**Spring**

## 109. What is Spring framework?

Spring is development framework for Java programming. It is an open source development framework for Enterprise Java.
The core features of Spring Framework can be used in developing a Java Enterprise application.
It has many extensions and jars for developing web applications on top of Java EE platform.
With Spring we can develop large-scale complex Java applications very easily. It is also based on good design patterns like Dependency Injection, Aspect oriented programming for developing extensible feature rich software.

## 110. What are the benefits of Spring framework in software development?

Many benefits of Spring framework are:

Lightweight Framework: Basic Spring framework is very small in size. It is easy to use and does not add a lot of overhead on software. It just has 2 MB in basic Version.

Container: Spring framework provides the basic container that creates and manages the life cycle of application objects like Plain old Java objects (POJO). It also stores the configuration files of application objects to be created.
Dependency Injection (DI): Spring provided loose coupling is application by Dependency Injection. It uses Inversion of Control technique by which objects specify their dependencies to Spring container instead of creating new objects themselves.
Aspect Oriented Programming (AOP): Spring framework promotes and provides support for Aspect oriented programming in Java. This helps in separating application business logic from system services that are common across all the business logic. E.g. Logging can be a cross cutting concern in an Application.

Transaction Management: Spring provides a framework for transaction management. So a developer does not have to implement it from scratch. Spring Transaction Management is so powerful that we can scale it from one local transaction to global transactions in a cluster.

MVC Framework: For Web applications, Spring provides MVC framework. This framework is based on MVC design pattern and
has better features compared to other web frameworks.

Exception Handling: Spring also gives support for a common API to handle exceptions in various technologies like- Hibernate, JDBC etc.

### 111. What are the modules in Core Container of Spring framework?

Spring framework has a Core Container. Modules in Core Container are:

Core module
Bean module
Context module
Spring Expression Language module

### 112. What are the modules in Data Access/Integration layer of Spring framework?

Modules in Data Access/Integration Layer of Spring framework are:

JDBC module: An abstraction layer to remove tedious JDBC coding.
ORM module Integration layers for Object Relational Mapping OXM module: An abstraction layer to support Object XML mapping.
Java Messaging Service (JMS) module: Module for producing and consuming messages.
Transactions module: Transaction Management for POJO classes

### 113. What are the modules in Web layer of Spring framework?

Modules in Web Layer of Spring framework are:

Web module: This provides basic web-oriented integration features.
Servlet module: Support for Servlet Listeners.
WebSocket module: Support for Web Socket style messaging.
Portlet module: MVC implementation for Portlet environment.

### 114. What is the main use of Core Container module in Spring framework?

As the name suggests, Spring Core Container is the core of Spring framework. It gives the basic functionality of the Spring. All the parts of Spring Framework are built on top of Core Container.

Its main use is to provide Dependency Injection (DI) and Inversion of control (IOC) features.

**115. What kind of testing can be done in Spring Test Module?**

Spring Test Module provides support for Unit testing as well as Integration testing of Spring components. It allows using JUnit or TestNG testing frameworks. It also gives ability to mock objects to use the test code.

**116. What is the use of BeanFactory in Spring framework?**

BeanFactory is the main class that helps in implementing Inversion of Control pattern in Spring. It is based on the factory design pattern. It separates the configuration and dependencies of an application from the rest of application code.

Implementations of BeanFactory like XmlBeanFactory class are used by applications built with Spring.

**117. Which is the most popular implementation of BeanFactory in Spring?**

XMLBeanFactory is the most popular implementation of BeanFactory in Spring.

**118. What is XMLBeanFactory in Spring framework?**

XMLBeanFactory is one of the most useful implementation of BeanFactory in Spring. This factory loads its beans based on the definitions mentioned in an XML file.

Spring container reads bean configuration metadata from an XML file and creates a fully configured application with the help of XMLBeanFactory class.

**119. What are the uses of AOP module in Spring framework?**

AOP module is also known as Aspect Oriented Programming module. Its uses are:
Development of aspects in a Spring based application
Provides interoperability between Spring and other AOP frameworks
Supports metadata programming to Spring

**120. What are the benefits of JDBC abstraction layer module in Spring framework?**

Spring provides JDBC abstraction layer module. Main benefits of this module are:

Helps in keeping the database code clean and simple.
Prevents problems that result from a failure to close database resources.
Provides a layer of useful exceptions on top of the error messages given by different database servers. Based on Spring's AOP module
Provides transaction management services for objects in a Spring application

**121. How does Spring support Object Relational Mapping (ORM) integration?**

Spring supports Object Relational Mapping (ORM) by providing ORM Module. This module helps in integrating with popular ORM framework like Hibernate, JDO, and iBATIS SQL Maps etc.

Transaction Management module of Spring framework supports all of these ORM frameworks as well as JDBC.

**122. How does Web module work in Spring framework?**

Spring provides support for developing web application by using Web module. This module is built on application context module that provides context for web-based applications.

This module also supports web-oriented integration features like-transparently handling multipart requests for uploading files, programmatically binding request parameters to business objects etc.

This module also supports integration with popular web frameworks like Jakarta Struts, JSF, and Tapestry etc.

**123. What are the main uses of Spring MVC module?**

Spring-webmvc module is also known as Web-servlet module. It is based on Web Model View Controller pattern.

Main uses of this module are:

Integration of Spring with other MVC frameworks
Supports IoC to provide clean separation of controller logic from business objects
Provides clean separation between domain model code and web forms
Allows developers to declaratively bind request parameters to business objects

**124. What is the purpose of Spring configuration file?**

Spring application can be configured by an XML file. This file contains information of classes and how these classes are configured and introduced to each other.

