**Spring Interview Question’s for 4-6 years**

#### What are benefits of Spring Framework?

1. **Lightweight:** Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 2MB.
2. **Inversion of control (IOC):** Loose coupling is achieved in Spring, with the [Inversion of Control technique](http://www.javacodegeeks.com/2011/08/what-is-dependency-inversion-is-it-ioc.html). The objects give their dependencies instead of creating or looking for dependent objects.
3. **Aspect oriented (AOP):** [Spring supports Aspect oriented programming](http://www.javacodegeeks.com/2011/01/aspect-oriented-programming-spring-aop.html) and separates application business logic from system services.
4. **Container:** Spring contains and manages the life cycle and configuration of application objects.
5. **MVC Framework:** Spring’s web framework is a well-designed [web MVC framework](http://www.javacodegeeks.com/2011/02/spring-mvc-development-tutorial.html), which provides a great alternative to web frameworks.
6. **Transaction Management:** Spring provides a consistent transaction management interface that can scale down to a local transaction and scale up to global transactions (JTA).
7. **Exception Handling:** Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO) into consistent, unchecked exceptions.

#### Explain the Core Container (Application context) module

This is the basic Spring module, which provides the fundamental functionality of the Spring framework. BeanFactory is the heart of any spring-based application. Spring framework was built on the top of this module, which makes the Spring container.

#### What is Spring IoC container?

The Spring IoC is responsible for creating the objects,managing them (with dependency injection (DI)), wiring them together, configuring them, as also managing their complete lifecycle.

#### What are the common implementations of the ApplicationContext?

The **FileSystemXmlApplicationContext** container loads the definitions of the beans from an XML file. The full path of the XML bean configuration file must be provided to the constructor.  
The **ClassPathXmlApplicationContext** container also loads the definitions of the beans from an XML file. Here, you need to set CLASSPATH properly because this container will look bean configuration XML file in CLASSPATH.  
The **WebXmlApplicationContext:** container loads the XML file with definitions of all beans from within a web application.

#### What is the difference between Bean Factory and ApplicationContext?

Application contexts provide a means for resolving text messages, a generic way to load file resources (such as images), they can publish events to beans that are registered as listeners. In addition, 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.

#### What are the different types of IoC (dependency injection) and which DI would you suggest Constructor-based or setter-based DI?

* **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.

You can use both Constructor-based and Setter-based Dependency Injection. The best solution is using constructor arguments for mandatory dependencies and setters for optional dependencies.

#### Explain the bean scopes supported by Spring

There are five scoped provided by the Spring Framework supports following five scopes:

* In **singleton** scope, Spring scopes the bean definition to a single instance per Spring IoC container.
* In **prototype** scope, a single bean definition has any number of object instances.
* In **request** scope, a bean is defined to an HTTP request. This scope is valid only in a web-aware Spring ApplicationContext.
* In **session** scope, a bean definition is scoped to an HTTP session. This scope is also valid only in a web-aware Spring ApplicationContext.
* In **global-session** scope, a bean definition is scoped to a global HTTP session. This is also a case used in a web-aware Spring ApplicationContext.

The default scope of a Spring Bean is Singleton.

#### Are Singleton beans thread safe in Spring Framework?

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

#### Bean lifecycle in Spring framework

* The spring container finds the bean’s definition from the XML file and instantiates the bean.
* Spring populates all of the properties as specified in the bean definition (DI).
* If the bean implements BeanNameAware interface, spring passes the bean’s id to setBeanName() method.
* If Bean implements BeanFactoryAware interface, spring passes the beanfactory to setBeanFactory() method.
* If there are any bean BeanPostProcessors associated with the bean, Spring calls postProcesserBeforeInitialization() method.
* If the bean implements IntializingBean, its afterPropertySet() method is called. If the bean has init method declaration, the specified initialization method is called.
* If there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization() methods will be called.
* If the bean implements DisposableBean, it will call the destroy() method.

