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.

New to Spring ? Check Spring tutorial

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.

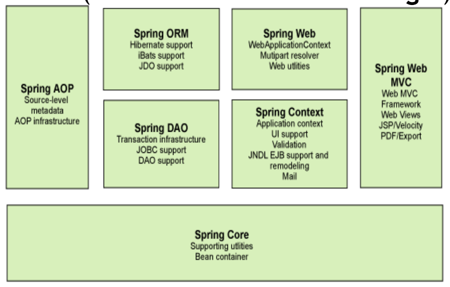
- **Transaction Management:**

  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

(Roll over to view the Image)



7. How many modules are there in Spring? What are they?

   Spring comprises of seven modules. They are..

- **The core container:**

  The core container provides the essential functionality of the Spring framework. A primary component of the core container is the BeanFactory, an implementation of the Factory pattern. The BeanFactory applies the *Inversion of Control* (IOC) pattern to separate an application's configuration and dependency specification from the actual application code.

- **Spring context:**

  The Spring context is a configuration file that provides context information to the Spring framework. The Spring context includes enterprise services such as JNDI, EJB, e-mail, internalization, validation, and scheduling functionality.

- **Spring AOP:**

  The Spring AOP module integrates aspect-oriented programming functionality directly into the Spring framework, through its configuration management feature. As a result you can easily AOP-enable any object managed by the Spring framework. The Spring AOP module provides transaction management services for objects in any Springbased application. With Spring AOP you can incorporate declarative transaction management into your applications without relying on EJB components.

- **Spring DAO:**

  The Spring JDBC DAO abstraction layer offers a meaningful exception hierarchy for managing the exception handling and error messages thrown by different database vendors. The exception hierarchy simplifies error handling and greatly reduces the amount of exception code you need to write, such as opening and closing connections. Spring DAO's JDBC-oriented exceptions comply to its generic DAO exception hierarchy.

- **Spring ORM:**

  The Spring framework plugs into several ORM frameworks to provide its Object Relational tool, including JDO, Hibernate, and iBatis SQL Maps. All of these comply to Spring's generic transaction and DAO exception hierarchies.

- **Spring Web module:**

  The Web context module builds on top of the application context module, providing 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.

- **Spring MVC framework:**

  The Model-View-Controller (MVC) framework is a full-featured MVC implementation for building Web applications. The MVC framework is highly configurable via strategy interfaces and accommodates numerous view technologies including JSP, Velocity, Tiles, iText, and POI.

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.

Are you a Spring Certified developer ? Spring Certification

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());

```

More about Spring-JSF integration » »

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.

Scope	Description
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.

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 behavior—this must be done programmatically.	• 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.

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.