Spring IoC container uses some kind of configuration metadata. This configuration metadata represents how an application developer tells the Spring container to instantiate, configure, and assemble the objects in your application. This configuration metadata is stored in Spring configuration file.

The other ways of specifying configuration metadata are Java based configuration and Annotation based configuration.

**125. What is the purpose of Spring IoC container?**

The Spring IoC Container is responsible for:

Creating the objects
Configuring the objects
Managing dependency between objects (with dependency injection (DI))
Wiring the objects together
Managing complete lifecycle of objects

**126. What is the main benefit of Inversion of Control (IOC) principle?**

Inversion of Control (IOC) principle is the base of Spring framework. It supports dependency injection in an application. With Dependency Injection, a programmer has to write minimal code. It also makes easier to test an application.

Most important benefit is that it leads to loose coupling within objects. With loose coupling it is easier to change the application with new requirements.

**127. Does IOC containers support**
**Eager Instantiation or Lazy loading of beans?**

IOC Container in Spring supports both the approaches. Eager instantiation as well as lazy loading of beans.

## 128. What are the benefits of ApplicationContext in Spring?

ApplicationContext in Spring provides following benefits:

Bean factory methods: These are used to access application components
Load File Resources: It helps in loading file resources in a generic fashion
Publish Events: It enables publishing events to registered listeners
Internationalization Support: Ability to resolve messages to support internationalization
Parent Context: Ability to inherit from a parent context

## 129. How will you implement ApplicationContext in Spring framework?

ApplicationContext in Spring can be implemented in one of the following three ways:

FileSystemXmlApplicationContext: If we want to load the definitions of beans from an XML file then FileSystemXmlApplicationContext is used. The full path of XML bean configuration file is provided to the constructor.
ClassPathXmlApplicationContext: To loads the definitions of beans from an XML file in the CLASSPATH, we use ClassPathXmlApplicationContext. It is used for application context embedded in jars.

WebXmlApplicationContext: To provide configuration for a web application WebXmlApplicationContext is used. While the application is running, it is read only. But it can be reloaded if underlying application supports it.

### 130. Explain the difference between ApplicationContext and BeanFactory in Spring?

Main differences between ApplicationContext and BeanFactory are:

Automatic BeanPostProcessor registration: BeanFactory does not support BeanPostProcessor registration. Whereas ApplicationContext support this.
Automatic BeanFactoryPostProcessor registration: BeanFactory also does not allow Automatic BeanFactoryPostProcessor registration. Whereas ApplicationContext allows this.
MessageSource access: BeanFactory is not convenient for MessageSource access. ApplicationContext is quite convenient for MessageSource access.

ApplicationEvent: We cannot publish ApplicationEvent with BeanFactory. But ApplicationContext provides ability to publish ApplicationEvent.

### 131. Between ApplicationContext and BeanFactory which one is preferable to use in Spring?

Spring documentation recommends using ApplicationContext in almost all the cases. ApplicationContext has all the functionality of BeanFactory.

### 132. What are the main components of a typical Spring based application?

In a Spring based application, main components are:
Spring configuration XML file: This is used to configure Spring application
API Interfaces: Definition of API interfaces for functions provided by application
Implementation: Application code with implementation of APIs
Aspects: Spring Aspects implemented by application
Client: Application at client side that is used for accessing functions

### 133. Explain Dependency Injection (DI) concept in Spring framework?

Dependency Injection is a software design pattern. It is used to implement Inversion of Control (IOC) in Spring framework. As per this pattern, we do not create objects in an application by calling new. Rather, we describe how an object should be created. In this way creation of an object is not tightly coupled with another object.
A container is responsible for creating and wiring the objects. The container can call injecting code and wire the objects as per the configuration at runtime.

### 134. What are the different roles in Dependency Injection (DI)?

There are four roles in Dependency Injection:

Service object(s) to be used
Client object that depends on the service Interface that
defines how client uses services
Injector responsible for constructing services and injecting them into client

### 135. Spring framework provides what kinds of Dependency Injection mechanism?

Spring framework provides two types of Dependency Injection mechanism:

Constructor-based Dependency Injection: Spring container can invoke a class constructor with a number of arguments. This represents a dependency on other class.

Setter-based Dependency Injection: Spring container can call setter method on a bean after creating it with a no-argument constructor or no-argument static factory method to instantiate another bean.

### 136. In Spring framework, which Dependency Injection is better? Constructor-based DI or Setter-based DI?

Spring framework provides support for both Constructor-based and Setter-based Dependency Injection. There are different scenarios in which these options can be used.

It is recommended to use Constructor-based DI for mandatory dependencies. Whereas Setter-based DI is used for optional dependencies.

## 137. What are the advantages of Dependency Injection (DI)?

Dependency Injection (DI) pattern has following advantages:

Dependency Injection reduces coupling between a class and its dependencies.

With Dependency Injection (DI), we can do concurrent or independent software development. Two teams can work parallel on classes that will be used by each other.

In Dependency Injection (DI), the client can be configured in multiple ways. It needs to just work with the given interface. Rest of the implementation can be changed and configured for different features.

Dependency injection is also used to export a system's configuration details into configuration files. So we can configure same application run in different environments based on configuration. E.g. Run in Test environment, UAT environment, and Production environment.

Dependency Injection (DI) applications provide more ease and flexibility of testing. These can be tested in isolation in Unit Test.

Dependency injection (DI) isolates client from the impact of design and implementation changes. Therefore, it promotes reusability, testability and maintainability.

