Spring Interview Questions form java2share.com

1. What is IOC (or Dependency Injection)?
The basic concept of the Inversion of Control pattern (also known as dependency injection) is that you do not create your objects but describe how they should be created. You don't directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A container (in the case of the Spring framework, the IOC container) is then responsible for hooking it all up.

i.e., Applying IoC, objects are given their dependencies at creation time by some external entity that coordinates each object in the system. That is, dependencies are injected into objects. So, IoC means an inversion of responsibility with regard to how an object obtains references to collaborating objects.

2. What are the different types of IOC (dependency injection) ?
There are three types of dependency injection:

- **Constructor Injection** (e.g. Pico container, Spring etc): Dependencies are provided as constructor parameters.
- **Setter Injection** (e.g. Spring): Dependencies are assigned through JavaBeans properties (ex: setter methods).
- **Interface Injection** (e.g. Avalon): Injection is done through an interface.

Note: Spring supports only Constructor and Setter Injection

3. What are the benefits of IOC (Dependency Injection)?
Benefits of IOC (Dependency Injection) are as follows:

- Minimizes the amount of code in your application. With IOC containers you do not care about how services are created and how you get references to the ones you need. You can also easily add additional services by adding a new constructor or a setter method with little or no extra configuration.
- Make your application more testable by not requiring any singletons or JNDI lookup mechanisms in your unit test cases. IOC containers make unit testing and switching implementations very easy by manually allowing you to inject your own objects into the object under test.
- Loose coupling is promoted with minimal effort and least intrusive mechanism. The factory design pattern is more intrusive because components or services need to be requested explicitly whereas in IOC the dependency is injected into requesting piece of code. Also some containers promote the design to interfaces not to implementations design concept by encouraging managed objects to implement a well-defined service interface of your own.
- IOC containers support eager instantiation and lazy loading of services. Containers also provide support for instantiation of managed objects, cyclical dependencies, life cycles management, and dependency resolution between managed objects etc.

4. What is Spring ?
Spring is an open source framework created to address the complexity of enterprise application development. One of the chief advantages of the Spring framework is its layered architecture, which allows you to be selective about which of its components you use while also providing a cohesive framework for J2EE application development.

5. What are the advantages of Spring framework?
The advantages of Spring are as follows:

- Spring has layered architecture. Use what you need and leave you don't need now.
- Spring Enables POJO Programming. There is no behind the scene magic here. POJO programming enables continuous integration and testability.
- Dependency Injection and Inversion of Control Simplifies JDBC
- Open source and no vendor lock-in.

6. What are features of Spring ?

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.

MVC Framework:
Spring comes with MVC web application framework, built on core Spring functionality. This framework is highly configurable via strategy interfaces, and accommodates multiple view technologies like JSP, Velocity, Tiles, iText, and POI. But other frameworks can be easily used instead of Spring MVC Framework.

Transaction Management:
Spring framework provides a generic abstraction layer for transaction management. This allowing the developer to add the pluggable transaction managers, and making it easy to demarcate transactions without dealing with low-level issues. Spring's transaction support is not tied to J2EE environments and it can be also used in container less environments.

JDBC Exception Handling:
The JDBC abstraction layer of the Spring offers a meaningful exception hierarchy, which simplifies the error handling strategy. Integration with Hibernate, JDO, and iBATIS: Spring provides best Integration services with Hibernate, JDO and iBATIS
contexts for Web-based applications. As a result, the Spring framework supports integration with Jakarta Struts. The Web module also eases the tasks of handling multi-part requests and binding request parameters to domain objects.

7 . What is web module?

This module is built on the application context module, providing a context that is appropriate for web-based applications. This module also contains support for several web-oriented tasks such as transparently handling multipart requests for file uploads and programmatic binding of request parameters to your business objects. It also contains integration support with *Jakarta Struts*.

8. What are the types of Dependency Injection Spring supports?

Setter Injection:
Setter-based DI is realized by calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

Constructor Injection:
Constructor-based DI is realized by invoking a constructor with a number of arguments, each representing a collaborator.

9. What is Bean Factory ?
A BeanFactory is like a factory class that contains a collection of beans. The BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients.

- BeanFactory is able to create associations between collaborating objects as they are instantiated. This removes the burden of configuration from bean itself and the beans client.
- BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

10. What is Application Context?
A bean factory is fine to simple applications, but to take advantage of the full power of the Spring framework, you may want to move up to Springs more advanced container, the application context. On the surface, an application context is same as a bean factory.Both load bean definitions, wire beans together, and dispense beans upon request. But it also provides:

- A means for resolving text messages, including support for internationalization.
- A generic way to load file resources.
- Events to beans that are registered as listeners.

