1. What is spring?

Spring is a **light weight framework**. It provides support to various framework like hibernate, struts etc. so, it can be treated as frame work of frame work of framework.

Spring framework comprises of serval modules such as IOC, AOP, ORM,MVC etc.

1. **What is dependency injection?**

Dependency injection is a **design pattern** which provides loosely coupling between application classes.

For example, you want to call DAO class method in service class then we usually create object by using new keyword. But by using spring instead of using new keyword we create spring beans and provide reference to DAO class object and call the methods which are done by using **spring configuration files(.xml)** or by **auto wiring** them and dependency injection will take care during runtime.

1. What are the advantages of Spring framework?

* Makes the code **loosely coupled** so it’s easy to maintain.
* Makes the code **easy to test**.
* Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies. So, there is no need to write too much code. It hides the basic steps of these technologies.

1. What are the different types of IOC (dependency injection)?

Spring frame work provides two way to inject dependency’s

We are going to inject

* Primitive and String-based values
* Dependent object
* Collections values.

**By constructor:** The **<constructor-arg>** sub-element of **<bean>** is used for constructor injection.

**By Setter method:** The **<property>** sub-element of **<bean>** is used for setter injection.

1. **What are the benefits of IOC (Dependency Injection)?**

* **Loose coupling:** Components can be added Declaratively so we can add and remove the components without code change.

**For Example:**

<bean id="createCreditCard" class="springexample.creditcardaccount.CreateCreditCardAccount">

<property name="emailInterface">

<ref bean="email" />

</property>

<property name="smsInterface">

<ref bean="sms" />

</property>

<property name="daoInterface">

<ref bean="dao" />

</property>

</bean>

There are three components associated with createCreditCard.

Tommorow you don't want smsInterface so just remove the xml configuration.

<property name="smsInterface">

<ref bean="sms" />

</property>

* **Not Required any singletons:** Don't need to code for singleton class. Every class is by default singleton.
* You can make not singleton by making singleton="false".

**For Example:**  
<bean id="sms" class="springexample.sms.SMS" singleton="false">   
</bean>

* + Objects are created Lazily.
  + Initialization of properties is easy? no need to read from properties file.

**For Example:**   
<bean id="email" class="springexample.email.Email" >   
        <property name="smtpHost">   
                <value>smtp.sa.com</value>   
        </property>   
</bean>   
IOC Contained set the setter method of Email class. setSmtpHost(). You don't need to read from properties file with extra coding.

1. What are features of Spring?

Coming to features of spring framework, Spring is **Lightweight, Inversion of control (IOC), Aspect oriented (AOP), Container, MVC Framework etc.**

* **Lightweight:** Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 1MB. And the processing overhead is also very negligible.
* **Inversion of control (IOC):** Loose coupling is achieved in spring using the technique Inversion of Control. The objects give their dependencies instead of creating or looking for dependent objects.
* **Aspect oriented (AOP):** Spring supports Aspect oriented programming and enables cohesive development by separating application business logic from system services.
* **Container:** Spring contains and manages the life cycle and configuration of application objects.

1. Which one is better in injection, setter or constructor?

* setter Injection in more **readable** than constructor injection in Spring configuration file usually applicationContext.xml . Since setter method has name e.g. setFirstName() by reading Spring XML config file you know which dependency you are setting. While in constructor injection, since it uses an index to inject the dependency, it's **not as readable** as setter injection and you need to refer either Java documentation or code to find which index corresponds to which property.
* Spring and one of the drawback of setter injection is that it **does not ensures** dependency Injection. You cannot guarantee that certain dependency is injected or not, which means you may have an object with incomplete dependency. On other hand constructor Injection, does not **allow you** to construct object, until your dependencies are ready.
* By using **setter injection,** **you can override certain dependency** which is not possible which is **not possible with constructor injection** because every time you call the constructor, a new object is gets created.

1. What is the difference between Bean Factory and Application Context?

* Application contexts provide a means for resolving text messages, including support for i18n of those messages.
* Application contexts provide a generic way to load file resources, such as images.
* One of the popular implementation of BeanFactory interface is XMLBeanFactory while one of the popular implementation of ApplicationContext interface is ClassPathXmlApplicationContext. On Java web application, we use WebApplicationContext which extends ApplicationContext interface and adds getServletContext method.
* BeanFactory is OK for testing and nonproduction use but ApplicationContext is more feature rich container implementation and should be favored over BeanFactory.