## 138. What are the disadvantages of Dependency Injection (DI)?

Dependency Injection (DI) pattern has following disadvantages:

Most of the time Dependency Injection forces developers to use an injection framework like Spring. This causes dependency on a framework.

With Dependency Injection, clients are dependent on the configuration data. This becomes extra task for developers when the application does not need so many custom configuration values.

Code is difficult to trace and read in Dependency Injection. DI separates behavior from construction of objects.

Dependency injection increases complexity in the linkages between classes. It may become harder to manage such complexity outside the implementation of a class.

**139. What is a Spring Bean?**

A Spring Bean is a plain old Java object (POJO) that is created and managed by a Spring container.

There can be more than one bean in a Spring application. But all these Beans are instantiated and assembled by Spring container.

Developer provides configuration metadata to Spring container for creating and managing the lifecycle of Spring Bean.

In general, a Spring Bean is singleton. Evert bean has an attribute named "singleton". If its value is true, then bean is a singleton. If its value is false then bean is a prototype bean.

By default, the value of this attribute is true. Therefore, by default all the beans in spring framework are singleton in nature.

**140. What does the definition of a Spring Bean contain?**

A Spring Bean definition contains configuration metadata for bean.
This configuration metadata is used by Spring container to:

Create the bean
Manage its lifecycle
Resolve its dependencies

**141. What are the different ways to provide configuration metadata to a Spring Container?**

Spring supports three ways to provide configuration metadata to
Spring Container:

XML based configuration: We can specify configuration data in an XML file.

Annotation-based configuration: We can use Annotations to specify configuration. This was introduced in Spring 2.5.

Java-based configuration: This is introduced from Spring 3.0. We can embed annotations like @Bean, @Import, @Configuration in Java code to specify configuration metadata.

## 142. What are the different scopes of a Bean supported by Spring?

Spring framework support seven types of scopes for a Bean. Out of these only five scopes are available for a web-aware ApplicationContext application:

singleton: This is the default scope of a bean. Under this scope, there is a single object instance of bean per Spring IoC container.

prototype: Under this scope a single bean definition can have multiple object instances.

request: In this scope, a single bean definition remains tied to the lifecycle of a single HTTP request. Each HTTP request will have its own instance of a bean for a single bean definition. It is only valid in the context of a web-aware Spring ApplicationContext.

session: Under this scope, a single bean definition is tied to the lifecycle of an HTTP Session. Each HTTP Session will have one instance of bean. It is also valid in the context of a web-aware Spring ApplicationContext.

globalSession: This scope, ties a single bean definition to the lifecycle of a global HTTP Session. It is generally valid in a Portlet context. It is also valid in the context of a web-aware Spring ApplicationContext.

application: This scope, limits a single bean definition to the lifecycle of a ServletContext. It is also valid in the context of a web-aware Spring ApplicationContext.

websocket: In this scope, a single bean definition is tied to the lifecycle of a WebSocket. It is also valid in the context of a web-aware Spring ApplicationContext.

**143. How will you define the scope of a bean in Spring?**

In configuration xml, we can specify the scope of bean in its definition. This is used by container to decide the scope of bean in Spring.

E.g. <bean id="userService" class="com.um.UserService" scope="prototype"/>

This is an example of userService bean with prototype scope.

**144. Is it safe to assume that a Singleton bean is thread safe in Spring Framework?**

No, Spring framework does not guarantee anything related to multi-threaded behavior of a singleton bean. Developer is responsible for dealing with concurrency issues and maintaining thread safety of a singleton bean.

**145. What are the design-patterns used in Spring framework?**

Spring framework uses many Design patterns. Some of these patterns are:

Singleton – By default beans defined in spring config files are singleton. These are based on Singleton pattern.

Template – This pattern is used in many classes like-JdbcTemplate, RestTemplate, JmsTemplate, JpaTemplate etc.

Dependency Injection – This pattern is the core behind the design of BeanFactory and ApplicationContext.

Proxy – Aspect Oriented Programming (AOP) heavily uses proxy design pattern.

Front Controller – DispatcherServlet in Spring is based on Front Controller pattern to ensure that incoming requests are dispatched to other controllers.

Factory pattern – To create an instance of an object, BeanFactory is used. This is based on Factory pattern.

View Helper – Spring has multiple options to separating core code from presentation in views. Like- Custom JSP tags, Velocity macros etc.

**146. What is the lifecycle of a Bean in Spring framework?**

A Bean in Spring framework goes through following phases in its lifecycle.

Initialization and creation: Spring container gets the definition of Bean from XML file and instantiates the Bean. It populates all the properties of Bean as mentioned in the bean definition.

Setting the Behavior of Bean: In case a Bean implements BeanNameAware interface, Spring uses setBeanName() method to pass the bean's id. In case a Bean implements BeanFactoryAware interface, Spring uses setBeanFactory() to pass the BeanFactory to bean.

Post Processing: Spring container uses postProcesserBeforeInitialization() method to call BeanPostProcessors associated with the bean. Spring calls afterPropertySet() method to call the specific initialization methods. In case there are any BeanPostProcessors of a bean, the postProcessAfterInitialization() method is called.

Destruction: During the destruction of a bean, if bean implements DisposableBean, Spring calls destroy() method.

**147. What are the two main groups of methods in a Bean's lifecycle?**

A Bean in Spring has two main groups of lifecycle methods.

Initialization Callbacks: Once all the necessary properties of a Bean are set by the container, Initialization Callback methods are used for performing initialization work. A developer can implement method afterPropertiesSet() for this work.