11. What is the difference between Bean Factory and Application Context ?
On the surface, an application context is same as a bean factory. But application context offers much more..

- 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.
- Application contexts can publish events to beans that are registered as listeners.
- Certain 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.
- ResourceLoader support: Spring's Resource interface us a flexible generic abstraction for handling low-level resources. An application context itself is a ResourceLoader, Hence provides an application with access to deployment-specific Resource instances.

- MessageSource support: The application context implements MessageSource, an interface used to obtain localized messages, with the actual implementation being pluggable

12. What are the common implementations of the Application Context ?
The three commonly used implementation of 'Application Context' are

- **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.

13. How is a typical spring implementation look like ?
For a typical Spring Application we need the following files:

- An interface that defines the functions.
- An Implementation that contains properties, its setter and getter methods, functions etc.,
- Spring AOP (Aspect Oriented Programming)
- A XML file called Spring configuration file.
- Client program that uses the function.

14. What is the typical Bean life cycle in Spring Bean Factory Container ?
Bean life cycle in Spring Bean Factory Container is as follows:

- The spring container finds the bean's definition from the XML file and instantiates the bean.
- Using the dependency injection, spring populates all of the properties as specified in the bean definition
- If the bean implements the BeanNameAware interface, the factory calls setBeanName() passing the bean's ID.
- If the bean implements the BeanFactoryAware interface, the factory calls setBeanFactory(), passing an instance of itself.
- If there are any BeanPostProcessors associated with the bean, their post-ProcessBeforeInitialization() methods will be called.
- If an init-method is specified for the bean, it will be called.
- Finally, if there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization() methods will be called.

15. What do you mean by Bean wiring ?
The act of creating associations between application components (beans) within the Spring container is reffered to as Bean wiring.

16. What do you mean by Auto Wiring?
The Spring container is able to autowire relationships between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for

your bean by inspecting the contents of the BeanFactory. The autowiring functionality has *five modes*.

- no
- byName
- byType
- constructor
- autodirect

17. What is DelegatingVariableResolver?
Spring provides a custom JavaServer Faces VariableResolver implementation that extends the standard Java Server Faces managed beans mechanism which lets you use JSF and Spring together. This variable resolver is called as *DelegatingVariableResolver*

18. How to integrate Java Server Faces (JSF) with Spring?
JSF and Spring do share some of the same features, most noticeably in the area of IOC services. By declaring JSF managed-beans in the faces-config.xml configuration file, you allow the FacesServlet to instantiate that bean at startup. Your JSF pages have access to these beans and all of their properties.We can integrate JSF and Spring in two ways:

- **DelegatingVariableResolver:** Spring comes with a JSF variable resolver that lets you use JSF and Spring together.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
  "http://www.springframework.org/dtd/spring-beans.dtd">

<faces-config>
  <application>
    <variable-resolver>
      org.springframework.web.jsf.DelegatingVariableResolver
    </variable-resolver>
  </application>
</faces-config>
```

The DelegatingVariableResolver will first delegate value lookups to the default resolver of the underlying JSF implementation, and then to Spring's 'business context' WebApplicationContext. This allows one to easily inject dependencies into one's JSF-managed beans.

- FacesContextUtils:custom VariableResolver works well when mapping one's properties to beans in faces-config.xml, but at times one may need to grab a bean explicitly. The FacesContextUtils class makes this easy. It is similar to WebApplicationContextUtils, except that it takes a FacesContext parameter rather than a ServletContext parameter.