1. What are the common implementations of the Application Context?

**ClassPathXmlApplicationContext, FileSystemXmlApplicationContext , XmlWebApplicationContext  are the 3 commonly used** implementations of Application Context. Coming to

**ClassPathXmlApplicationContext:** It Loads context definition from an XML file located in the classpath, treating context definitions as classpath resources. The application context is loaded from the application's classpath by using the code.  
ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");

**FileSystemXmlApplicationContext:** It loads context definition from an XML file in the filesystem. The application context is loaded from the file system by using the code.  
ApplicationContext context = new FileSystemXmlApplicationContext("bean.xml");

**XmlWebApplicationContext:**It loads context definition from an XML file contained within a web application.

1. How do you load spring configuration file?
2. What is the typical Bean life cycle in Spring Bean Factory Container?

Spring framework provides following **4 ways for controlling life cycle events** of bean:

* InitializingBean and DisposableBean callback interfaces.
* Other Aware interfaces for specific behavior.
* custom init() and destroy() methods in bean configuration file.
* @PostConstruct and @PreDestroy annotations.

The spring container finds the bean’s definition from the XML file and instantiates the bean.Spring populates all 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.

1. What do you mean by Bean wiring?

The act of creating associations between application components (beans) within the Spring container is referred to as Bean wiring.

1. What do you mean by Auto Wiring?

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 only.

We have different types of autowiring such as **no**, **byName**, **byType**, **constructor**, **autodetect**.

1. What are the ways to access Hibernate using Spring?

We can access hibernate by using spring

* Inversion of Control with a Hibernate Template and Callback.
* Extending HibernateDAOSupport and Applying an AOP Interceptor node.

15) Steps to integrate Hibernate in Spring?

The process of integration follows

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

1. What are Bean scopes in Spring Framework?

* 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.

1. Why Spring beans are Singleton?

Default scope is singleton to make sure that minimum number of objects are created during bean factory instantiation.

**Quote from Spring documentation:**

When a bean is a singleton, only one shared instance of the bean will be managed and all requests for beans with an id or ids matching that bean definition will result in that one specific bean instance being returned.

1. When do you need to have prototype for Spring Beans?

If single bean instance does not fit into specific scenario, user can choose to set beans scope as prototype.

1. What is AOP?

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

1. How the AOP used in Spring?

Spring AOP is used as follows

* By Spring1.2 Old style (dtd based) (also supported in Spring3)
* By AspectJ annotation-style
* By Spring XML configuration-style (schema based)

1. What do you mean by Aspect?

A modularization of a concern that cuts across multiple objects.

What you want to achieve through aspect oriented programming.

For example: At every method call in business layer we want to intercept and log input parameter her our aspect is logging.

1. What do you mean by Joint Point?

Join point is a concept which is a point in program such as method execution, exception handling, field access etc. Spring supports only method execution join point.

1. What do you mean by Advice?

Advice is a concept which represents an action taken by aspect at a joint point.

1. What are the types of Advice?

* **Before Advice**: it executes before a join point.
* **After Returning Advice**: it executes after a joint point completes normally.
* **After Throwing Advice**: it executes if method exits by throwing an exception.
* **After (finally) Advice**: it executes after a join point regardless of join point exit whether normally or exceptional return.
* **Around Advice**: It executes before and after a join point.

1. For what purpose, you have used Spring AOP?

I have used Spring AOP for **logging** purpose. In my business layer at every method call I intercepted it and logged **input** and **output** **parameter**. Here my aspect is logging and

1. How do you use Spring AOP for transaction management?
2. What are the types of the transaction management Spring supports?

* Spring Framework supports:
  + - * Programmatic transaction management.
      * Declarative transaction management.

1. What are the benefits of the Spring Framework transaction management?

The Spring Framework provides a consistent abstraction for transaction management that delivers the following benefits:

* + - * Provides a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA, and JDO.
      * Supports declarative transaction management.
      * Provides a simpler API for programmatic transaction management than several complex transaction APIs such as JTA.
      * Integrates very well with Spring's various data access abstractions.

1. Explain about the Spring DAO support?

The Data Access Object (DAO) support in Spring is aimed at making it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows one to switch between the persistence technologies easily and it also allows one to code without worrying about catching exceptions that are specific to each technology.

1. What are the exceptions thrown by the Spring DAO classes?
   * + - Spring DAO classes throw exceptions which are subclasses ofDataAccessException(org.springframework.dao.DataAccessException).
       - Spring provides a convenient translation from technology-specific exceptions like SQLException to its own exception class hierarchy with the DataAccessException as the root exception. These exceptions wrap the original exception.
2. What is Spring's Jdbc Template?

Spring's JdbcTemplate is central class to interact with a database through JDBC. JdbcTemplate provides many convenience methods for doing things such as converting database data into primitives or objects, executing prepared and callable statements, and providing custom database error handling.

**JdbcTemplate template = new JdbcTemplate(myDataSource);**