Destruction Callbacks: When the Container of a Bean is destroyed, it calls the methods in DisposableBean to do any cleanup work. There is a method called destroy() that can be used for this purpose to make Destruction Callbacks.

Recent recommendation from Spring is to not use these methods, since it can strongly couple your code to Spring code.

**148. Can we override main lifecycle methods of a Bean in Spring?**

Yes, Spring framework allows developers to override the lifecycle methods of a Bean. This is used for writing any custom behavior for Bean.

### 149. What are Inner beans in Spring?

A bean that is used as a property of another bean is known as Inner bean. It can be defined as a <bean/> element in <property/> or <constructor-arg/> tags.

It is not mandatory for an Inner bean to have id or a name. These are always anonymous.

Inner bean does not need a scope. By default it is of prototype scope.

### 150. How can we inject a Java Collection in Spring framework?

Spring promotes Dependency Injection (DI) in code. It gives support for injecting not only objects but also collection of objects.

We can inject collections like- list, set, map etc. in Spring. Following tags can be used for this purpose:

<list> : This type is used for injecting a list of values. In a <list> duplicates are allowed.

<set> : This type is used for injecting a set of values. As per set property, duplicates are not allowed.

<map> : This type is used for injecting name-value pairs in form of map. Name and value can be of any type that is allowed for a map.

<props> : This type is used to inject a collection of String based name-value. It is like a properties file.

### 151. What is Bean wiring in Spring?

A Spring container is responsible for injecting dependencies between beans. This process of connecting beans is called wiring.

Developer mentions in configuration file, the dependencies between beans. And Spring container reads these dependencies and wires the beans on creation.

**152. What is Autowiring in Spring?**

Autowiring is a feature of Spring in which container can automatically wire/connect the beans by reading the configuration file.

Developer has to just define “autowire” attribute in a bean.

Spring resolves the dependencies automatically by looking at this attribute of beans that are autowired.

**153. What are the different modes of Autowiring supported by Spring?**

There are five modes of Autowiring supported by Spring framework:

no: This is default setting for Autowiring. In this case, we use “ref” mode to mention the explicit bean that is being referred for wiring.

E.g. In this example Employee bean refers Manager bean.

```
<bean id="employee" class="com.dept.Employee"> <property
                name="manager" ref="manager" />
</bean>
<bean id="manager" class="com.dept.Manager" />
```

byName: In this case, Spring container tries to match beans by name during Autowiring. If the name of a bean is same as the name of bean referred in autowire byname, then it automatically wires it.

E.g. In following example, Manager bean is wired to Employee bean by Name.

```
<bean id="employee" class="com.dept.Employee" autowire="byName" />
<bean id="manager" class="com.dept.Manager" />
```

byType: In this case, Spring container check the properties of beans referred with attribute byType. Then it matches the type of bean and wires. If it finds more than one such bean of that type, it throws a fatal exception.

E.g. In following example, Manager bean is wired by type to

Employee bean.

<bean id="employee" class="com.dept.Employee" autowire="byType" />
<bean id="manager" class="com.dept.Manager" />

constructor: In this case, Spring container looks for byType attribute in constructor argument. It tries to find the bean with exact name. If it finds more than one bean of same name, it throws fatal exception. This case is similar to byType case.

E.g. In following example "constructor" mode is usedfor autowiring.
<bean id="employee" class="com.dept.Employee" autowire="constructor" />

<bean id="manager" class="com.dept.Manager" />

autodetect: This is an advanced mode for autowiring. In this case, by default Spring tries to find a constructor match. If it does not find constructor then it uses autowire by Type.

E.g. This is an example of autodetect Autowiring.
<bean id="employee" class="com.dept.Employee" autowire="autodetect" />
<bean id="manager" class="com.dept.Manager" />

**154. What are the cases in which Autowiring may not work in Spring framework?**

Autowiring is a great feature in Spring. It can be used in most of the cases. But there are certain scenarios in which Autowiring may not work.

Explicit wiring: Since Autowiring is done by Spring, developer does not have full control on specifying the exact class to be used. It is preferable to use Explicit wiring in case of full control over wiring.

Primitive Data types: Autowiring does not allow wiring of properties that are based on primitive data types like- int, float etc.

**155. Is it allowed to inject null or empty String values in Spring?**

Yes, Spring allows injecting null or empty String values.

**156. What is a Java-based Configuration in Spring?**

Spring allows for Java-based configuration in which a developer can specify configuration by using Java-based annotations. This feature was introduced in Spring 3.0.

You can use annotations like- @Configuration, @Bean, @Import and @DependsOn in Java classes for specifying the configuration.

**157. What is the purpose of @Configuration annotation?**

This annotation is used in a class to indicate that this is class is the primary source of bean definitions. This class can also contain inter-bean dependencies that are annotated by @Bean annotation.

**158. What is the difference between Full @Configuration and 'lite' @Beans mode?**

Spring allows for using @Bean annotation on methods that are declared in classes not annotated with @Configuration. This is known as “lite” mode. In this mode, bean methods can be declared in a @Component or a plain java class without any annotation.

In the “lite” mode, @Bean methods cannot declare inter-bean dependencies.

It is recommended that one @Bean method should not invoke another @Bean method in 'lite' mode.

Spring recommends that @Bean methods declared within @Configuration classes should be used for full configuration. This kind of full mode can prevent many bugs.

**159. In Spring framework, what is Annotation-based container configuration?**

From Spring 2.5 version it is possible to provide configuration by using annotation.

To turn this configuration on, we need to mention <context:annotation-config/> in spring XML file.