```
ApplicationContext ctx =
FacesContextUtils.getWebApplicationContext(FacesContext.getCurrentInstance());
```

19. What is Java Server Faces (JSF) - Spring integration mechanism?
Spring provides a custom JavaServer Faces VariableResolver implementation that extends

the standard JavaServer Faces managed beans mechanism. When asked to resolve a variable name, the following algorithm is performed:

- Does a bean with the specified name already exist in some scope (request, session, application)? If so, return it
- Is there a standard JavaServer Faces managed bean definition for this variable name? If so, invoke it in the usual way, and return the bean that was created.
- Is there configuration information for this variable name in the Spring WebApplicationContext for this application? If so, use it to create and configure an instance, and return that instance to the caller.
- If there is no managed bean or Spring definition for this variable name, return null instead.
- BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

As a result of this algorithm, you can transparently use either JavaServer Faces or Spring facilities to create beans on demand.

20. What is Significance of JSF- Spring integration ?
Spring - JSF integration is useful when an event handler wishes to explicitly invoke the bean factory to create beans on demand, such as a bean that encapsulates the business logic to be performed when a submit button is pressed.

21. How to integrate your Struts application with Spring?
To integrate your Struts application with Spring, we have two options:

- Configure Spring to manage your Actions as beans, using the ContextLoaderPlugin, and set their dependencies in a Spring context file.
- Subclass Spring's ActionSupport classes and grab your Spring-managed beans explicitly using a getWebApplicationContext() method.

22. What are ORM's Spring supports ?
Spring supports the following ORM's :

- Hibernate
- iBatis
- JPA (Java Persistence API)
- TopLink
- JDO (Java Data Objects)
- OJB

23. What are the ways to access Hibernate using Spring ?
There are two approaches to Spring's Hibernate integration:

- Inversion of Control with a HibernateTemplate and Callback
- Extending HibernateDaoSupport and Applying an AOP Interceptor

24. How to integrate Spring and Hibernate using HibernateDaoSupport?
Spring and Hibernate can integrate using Spring's SessionFactory called LocalSessionFactory. The integration process is of 3 steps.

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

25. What are Bean scopes in Spring Framework ?
The Spring Framework supports exactly five scopes (of which three are available only if you are using a web-aware ApplicationContext). The scopes supported are listed below:

Scope	Description
singleton	Scopes a single bean definition to a single object instance per Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request; that is each and every HTTP request will have 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	Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.
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.

26. 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.

27. How the AOP used in Spring?
*AOP is used in the Spring Framework:*To provide declarative enterprise services, especially as a replacement for EJB declarative services. The most important such service is declarative transaction management, which builds on the Spring Framework's transaction abstraction.To allow users to implement custom aspects, complementing their use of OOP with AOP.

28. What do you mean by Aspect ?
A modularization of a concern that cuts across multiple objects. Transaction management is a good example of a crosscutting concern in J2EE applications. In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the @Aspect annotation (@AspectJ style).

29. What do you mean by JointPoint?
A point during the execution of a program, such as the execution of a method or the handling of an exception. In Spring AOP, a join point always represents a method execution.

30. What do you mean by Advice?
Action taken by an aspect at a particular join point. Different types of advice include "around," "before" and "after" advice. Many AOP frameworks, including Spring, model an advice as an interceptor, maintaining a chain of interceptors "around" the join point.

31. What are the types of Advice?
Types of advice:

- *Before advice*: Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).
- *After returning advice*: Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
- *After throwing advice*: Advice to be executed if a method exits by throwing an exception.
- *After (finally) advice*: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
- *Around advice*: Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception

32. What are the types of the transaction management Spring supports ?
Spring Framework supports:

- Programmatic transaction management.
- Declarative transaction management.

33. 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 a number of complex transaction APIs such as JTA.
- Integrates very well with Spring's various data access abstractions.

34. Why most users of the Spring Framework choose declarative transaction management ?
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.

35. Explain the similarities and differences between EJB CMT and the Spring Framework's declarative transaction management ?
The basic approach is similar: it is possible to specify transaction behavior (or lack of it) down to individual method level. It is

possible to make a setRollbackOnly() call within a transaction context if necessary. The differences are:

- Unlike EJB CMT, which is tied to JTA, the Spring Framework's declarative transaction management works in any environment. It can work with JDBC, JDO, Hibernate or other transactions under the covers, with configuration changes only.
- The Spring Framework enables declarative transaction management to be applied to any class, not merely special classes such as EJBs.
- The Spring Framework offers declarative rollback rules: this is a feature with no EJB equivalent. Both programmatic and declarative support for rollback rules is provided.
- The Spring Framework gives you an opportunity to customize transactional behavior, using AOP. With EJB CMT, you have no way to influence the container's transaction management other than setRollbackOnly().
- The Spring Framework does not support propagation of transaction contexts across remote calls, as do high-end application servers.

36. What are object/relational mapping integration module?

Spring also supports for using of an object/relational mapping (ORM) tool over straight JDBC by providing the ORM module. Spring provide support to tie into several popular *ORM frameworks*, including *Hibernate*, *JDO*, and *iBATIS SQL Maps*. Spring’s transaction management supports each of these *ORM frameworks* as well as *JDBC*.

37. When to use programmatic and declarative transaction management ?
