**1. 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.

**2. 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.

**3. 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

**4. 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 Spring-based 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.

**5. 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.

**6. 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

**7. 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.

**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 bean scopes does Spring support? Explain them.**

The Spring Framework supports following five scopes, three of which are available only if the users use a web-aware Application Context.

* Singleton: This scopes the bean definition to a single instance per Spring IoC container.
* Prototype: This scopes a single bean definition to have any number of object instances.
* Request: This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext
* Session: This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext. Global-session: This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

**13. Explain Bean lifecycle in Spring framework?**

Following is sequence of a bean lifecycle in Spring:

* Instantiate: First the spring container finds the bean’s definition from the XML file and instantiates the bean.
* Populate properties: Using the dependency injection, spring populates all of the properties as specified in the bean definition.
* Set Bean Name: If the bean implements BeanNameAware interface, spring passes the bean’s id to setBeanName() method.
* Set Bean factory: If Bean implements BeanFactoryAware interface, spring passes the beanfactory to setBeanFactory() method.
* Pre Initialization: Also called post process of bean. If there are any bean BeanPostProcessors associated with the bean, Spring calls postProcesserBeforeInitialization() method.
* Initialize beans: If the bean implements IntializingBean,its afterPropertySet() method is called. If the bean has init method declaration, the specified initialization method is called.
* Post Initialization: – If there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization() methods will be called.
* Ready to use: Now the bean is ready to use by the application
* Destroy: If the bean implements DisposableBean , it will call the destroy() method

**14. Explain different modes of auto wiring?**

The autowiring functionality has five modes which can be used to instruct Spring container to use autowiring for dependency injection:

1. no: This is default setting. Explicit bean reference should be used for wiring.
2. byName: When autowiring byName, the Spring container looks at the properties of the beans on which autowireattribute is set to byName in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file.
3. byType: When autowiring by datatype, the Spring container looks at the properties of the beans on which autowireattribute is set to byType in the XML configuration file. It then tries to match and wire a property if its type matches with exactly one of the beans name in configuration file. If more than one such beans exist, a fatal exception is thrown.
4. constructor:This mode is similar to byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised.
5. autodetect: Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire bybyType.

**15. Are there limitations with autowiring?**

Limitations of autowiring are:

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

**16. Can you inject null and empty string values in Spring?**

Yes, you can.

**17. 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.

**18. 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.

**20. 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

**20. 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.