Now developer can use annotations like @Required, @Autowired, @Qualifier etc. in a class file to specify the configuration for beans. Spring container can use this information from annotation for creating and wiring the beans.

**160. How will you switch on Annotation based wiring in Spring?**

To use Annotation based wiring, we need to turn on Annotation based configuration in Spring.

By default, Annotation based configuration is switched off in Spring. To turn it is we can specify <context:annotation-config/> element in Spring config file.

Once it is turned on, we can use @Autowired annotation or @Required annotation in a Java class for wiring in Spring.

**161. What is @Autowired annotation?**

We can use @Autowired annotation to auto wire a bean on a setter method, constructor or a field. @Autowired auto wiring is done by matching the data type.

Before using @Autowired annotation we have to register AutowiredAnnotationBeanPostProcessor. This can be done by including <context:annotation-config /> in bean configuration file.

**162. What is @Required annotation?**

We use @Required annotation to a property to check whether the property has been set or not.

Spring container throws BeanInitializationException if the @Required annotated property is not set.

When we use @Required annotation, we have to register RequiredAnnotationBeanPostProcessor in Spring config file.

**163. What are the two ways to enable RequiredAnnotationBeanPostProcesso in Spring?**

RequiredAnnotationBeanPostProcessor can be enabled in two ways in Spring:

Include <context:annotation-config />
Add Spring context and <context:annotation-config /> in bean configuration file.
E.g.

```
<beans
  ...
  xmlns:context="http://www.springframework.org/schema/context"
  ...
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context-
2.5.xsd">
  ...
  <context:annotation-config />
  ...
</beans>
```

Include RequiredAnnotationBeanPostProcessor in bean configuration file

E.g.

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/be
  http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd">

<bean
class="org.springframework.beans.factory.annotation.RequiredAnno

  <bean id="BookBean" class="com.foo.Book"> <property
       name="action"      value="price"      />      <property
       name="type" value="1" />
  </bean>

  <bean id="AuthorBean" class="com.foo.Author"> <property
       name="name" value="Rowling" />
  </bean>