Programmatic transaction management is usually a good idea only if you have a small number of transactional operations.
On the other hand, if your application has numerous transactional operations, declarative transaction management is usually worthwhile. It keeps transaction management out of business logic, and is not difficult to configure.

38. 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 fairly easily and it also allows one to code without worrying about catching exceptions that are specific to each technology.

39. What are the exceptions thrown by the Spring DAO classes ?
Spring DAO classes throw exceptions which are subclasses of DataAccessException(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.

40. What is SQLExceptionTranslator ?
SQLExceptionTranslator, is an interface to be implemented by classes that can translate between SQLExceptions and Spring's own data-access-strategy-agnostic org.springframework.dao.DataAccessException.

41. What is Spring's JdbcTemplate ?
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);

42. What is PreparedStatementCreator ?
PreparedStatementCreator:

- Is one of the most common used interfaces for writing data to database.
- Has one method – createPreparedStatement(Connection)
- Responsible for creating a PreparedStatement.
- Does not need to handle SQLExceptions.

43. What is SQLProvider ?
SQLProvider:

- Has one method – getSql()
- Typically implemented by PreparedStatementCreator implementers.
- Useful for debugging.

44. What is RowCallbackHandler ?
The RowCallbackHandler interface extracts values from each row of a ResultSet.

- Has one method – processRow(ResultSet)
- Called for each row in ResultSet.
- Typically stateful.

45. What are the differences between EJB and Spring ?

Spring and EJB feature comparison.

Feature	EJB	Spring
Transaction management	• Must use a JTA transaction manager. • Supports transactions that span remote method calls.	• Supports multiple transaction environments through its PlatformTransactionManager interface, including JTA, Hibernate, JDO, and JDBC. • Does not natively support distributed transactions—it must be used with a JTA transaction manager.
Declarative transaction support	• Can define transactions declaratively through the deployment descriptor. • Can define transaction behavior per method or per class by using the wildcard character *. • Cannot declaratively define rollback	• Can define transactions declaratively through the Spring configuration file or through class metadata. • Can define which methods to apply transaction behavior explicitly or by using regular expressions. • Can declaratively define rollback behavior per method and per exception type.

	behavior—this must be done programmatically.	
Persistence	Supports programmatic bean-managed persistence and declarative container managed persistence.	Provides a framework for integrating with several persistence technologies, including JDBC, Hibernate, JDO, and iBATIS.
Declarative security	• Supports declarative security through users and roles. The management and implementation of users and roles is container specific. • Declarative security is configured in the deployment descriptor.	• No security implementation out-of-the box. • Acegi, an open source security framework built on top of Spring, provides declarative security through the Spring configuration file or class metadata.
Distributed computing	Provides container-managed remote method calls.	Provides proxying for remote calls via RMI, JAX-RPC, and web services.

46 . Do I need any other SOAP framework to run Spring Web Services?

You don't need any other SOAP framework to use Spring Web services, though it can use

some of the features of Axis 1 and 2.

47 . I get NAMESPACE_ERR exceptions when using Spring-WS. What can I do about it?
If you get the following Exception:
NAMESPACE_ERR: An attempt is made to create or change an object in a way which is incorrect with regard to namespaces.

Most often, this exception is related to an older version of Xalan being used. Make sure to upgrade to 2.7.0.

48 .Does Spring-WS run under Java 1.3?
Spring Web Services requires Java 1.4 or higher.

49. Does Spring-WS work under Java 1.4?
Spring Web Services works under Java 1.4, but it requires some effort to make it work. Java 1.4 is bundled with the older XML parser Crimson, which does not handle namespaces correctly. Additionally, it is bundled with an older version of Xalan, which also has problems. Unfortunately, placing newer versions of these on the class path does not override them. . The only solution that works is to add newer versions of Xerces and Xalan in the lib/endorsed directory of your JDK, as explained in those FAQs (i.e.$JAVA_HOME/lib/endorsed). The following libraries are known to work with Java 1.4.2:

Version

Library	
Xerces	2.8.1
Xalan	2.7.0
XML-APIs	1.3.04
SAAJ	1.2

If you want to use WS-Security, note that the XwsSecurityInterceptor requires Java 5, because an underlying library (XWSS) requires it. Instead, you can use the Wss4jSecurityInterceptor.

50 .Does Spring-WS work under Java 1.6?
Java 1.6 ships with SAAJ 1.3, JAXB 2.0, and JAXP 1.4 (a custom version of Xerces and Xalan). Overriding these libraries by putting different version on the classpath will result in various classloading issues, or exceptions in org.apache.xml.serializer.ToXMLSAXHandler. The only option for using more recent versions is to put the newer version in the endorsed directory (see above).

51 . Why do the Spring-WS unit tests fail under Mac OS X?
For some reason, Apple decided to include a Java 1.4 compatibility jar with their JDK 1.5. This jar includes the XML parsers which were included in Java 1.4. No other JDK distribution does this, so it is unclear what the purpose of this compatibility jar is.
The jar can be found at /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Classes/.compatibility/14compatibility.jar. You can safely remove or rename it, and the tests will run again.

52 . What is SAAJ?
SAAJ is the SOAP with Attachments API for Java. Like most Java EE libraries, it consists of a set of interfaces (saaj-api.jar), and implementations (saaj-impl.jar). When running in a Application Server, the implementation is typically provided by the application server. Previously, SAAJ has been part of JAXM, but it has been released as a seperate API as part of the , and also as part of J2EE 1.4. SAAJ is generally known as the packagejavax.xml.soap.
Spring-WS uses this standard SAAJ library to create representations of SOAP messages. Alternatively, it can use

53 . What version of SAAJ does my application server support?

Application Server	SAAJ Version
BEA WebLogic 8	1.1
BEA WebLogic 9	1.1/1.2*
BEA WebLogic 10	1.3**
IBM WebSphere 6	1.2
SUN Glassfish 1	1.3
JBoss 4.2	1.3***

54 .I get a NoSuchMethodError when using SAAJ. What can I do about it?
If you get the following stack trace:
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'org.springframework.ws.soap.saaj.SaajSoapMessageFactory' defined in ServletContext resource [/WEB-INF/springws-servlet.xml]:
Invocation of init method failed;