#### Which are the important beans lifecycle methods? Can you override them?

There are two important bean lifecycle methods. The first one is setup which is called when the bean is loaded in to the container. The second method is the teardown method which is called when the bean is unloaded from the container.  
The bean tag has two important attributes (init-method and destroy-method) with which you can define your own custom initialization and destroy methods. There are also the correspondive annotations(@PostConstruct and @PreDestroy).

#### Are there limitations with autowiring?

Limitations of autowiring are:

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

#### What is @Qualifier annotation

When there are more than one beans of the same type and only one is needed to be wired with a property, the @Qualifier annotation is used along with @Autowired annotation to remove the confusion by specifying which exact bean will be wired.

#### How can we integrate Spring and Hibernate using HibernateDaoSupport?

Use Spring’s SessionFactory called LocalSessionFactory. The integration process is of 3 steps:

* Configure the Hibernate SessionFactory
* Extend a DAO Implementation from HibernateDaoSupport
* Wire in Transaction Support with AOP

#### Which Transaction management type is more preferable?

Most users of the Spring Framework choose declarative transaction management because it is the option with the least impact on application code, and hence is most consistent with the ideals of a non-invasive lightweight container. 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.

#### What do you understand by @Qualifier annotation?

When you create more than one bean of the same type and want to wire only one of them with a property you can use the **@Qualifier** annotation along with **@Autowired** to remove the ambiguity by specifying which exact bean should be wired.

#### Name the types of transaction management that Spring supports.

Two types of transaction management are supported by Spring. They are:

1. **Programmatic transaction management:** In this, the transaction is managed with the help of programming. It provides you extreme flexibility, but it is very difficult to maintain.
2. **Declarative transaction management:** In this, the transaction management is separated from the business code. Only annotations or XML based configurations are used to manage the transactions.

#### What do you mean by Aspect?

Aspect is a modularization of concern which cuts across multiple objects. Transaction management is a good example of a crosscutting concern in J2EE applications. Aspects are implemented using regular classes or regular classes annotated with the @Aspect annotation in Spring Framework.

#### How to inject a java.util.Properties into a Spring Bean?

We need to define propertyConfigurer bean that will load the properties from the given property file. Then we can use Spring EL support to inject properties into other bean dependencies. For example;

<bean id="propertyConfigurer"

class="org.springframework.context.support.PropertySourcesPlaceholderConfigurer">

<property name="location" value="/WEB-INF/application.properties" />

</bean>

<bean class="com.journaldev.spring.EmployeeDaoImpl">

<property name="maxReadResults" value="${results.read.max}"/>

</bean>

If you are using annotation to configure the spring bean, then you can inject property like below.

Copy

@Value("${maxReadResults}")

private int maxReadResults;

#### What requirements does JPA set for Embeddable classes?

1. Such classes must satisfy the same rules as the Entity classes, except that they do not have to contain a primary key and be marked with the Entity annotation (see question 10),
2. The Embeddable class must be marked with the Embeddable annotation or described in the XML configuration file JPA.

#### What types of connections (relationship) between Entity do you know (list eight types, or specify four types of connections, each of which can be further divided into two types)?

There are four types of connections

1. OneToOne (one-to-one connection, that is, one Entity object can be associated with no more than one object of another Entity),

1. OneToMany (one-to-many connection, one Entity object can be associated with a whole collection of Entity)
2. ManyToOne (many to one link, feedback for OneToMany),
3. ManyToMany (many to many link) Each of which can be divided into two types:
4. Bidirectional
5. Unidirectional – a link to a link is set for all Entity, that is, in the case of OneToOne AB, Entity A has a link to Entity B, Entity B has a link to Entity A, Entity A is considered the owner of this link (this is important for cases of cascading data deletion , then deleting A will also delete B, but not vice versa). Undirectional- link to link is established only on one side, that is, in the case of OneToOne AB only Entity A will have link to Entity B, Entity B will not have link to A .