</beans>
```

### 164. What is @Qualifier annotation in Spring?

We use @Qualifier annotation to mark a bean as ready for auto wiring. This annotation is used along with @Autowired annotation to specify the exact bean for auto wiring by Spring container.

### 165. How Spring framework makes JDBC coding easier for developers?

Spring provides a mature JDBC framework to provide support for JDBC coding. Spring JDBC handled resource management as well as error handling in a generic way. This reduces the work of software developers.

They just have to write queries and related statements to fetch the data or to store the data in database.

### 166. What is the purpose of JdbcTemplate?

Spring framework provides JdbcTemplate class that contains many convenient methods for regular tasks like- converting data into primitives or objects, executing prepared or callable statements etc.

This class makes it very easy to work with database in our Application and it also provides good support for custom error handling in database access code.

### 167. What are the benefits of using Spring DAO?

Some of the benefits of using Spring DAO are:

It makes it easier to work on different data access methods like-JDBC, Hibernate etc.
It provides a consistent and common way to deal with different data access methods.
Spring DAO makes it easier to switch between different data persistence frameworks.
No need for catching framework specific exceptions.

### 168. What are the different ways to use Hibernate in Spring?

Spring provides two ways to use Hibernate:

We can extend HibernateDAOSupport and apply an AOP interceptor node to use Hibernate.
We can also use HibernateTemplate and Callback to access Hibernate. This is based on Inversion of Control.

### 169. What types of Object Relational Mapping (ORM) are supported by Spring?

Spring supports following Object Relational Mapping (ORM) frameworks:

Hibernate
Java Persistence API (JPA)
TopLink
Java Data Objects (JDO)
Apache Object Relational Bridge (ORB)

**170. How will you integrate Spring and Hibernate by using HibernateDaoSupport?**

We can use following steps for integrating Spring and Hibernate:

Add dependencies for Spring and Hibernate in pom.xml Implement
DAO from HibernateDaoSupport
Use Hibernate functions via getHibernateTemplate() method

**171. What are the different types of the Transaction Management supported by Spring framework?**

Spring framework provides support for two types of Transaction Management:

Programmatic: In this method, we have to manage Transaction by programming explicitly. It provides flexibility to a developer, but it is not easier to maintain.

Declarative: In this approach, we can separate Transaction Management from the Application Business code. We can use annotations or XML based configuration to manage the transactions in declarative approach.

**172. What are the benefits provided by Spring Framework's Transaction Management?**

Main benefits provided by Spring Transaction Management are:

Consistent: By using Spring Transaction management, we can use consistent programming model across different transaction APIs like- JPA, JDBC, JTA, Hibernate, JPA, JDO etc.

Simplicity: Spring TM provides simple API for managing the transaction programmatically.

Declarative: Spring also supports annotation or xml based declarative transaction management.

Integration: Spring Transaction management is easier to integrate with other data access abstractions of Spring.

**173. Given a choice between declarative and programmatic Transaction Management, which method will you choose?**

In Spring, Declarative Transaction Management is the preferred choice. This method is very less invasive and it has very less impact in Application Business Logic.

Although Declarative method gives less flexibility than Programmatic method, it is simpler to use and easier to maintain in long run.

**174. What is Aspect Oriented Programming (AOP)**

Aspect Oriented Programming (AOP) is a programming paradigm that promotes programmers to develop code in different modules that can be parallel or in crosscutting concerns.

E.g. To develop banking software, one team can work on business logic for Money withdrawal, Money deposit, Money Transfer etc. The other team can work on Transaction Management for committing the transaction across multiple accounts.

In an Auto company, one team can work on software to integrate with different components of car. The other team can work on how all the components will send signal and current information to a common dashboard.

**175. What is an Aspect in Spring?**

An Aspect is the core construct of AOP. It encapsulates the behavior that affects multiple classes in a reusable module.

An Aspect can have a group of APIs that provide cross-cutting features.

E.g. A logging module can be an Aspect in an Application.

An application can have multiple of Aspects based on the different requirements.

An Aspect can be implemented by using annotation @Aspect on a class.

### 176. In Spring AOP, what is the main difference between a Concern and a Cross cutting concern?

A Concern in Spring is the behavior or expectation from an application. It can be the main feature that we want to implement in the application.

A Cross cutting concern is also a type of Concern. It is the feature or functionality that is spread throughout the application in a thin way.

E.g. Security, Logging, Transaction Management etc. are cross cutting concerns in an application.

### 177. What is a Joinpoint in Spring AOP?

In Spring AOP, Joinpoint refers to a candidate point in application where we can plug in an Aspect.

Joinpoint can be a method or an exception or a field getting modified.

This is the place where the code of an Aspect is inserted to add new behavior in the existing execution flow.

### 178. What is an Advice in Spring AOP?

An Advice in Spring AOP, is an object containing the actual action that an Aspect introduces.

An Advice is the code of cross cutting concern that gets executed.

There are multiple types of Advice in Spring AOP.

### 179. What are the different types of Advice in Spring AOP?

Spring AOP provides five kinds of Advice:

1. Before Advice: This type of advice runs just before a method executes. We can use @Before annotation for this.

2. After (finally) Advice: This type of advice runs just after a method executes. Even if the method fails, this advice will run. We can use @After annotation here.

3. After Returning Advice: This type of advice runs after a method executes successfully. @AfterReturning annotation can be used here.

4. After Throwing Advice: This type of advice runs after a method executes and throws an exception. The annotation to be used is @AfterThrowing.

5. Around Advice: This type of advice runs before and after the method is invoked. We use @Around annotation for this.

**180. What is a Pointcut in Spring AOP?**

A Pointcut in Spring AOP refers to the group of one or more Joinpoints where an advice can be applied.

We can apply Advice to any Joinpoint. But we want to limit the places where a specific type of Advice should be applied. To achieve this we use Pointcut.

We can use class names, method names or regular expressions to specify the Pointcuts for an Advice.

**181. What is an Introduction in Spring AOP?**

In Spring AOP we can declare additional methods or fields on behalf of a type. To do this we use an Introduction. It is also known as inter-type declaration.

E.g. We can use an Introduction for making a bean implement IsModified interface.

**182. What is a Target object in Spring AOP?**

A Target object is the object that gets Advice from one or more Aspects.

This is also known as advised object.

In most cases it is a proxy object.

**183. What is a Proxy in Spring AOP?**

In Spring AOP, a Proxy is an object created by the AOP framework to implement Aspect contracts. It is generally a JDK dynamic proxy or CGLIB proxy.

### 184. What are the different types of AutoProxy creators in Spring?

Spring AOP provides following standard types of Autoproxy creators:

1. BeanNameAutoProxyCreator: This is a BeanPostProcessor that creates AOP proxies for beans automatically by matching names.

2. DefaultAdvisorAutoProxyCreator: This creator is more powerful that other Proxy Creators. This also applies eligible advisors automatically to bean in the current context.

3. AbstractAdvisorAutoProxyCreator: This is the parent class of DefaultAdvisorAutoProxyCreator. We can create our own auto-proxy creators by extending this class.

### 185. What is Weaving in Spring AOP?

In Aspect oriented programming, linking Aspects with the other application types creates an Advised object. This process is known as Weaving.

Without Weaving, we just have definition of Aspects. Weaving makes use realize full potential of the AOP.

Weaving can be done at compile time, load time or at run time.

### 186. In Spring AOP, Weaving is done at compile time or run time?

Spring container performs Weaving at run time.

### 187. What is XML Schema-based Aspect implementation?

Spring allows for implementing Aspect by using regular classes and XML based configurations. This is different from Annotation based Aspect implementation. But it achieves the same goal of AOP.

We can use elements like <aop:aspect id=”testAspect" ref="testBean" /> and <aop:pointcut id="testPointcut" /> in Spring XML config file.

To use this we need to import Spring AOP schema as follows:

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
```

```
                xsi:schemaLocation="http://www.springframework.org/schema/bea
                    http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
">
```

**188. What is Annotation-based aspect implementation in Spring AOP?**

This is a declarative style AOP implementation. In this case, we use annotations like @Aspect, @Pointcut, @Joinpoint etc. to annotate code with different types of AOP elements.

This can be used Java 5 onwards, when the support for Annotations was introduced.

**189. How does Spring MVC framework work?**

Spring provides its own Model View Controller (MVC) framework for developing web applications.

Spring MVC framework is based on Inversion of Control (IOC) principle. It separates the business objects from controller.

It is designed around the DispatcherServlet that is responsible for dispatching requests to relevant handlers.

Spring MVC framework also supports annotation based binding of request parameters.

**190. What is DispatcherServlet?**

In Spring MVC, DispatcherServlet is the core servlet that is responsible for handling all the requests and dispatching these to handlers.

Dispatcher servlet knows the mapping between the method to be called and the browser request. It calls the specific method and combines the results with the matching JSP to create an html document, and then sends it back to browser.

In case of RMI invocation, it sends back response to the client application.