```
nested exception is java.lang.NoSuchMethodError:
javax.xml.soap.MessageFactory.newInstance(Ljava/lang/String;)Ljavax/xml/soap/MessageF
actory;
Caused by:
java.lang.NoSuchMethodError:
javax.xml.soap.MessageFactory.newInstance(Ljava/lang/String;)Ljavax/xml/soap/MessageF
actory;
```

Like most J2EE libraries, SAAJ consists of two parts: the API that consists of interfaces (saaj-api.jar) and the implementation (saaj-impl.jar). The stack trace is due to the fact that you are using a new version of the API (SAAJ 1.3), while your application server provides an earlier version of the implementation (SAAJ 1.2 or even 1.1). Spring-WS supports all three versions of SAAJ (1.1 through 1.3), but things break when it sees the 1.3 API, while there is no 1.3 implementation.
The solution therefore is quite simple: to remove the newer 1.3 version of the API, from the class path, and replace it with the version supported by your application server.

55 . I get an UnsupportedOperationException "This class does not support SAAJ 1.1" when I use SAAJ under WebLogic 9. What can I do about it?
WebLogic 9 has a known bug in the SAAJ 1.2 implementation: it implement all the 1.2 interfaces, but throws UnsupportedOperationExceptions when you call them. Confusingly, the exception message is This class does not support SAAJ 1.1, even though it supports SAAJ 1.1 just fine; it just doesn't support SAAJ **1.2**. See alsot
Spring-WS has a workaround for this, we basically use SAAJ 1.1 only when dealing with WebLogic 9. Unfortunately, other frameworks which depend on SAAJ, such as XWSS, do not have this workaround. These frameworks happily call SAAJ 1.2 methods, which throw this exception.
The solution is to not use BEA's version of SAAJ, but to use another implementation, like the one from Axis 1, or SUN. In you application context, use the following:

```
<bean id="messageFactory"
class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">
   <property name="messageFactory">
      <bean class="com.sun.xml.messaging.saaj.soap.MessageFactoryImpl"/>
   </property>
</bean>
```

56 . I get an UnsupportedOperationException "This class does not support SAAJ 1.1" when I use SAAJ under WebLogic 10. What can I do about it?
Weblogic 10 ships with two SAAJ implementations. By default the buggy 9.x implementation is used (which lives in the package weblogic.webservice.core.soap), but there is a new implementation, which supports SAAJ 1.3 (which lives in the package weblogic.xml.saaj). By looking at the DEBUG logging when Spring Web Services starts up, you can see which SAAJ implementation is used.
To use this new version, you have to create a message factory bean like so:

```
<bean id="messageFactory"
class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">
   <property name="messageFactory">
      <bean class="weblogic.xml.saaj.MessageFactoryImpl"/>
   </property>
</bean>
```

57 . I get an IndexOutOfBoundsException when I use SAAJ under JBoss. What can I do about it?
The SAAJ implementation provided by JBoss has some issues. The solution is therefore not to use the JBoss implementation, but to use another implementation. For instance, you can use SUN's reference implementation like so:

```
<bean id="messageFactory"
class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">
   <property name="messageFactory">
      <bean
class="com.sun.xml.messaging.saaj.soap.ver1_1.SOAPMessageFactory1_1Impl"/>
   </property>
</bean>
```

58 . Does Spring-WS run on IBM WebSphere?
WebSphere bundles some libraries which are out-of-date, and need to be upgraded with more recent versions. Specifically, this includes XML-apis, Xerces, Xalan, and WSDL4J. There are a couple of ways to upgrade these libraries, all using parent-last or application-first classloading.

- Package the libraries as part of the WAR (in WEB-INF/lib), and run the web application with the parent-last (application-first) classloading.
- Package the libraries as part of the EAR, add class-path entries to the manifest of the web application, and run the entire application with the parent-last classloading.
- Create a new classloader in the WebSphere console, and associate the libraries with. Set this classloader to parent-last.

The last approach has the advantage of restricting the parent-last classloading to the conflicting libraries only, and not to the entire application.

59. Why does Spring-WS only support contract-first?
Note that Spring-WS only requires you to write the XSD; the WSDL can be generated from that.

60 . How do I retrieve the WSDL from a Service?
The &WSDL query parameter does not work.
The &WSDL query parameter is a way to get a WSDL of a class. In SWS, a service is generally not implemented as a single class, but as a collection of endpoints.
There are two ways to expose a WSDL:

- Simply add the WSDL to the root of the WAR, and the file is served normally. This has the disadvantage that the "location" attribute in the WSDL is static, i.e. it does not necessarily reflect the host name of the server. You can transform locations by using a WsdlDefinitionHandlerAdapter.
- Use theMessageDispatcherServlet, which is done is the samples. Every WsdlDefinition listed in the *-servlet.xml will be exposed under the bean name. So if you define a WsdlDefinition namedecho, it will be exposed as echo.wsdl (i.e.http://localhost:8080/echo/echo.wsdl).

61 What is web module?

Spring comes with a full-featured MVC framework for building web applications. Although Spring can easily be integrated with other MVC frameworks, such as Struts, Spring's MVC framework uses IoC to provide for a clean separation of controller logic from business objects. It also allows you to declaratively bind request parameters to your business objects. It also can take advantage of any of Spring's other services, such as I18N messaging and validation.

62 What is a BeanFactory?

A BeanFactory is an implementation of the factory pattern that applies Inversion of Control to separate the application's configuration and dependencies from the actual application code.

63 What is AOP Alliance?

AOP Alliance is an open-source project whose goal is to promote adoption of AOP and interoperability among different AOP implementations by defining a common set of interfaces and components.

64 What is Spring configuration file?

Spring configuration file is an XML file. This file contains the classes information and describes how these classes are configured and introduced to each other.

65 .What does a simple spring application contain?

These applications are like any Java application. They are made up of several classes, each performing a specific purpose within the application. But these classes are configured and introduced to each other through an XML file. This XML file describes how to configure the classes, known as theSpring configuration file.

66 What is XMLBeanFactory?
BeanFactory has many implementations in Spring. But one of the most useful one is ***org.springframework.beans.factory.xml.XmlBeanFactory***, which loads its beans based on the definitions contained in an XML file. To create an ***XmlBeanFactory***, pass a java.io.InputStream to the constructor. The ***InputStream*** will provide the XML to the factory. For example, the following code snippet uses a java.io.***FileInputStream*** to provide a bean definition XML file to ***XmlBeanFactory***.

```
        BeanFactory factory = new XmlBeanFactory(new FileInputStream("beans.xml"));
```
To retrieve the bean from a BeanFactory, call the getBean() method by passing the name of the bean you want to retrieve.

```
        MyBean myBean = (MyBean) factory.getBean("myBean")
```

67 . What are important ApplicationContext implementations in spring framework?

- **ClassPathXmlApplicationContext –** This context loads a context definition from an XML file located in the class path, treating context definition files as class path resources.

- **FileSystemXmlApplicationContext –** This context loads a context definition from an XML file in the filesystem.
- **XmlWebApplicationContext –** This context loads the context definitions from an XML file contained within a web application.

68 . Explain Bean lifecycle in Spring framework?

- The spring container finds the bean's definition from the XML file and instantiates the bean.
- Using the dependency injection, spring populates all of the properties as specified in the bean definition.
- If the bean implements the **BeanNameAware** interface, the factory calls **setBeanName()** passing the bean's ID.
- If the bean implements the **BeanFactoryAware** interface, the factory calls **setBeanFactory()**, passing an instance of itself.
- If there are any **BeanPostProcessors** associated with the bean, their **post-ProcessBeforeInitialization()** methods will be called.
- If an init-method is specified for the bean, it will be called.
- Finally, if there are any **BeanPostProcessors** associated with the bean, their **postProcessAfterInitialization()** methods will be called.

69 What is bean wiring?
Combining together beans within the Spring container is known as bean wiring or wiring. When wiring beans, you should tell the container what beans are needed and how the container should use dependency injection to tie them together.

70 How do add a bean in spring application?

