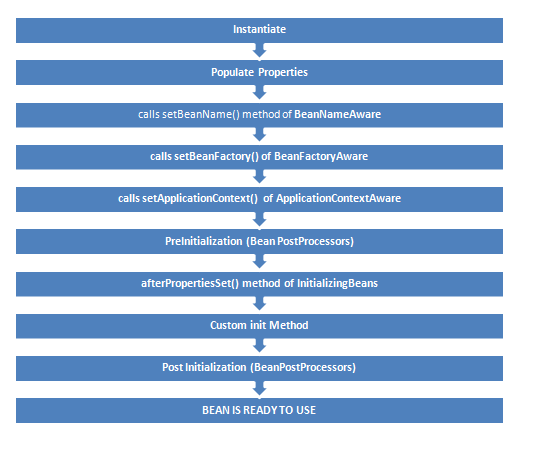
**1.** I always prefer to load up via **web.xml**.

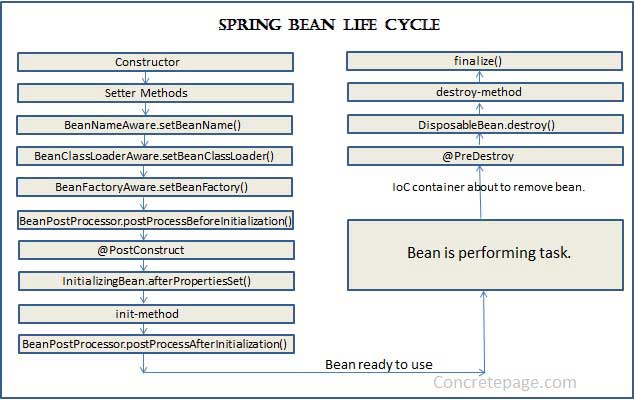
Because , this allows me to keep all contexts isolated from each other. With tests, we can load just the contexts that we need to run those tests. This makes development more modular too as components stay loosely coupled, so that in the future I can extract a package or vertical layer and move it to its own module.

**2.** If you are loading contexts into a non-web application, I would use the import resource.

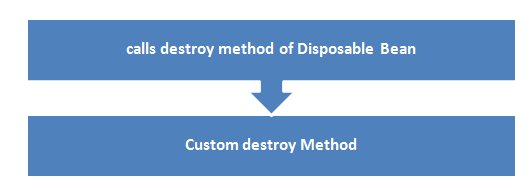
Q-spring life cycle?

Ans=

Below diagram shows the complete lifecycle methods (from instantiate to Ready To use )



Following diagram shows the method calling  at the time of destruction.



Q=what is @PostConstruct and @PreDestroy in spring? And where they are live in spring lifecycle?

Ans- @PostConstruct and @Predestroy method is java annotations.there are not related to spring but we can use this.

1. @PostConstruct is a JSR-250 annotation while init-method is Spring's way of having an initializing method.
2. If you have a @PostConstruct method, this will be called first before the initializing methods are called.
3. If your bean implements InitializingBean and overrides afterPropertiesSet , first @PostConstruct is called, then the afterPropertiesSet and then init-method.

<https://stackoverflow.com/questions/8519187/spring-postconstruct-vs-init-method-attribute>

Q=how to get object of spring bean which has no id?

Q=in real time how will u use prototype scope of bean?