### 191. Can we have more than one DispatcherServlet in Spring MVC?

Yes, a Spring MVC web application can have more than one DispatcherServlets. Each DispatcherServlet has to operate in its own namespace. It has to load its own ApplicationContext with mappings, handlers, etc.

Only the root application context will be shared among these Servlets.

### 192. What is WebApplicationContext in Spring MVC?

WebApplicationContext is the child of plain ApplicationContext. It is used in web applications. It provides features to deal with web-related components like-controllers, view resolvers etc.

A Web Application can have multiple WebApplicationContext to handle requests.

Each DispatcherServlet is associated with one WebApplicationContext.

### 193. What is Controller in Spring MVC framework?

Controller is an interface in Spring MVC. It receives HttpServletRequest and HttpServletResponse in web app just like an HttpServlet, but it is able to participate in an MVC flow.

Controllers are similar to a Struts Action in a Struts based Web application.

Spring recommends that the implementation of Controller interface should be a reusable, thread-safe class, capable of handling multiple HTTP requests throughout the lifecycle of an application.

It is preferable to implement Controller by using a JavaBean.

Controller interprets user input and transforms it into a model. The model is represented to the user by a view.

Spring implements a controller in a very generic way. This enables us to create a wide variety of controllers.
What is @Controller annotation in Spring MVC?

We use @ Controller annotation to indicate that a class is a Controller in Spring MVC.

The dispatcher in Spring scans for @Controller annotated classes for mapped methods and detects @RequestMapping.

## 194. What is @RequestMapping annotation in Spring?

In Spring MVC, we use @RequestMapping annotation to map a web request to either a class or a handler method.

In @RequestMapping we can specify the path of URL as well as HTTP method like- GET, PUT, POST etc.

@RequestMapping also supports specifying HTTP Headers as attributes.

We can also map different media types produced by a controller in @RequestMapping. We use HTTP Header Accepts for this purpose.

E.g. @RequestMapping(
  value = "/test/mapping",
  method = GET,
  headers = "Accept=application/json")

## 195. What are the main features of Spring MVC?

Spring MVC has following main features:

1. Clear separation of role: In Spring MVC, each role like- controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver etc. is fulfilled by a specialized object.

2. Reusability: Spring MVC promotes reusable business code that reduces the need for duplication. We can use existing business objects as command or form objects instead of copying them to extend a particular framework base class.

3. Flexible Model Transfer: Spring MVC Model transfer supports easy integration with other view technologies as well.

4. Customizable binding and validation: In Spring MVC, we can to custom binding between Requests and Controllers. Even validation can be done on non-String values as well.

5. JSP form tag library: From Spring 2.0, there is a powerful JSP form tag library that makes writing forms in JSP pages much easier.

6. Customizable locale, time zone and theme resolution: Spring MVC supports customization in locale, timezone etc.

**196. What is the difference between a Singleton and Prototype bean in Spring?**

Every bean in Spring has a scope that defines its existence timeframe in the application.

Singleton scope for bean limits a bean to a single object instance per Spring IOC container.

This single instance is limited to a specific ApplicationContext. If there are multiple ApplicationContext then we can have more than one instance of bean.

By default all the beans in Spring framework are Singleton scope beans.

With Prototype scope a single bean definition can have multiple object instances in a Spring container.

In prototype scope bean, the Spring IoC container creates new bean instance of the object every time a request for that specific bean is made.

**197. How will you decide which scope-Prototype or Singleton to use for a bean in Spring?**

In general, we use prototype scope for all stateful beans and singleton scope for stateless beans.

Since a stateless bean does not maintain any state, we can use the same object instance again and again. Singleton scope bean serves the same purpose.

In a stateful bean, there is a need to maintain the state in each request, it is necessary to use a new instance of object with each call. A Prototype scope bean ensures that we get a new instance each time we request for the object.

**198. What is the difference between Setter and Constructor based Dependency Injection (DI) in Spring framework?**

Main differences between Setter and Constructor based Dependency Injection (DI) in Spring are:

Priority: Setter based injection has higher priority than a constructor based injection in Spring. If an application uses Setter as well as Constructor injection, Spring container uses the Setter injection.

Partial dependency: We can inject partial dependency by using Setter injection. In Constructor injection, it is not possible to do just a partial dependency injection.

E.g. If there are two properties in a class, we can use Setter method to inject just one property in the class.

Flexibility: Setter injection gives more flexibility in introducing changes. One can easily change the value by Setter injection. In case of Constructor injection a new bean instance has to be created always.

Readability: Setter injection is more readable than Constructor injection. Generally Setter method name is similar to dependency class being used in setter method.

**199. What are the drawbacks of Setter based Dependency Injection (DI) in Spring?**

Although Setter based Dependency Injection has higher priority than Constructor based DI, there are some disadvantages of it.

No Guarantee: In Setter based DI, there is no guarantee that a certain dependency is injected or not. We may have an object with partial or no dependency. Whereas in Constructor based DI, an object in not created till the time all the dependencies are ready.

Security: One can use Setter based DI to override another dependency. This can cause Security breach in a Spring application.

Circular Dependency: Setter based DI can cause circular dependency between objects. Where as Constructor based DI will throw ObjectCurrentlyInCreationException if there is a circular dependency during the creation of an object.

**200. What are the differences between Dependency Injection (DI) and Factory Pattern?**

Main differences between Dependency Injection (DI) and Factory Pattern are:

Coupling: Factory pattern adds tight coupling between an object, factory and dependency. In case of DI, there is no coupling between objects. We just mention the dependencies on different objects and container resolves and introduces these dependencies.