```
	<?xml version="1.0" encoding="UTF-8"?>
	<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
	<beans>
		<bean id="foo" class="com.act.Foo"/>
		<bean id="bar" class="com.act.Bar"/>
	</beans>
```

71. What are singleton beans and how can you create prototype beans?

Beans defined in spring framework are singleton beans. There is an attribute in bean tag named 'singleton' if specified true then bean becomes singleton and if set to false then the bean becomes a prototype bean. By default it is set to true. So, all the beans in spring framework are by default singleton beans.

```
	<beans>
	  <bean id="bar" class="com.act.Foo" singleton="false"/>
	</beans>
```

72 . What are the important beans lifecycle methods?

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.

73 . 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" class="com.act.Foo" init-method=”fooSetup”
destroy=”fooTeardown”/>
      </beans>
```

74 . What are Inner Beans?

When wiring beans, if a bean element is embedded to a property tag directly, then that bean is said to the Inner Bean. The drawback of this bean is that it cannot be reused anywhere else.

75. What are the different types of bean injections?

There are two types of bean injections.

- By setter
- By constructor

76. What is Auto wiring?

You can wire the beans as you wish. But spring framework also does this work for you. It can auto wire the related beans together. All you have to do is just set the autowire attribute of bean tag to an autowire type.

```
      <beans>
        <bean id="bar" class="com.act.Foo" Autowire=”autowire type”/>
      </beans>
```

77 . What are different types of Autowire types?

There are four different types by which autowiring can be done.

  - byName
  - byType
  - constructor
  - autodetect

78. What are the different types of events related to Listeners?

There are a lot of events related to ApplicationContext of spring framework. All the events are subclasses of org.springframework.context.Application-Event. They are

- ContextClosedEvent – This is fired when the context is closed.
- ContextRefreshedEvent – This is fired when the context is initialized or refreshed.
- RequestHandledEvent – This is fired when the web context handles any request.

79. What is an Aspect?

An aspect is the cross-cutting functionality that you are implementing. It is the aspect of your application you are modularizing. An example of an aspect is logging. Logging is something that is required throughout an application. However, because applications tend to be broken down into layers based on functionality, reusing a logging module through inheritance does not make sense. However, you can create a logging aspect and apply it throughout your application using AOP.

80 . What is a Jointpoint?

A joinpoint is a point in the execution of the application where an aspect can be plugged in. This point could be a method being called, an exception being thrown, or even a field being modified. These are the points where your aspect's code can be inserted into the normal flow of your application to add new behavior.

81 What is an Advice?

Advice is the implementation of an aspect. It is something like telling your application of a new behavior. Generally, and advice is inserted into an application at joinpoints.

82 What is a Pointcut?

A pointcut is something that defines at what joinpoints an advice should be applied. Advices can be applied at any joinpoint that is supported by the AOP framework. These Pointcuts allow you to specify where theadvice can be applied.

83 What is an Introduction in AOP?

An introduction allows the user to add new methods or attributes to an existing class. This can then be introduced to an existing class without having to change the structure of the class, but give them the new behavior and state.

84 What is a Target?

A target is the class that is being advised. The class can be a third party class or your own class to which you want to add your own custom behavior. By using the concepts of AOP, the target class is free to center on its major concern, unaware to any advice that is being applied.

85 . What is a Proxy?

A proxy is an object that is created after applying advice to a target object. When you think of client objects the target object and the proxy object are the same.

86 .What is meant by Weaving?

The process of applying aspects to a target object to create a new proxy object is called as Weaving. The aspects are woven intothe target object at the specified joinpoints.

87 What are the different points where weaving can be applied?

- Compile Time
- Classload Time
- Runtime

88 . What are the different advice types in spring?