Easier Testing: DI is easier to test, since we can inject the mock objects as dependency in Test environment. In case of Factory pattern, we need to create actual objects for testing.

Flexibility: DI allows for switching between different DI frameworks easily. It gives flexibility in the choice of DI framework.

Container: DI always needs a container for injecting the dependencies. This leads to extra overhead as well as extra code in your application. In factory pattern, you can just use POJO classes to implement the application without any container.

Cleaner Code: DI code is much cleaner than Factory pattern based code. In DI, we do not need to add extra code for factory methods.

**201. In Spring framework, what is the difference between FileSystemResource and ClassPathResource?**

In Spring we can specify configuration by using a file or classpath.

In FileSystemResource we have to give absolute path / relative path of Spring Configuration file spring-config.xml file.

In ClassPathResource Spring looks for Spring Configuration file spring-config.xml in ClassPath. Therefore, developer has to include spring-config.xml in classpath.

ClassPathResource looks for configuration file in CLASSPATH, whereas FileSystemResource looks for configuration file in file system.

**202. Name some popular Spring framework annotations that you use in your project?**

Spring has many Annotations to serve different purposes. For regular use we refer following popular Spring annotations:

@Controller: This annotation is for creating controller classes in a Spring MVC project.

@RequestMapping: This annotation maps the URI to a controller handler method in Spring MVC.

@ResponseBody: For sending an Object as response we use this annotation.

@PathVariable: To map dynamic values from a URI to handler method arguments, we use this annotation.

@Autowired: This annotation indicates to Spring for auto-wiring dependencies in beans.

@Service: This annotation marks the service classes in Spring.

@Scope: We can define the scope of Spring bean by this annotation.

@Configuration: This an annotation for Java based Spring configuration.

@Aspect, @Before, @After, @Around, @Joinpoint, @Pointcut:
These are the annotations in Spring for AspectJ AOP.

**203. How can you upload a file in Spring MVC Application?**

In Spring MVC framework we can use MultipartResolver interface to upload a file. We need to make configuration changes to make it work. After uploading the file, we have to create Controller handler method to process the uploaded file in application.

**204. What are the different types of events provided by Spring framework?**

Spring framework provides following five events for Context:

ContextRefreshedEvent: Whenever ApplicationContext is initialized or refreshed, Spring publishes this event. We can also
raise it by using refresh() method on ConfigurableApplicationContext interface.

ContextStartedEvent: When ApplicationContext is started using start() method on ConfigurableApplicationContext interface, ContextStartedEvent is published. We can poll database or restart any stopped application after receiving this event.

ContextStoppedEvent: Spring publishes this event when ApplicationContext is stopped using stop() method on ConfigurableApplicationContext interface. This is used for doing any cleanup work.

ContextClosedEvent: Once the ApplicationContext is closed using close() method, ContextClosedEvent is published. Once a context is closed, it is the last stage of its lifecycle. After this it cannot be refreshed or restarted.

RequestHandledEvent: This is a web specific event that informs to all beans that an HTTP request has been serviced.

**205. What is the difference between DispatcherServlet and ContextLoaderListener in Spring?**

DispatcherServlet is the core of Spring MVC application. It loads Spring bean configuration file and initialize all the beans mentioned in config file.

In case we have enabled annotations in Spring config file, it also scans the packages and configures any bean annotated with @Component, @Controller, @Repository or @Service annotations.

ContextLoaderListener is a listener to start up and shut down Spring's root WebApplicationContext. ContextLoaderListener links the lifecycle of ApplicationContext to the lifecycle of the ServletContext. It automates the creation of ApplicationContext. It can also be used to define shared beans used across different spring contexts.

**206. How will you handle exceptions in Spring MVC Framework?**

Spring MVC Framework provides following mechanisms to help us achieve exception handling:

Controller Based: A developer can define exception handler methods in a Controller class. To do so, they have to annotate the methods with @ExceptionHandler annotation.

Global Exception Handler: Spring provides @ControllerAdvice annotation for exception handling as cross-cutting concern. We can mark any class as global exception handler by using this annotation.

HandlerExceptionResolver implementation: Spring Framework provides HandlerExceptionResolver interface that can be implemented to create a global exception handler.

## 207. What are the best practices of Spring Framework?

In Spring Framework, following are some of the best practices:

We can Divide spring bean configurations based on their concerns such as spring-jdbc.xml, spring-security.xml.

It is better to avoid version numbers in schema reference. This makes sure that we have the latest config files.

It is a good practice to configure bean dependencies as much as possible. Unless there is a good reason, we try to avoid autowiring.

For spring beans that are used in multiple contexts in Spring MVC, we can create them in root context and initialize with listener.

Spring framework provides many features and modules. We should just use what we need for our application. An extra dependency has to be removed

For application properties, it is good to create a property file and read it in Spring configuration file.

Annotations are useful for smaller applications, but for larger applications annotations can become an overhead. It is easier to maintain if all the configurations are in xml files.

When we are doing AOP, we have to make sure to keep the Joinpoint as narrow as possible to avoid Advice on unwanted methods.

We should use right annotation for components or services. For services use @Service and for DAO beans use @Repository.
Dependency Injection (DI) has to be used when there is real benefit.
It should not be used just for the sake of loose coupling.

## 208. What is Spring Boot?

Spring Boot is a ready-made solution to create Spring applications with production grade features. It favors convention over configuration.
We can embed Tomcat or Jetty in in an application created with Spring Boot. Spring Boot automatically configures Spring in an application.

It does not require any code generation or xml configuration. It is an easy solution to create applications that can run stand-alone.