- Around : Intercepts the calls to the target method
- Before : This is called before the target method is invoked
- After : This is called after the target method is returned
- Throws : This is called when the target method throws and exception
- Around : org.aopalliance.intercept.MethodInterceptor
- Before : org.springframework.aop.BeforeAdvice
- After : org.springframework.aop.AfterReturningAdvice
- Throws : org.springframework.aop.ThrowsAdvice

89 What are the different types of AutoProxying?

- BeanNameAutoProxyCreator
- DefaultAdvisorAutoProxyCreator
- Metadata autoproxying
- **90 What kind of exceptions those spring DAO classes throw?**
- The springâ€™s DAO class does not throw any technology related exceptions such as SQLException. They throw exceptions which are subclasses of DataAccessException.
- **91 What is DataAccessException?**
- DataAccessException is a RuntimeException. This is an Unchecked Exception. The user is not forced to handle these kinds of exceptions.
- **92 How can you configure a bean to get DataSource from JNDI?**
-
```
        <bean id="dataSource"
class="org.springframework.jndi.JndiObjectFactoryBean">
            <property name="jndiName">
              <value>java:comp/env/jdbc/myDatasource</value>
            </property>
        </bean>
```
- **93 How can you create a DataSource connection pool?**
-
```
        <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
           <property name="driver">
              <value>${db.driver}</value>
           </property>
           <property name="url">
             <value>${db.url}</value>
           </property>
           <property name="username">
             <value>${db.username}</value>
           </property>
           <property name="password">
```

```
 <value>${db.password}</value>
 </property>
 </bean>
```

- **94 How JDBC can be used more efficiently in spring framework?**
- JDBC can be used more efficiently with the help of a template class provided by spring framework called as JdbcTemplate.
- **95 How JdbcTemplate can be used?**
- With use of Spring JDBC framework the burden of resource management and error handling is reduced a lot. So it leaves developers to write the statements and queries to get the data to and from the database.
-
```
JdbcTemplate template = new JdbcTemplate(myDataSource);
```
A simple DAO class looks like this.
-
```
public class StudentDaoJdbc implements StudentDao {
  private JdbcTemplate jdbcTemplate;
```
-
```
public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
   this.jdbcTemplate = jdbcTemplate;
}
more..
}
```
The configuration is shown below.
-
```
<bean id="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
   <property name="dataSource">
      <ref bean="dataSource"/>
   </property>
</bean>
<bean id="studentDao" class="StudentDaoJdbc">
   <property name="jdbcTemplate">
      <ref bean="jdbcTemplate"/>
   </property>
</bean>
<bean id="courseDao" class="CourseDaoJdbc">
   <property name="jdbcTemplate">
      <ref bean="jdbcTemplate"/>
   </property>
</bean>
```
- **96 How do you write data to backend in spring using JdbcTemplate?**
- The JdbcTemplate uses several of these callbacks when writing data to the database. The usefulness you will find in each of these interfaces will vary. There are two simple interfaces. One is PreparedStatementCreator and the other interface is BatchPreparedStatementSetter.
- **97 Explain about PreparedStatementCreator?**
- PreparedStatementCreator is one of the most common used interfaces for writing data to database. The interface has one method createPreparedStatement().
-
```
PreparedStatement createPreparedStatement(Connection conn)
throws SQLException;
```
When this interface is implemented, we should create and return a PreparedStatement from the Connection argument, and the exception handling is automatically taken care off. When this interface is implemented, another interface SqlProvider is also implemented which has a method called getSql() which is used to provide sql strings to JdbcTemplate.
- **98 Explain about BatchPreparedStatementSetter?**

- If the user what to update more than one row at a shot then he can go for BatchPreparedStatementSetter. This interface provides two methods

-
  ```
  setValues(PreparedStatement ps, int i) throws SQLException;

  int getBatchSize();
  ```
  The getBatchSize() tells the JdbcTemplate class how many statements to create. And this also determines how many times setValues() will be called.
- **99. Explain about RowCallbackHandler and why it is used?**
- In order to navigate through the records we generally go for ResultSet. But spring provides an interface that handles this entire burden and leaves the user to decide what to do with each row. The interface provided by spring is RowCallbackHandler. There is a method processRow() which needs to be implemented so that it is applicable for each and everyrow.

-
  ```
  void processRow(java.sql.ResultSet rs);
  ```
-
- **100 What is JDBC abstraction and DAO module?**
- Using this module we can keep up the database code clean and simple, and prevent problems that result from a failure to close database resources. A new layer of meaningful exceptions on top of the error messages given by several database servers is bought in this module. In addition, this module uses Spring’s AOP module to provide transaction management services for objects in a Spring application.
-