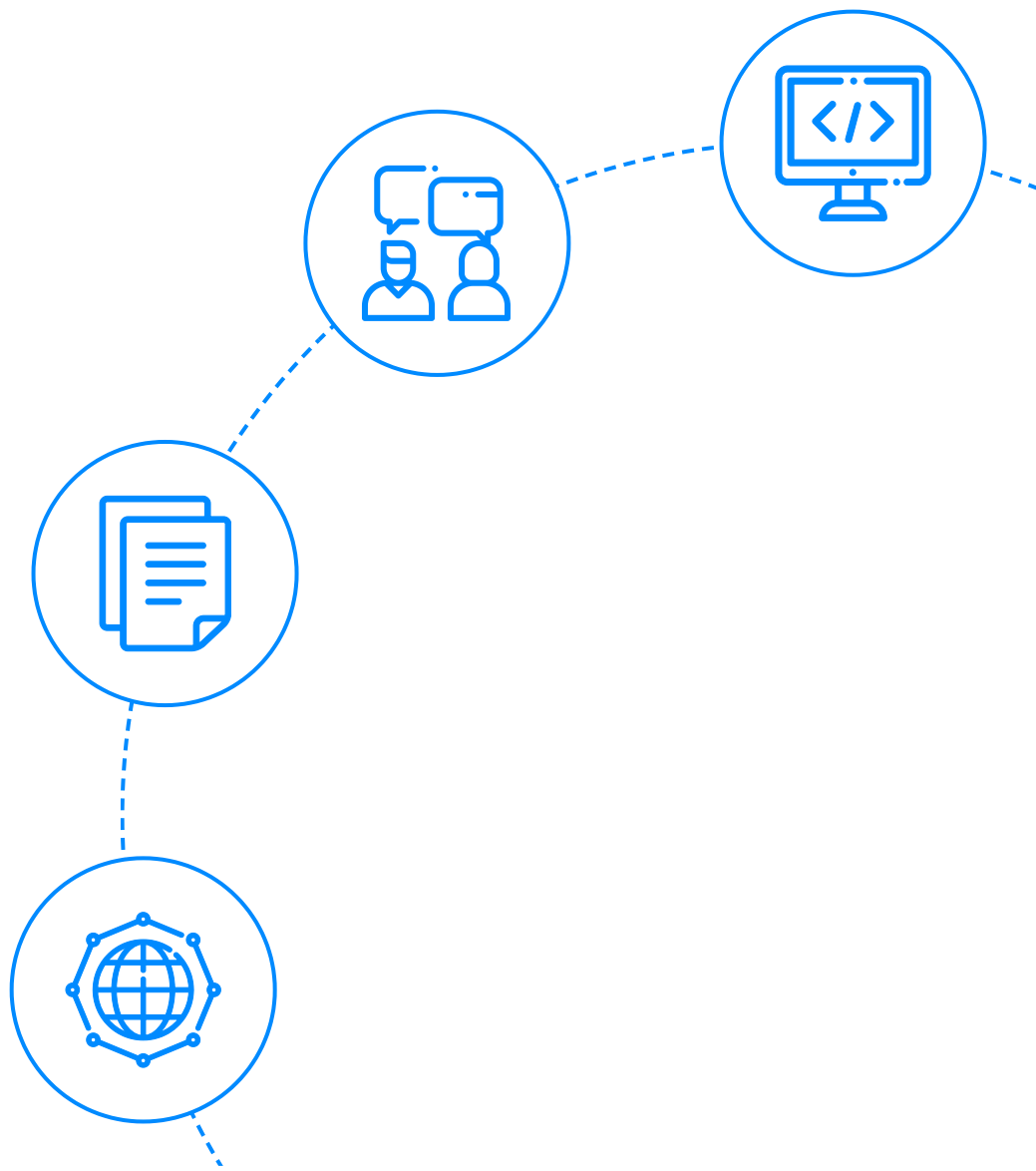




InterviewBit

Spring Interview Questions



To view the live version of the page, [click here](#).

© Copyright by Interviewbit

Contents

Spring, Spring Core, Spring IoC Interview Questions

1. What is Spring Framework?
2. What are the features of Spring Framework?
3. What is a Spring configuration file?
4. What do you mean by IoC (Inversion of Control) Container?
5. What do you understand by Dependency Injection?
6. Explain the difference between constructor and setter injection?
7. What are Spring Beans?
8. How is the configuration meta data provided to the spring container?
9. What are the bean scopes available in Spring?
10. Explain Bean life cycle in Spring Bean Factory Container.
11. What do you understand by Bean Wiring.
12. What is autowiring and name the different modes of it?
13. What are the limitations of autowiring?

Spring Boot Interview Questions

14. What do you understand by the term 'Spring Boot'?
15. Explain the advantages of using Spring Boot for application development.
16. Differentiate between Spring and Spring Boot.
17. What are the features of Spring Boot?
18. What does @SpringBootApplication annotation do internally?

Spring Boot Interview Questions

(.....Continued)

19. What are the effects of running Spring Boot Application as “Java Application”?
20. What is Spring Boot dependency management system?
21. What are the possible sources of external configuration?
22. Can we change the default port of the embedded Tomcat server in Spring boot?
23. Can you tell how to exclude any package without using the basePackages filter?
24. How to disable specific auto-configuration class?
25. Can the default web server in the Spring Boot application be disabled?
26. What are the uses of @RequestMapping and @RestController annotations in Spring Boot?

Spring AOP, Spring JDBC, Spring Hibernate Interview Questions

27. What is Spring AOP?
28. What is an advice? Explain its types in spring.
29. What is Spring AOP Proxy pattern?
30. What are some of the classes for Spring JDBC API?
31. How can you fetch records by Spring JdbcTemplate?
32. What is Hibernate ORM Framework?
33. What are the two ways of accessing Hibernate by using Spring.
34. What is Hibernate Validator Framework?
35. What is HibernateTemplate class?

Spring MVC Interview Questions

Spring MVC Interview Questions

(.....Continued)

36. What is the Spring MVC framework?
37. What are the benefits of Spring MVC framework over other MVC frameworks?
38. What is DispatcherServlet in Spring MVC?
39. What is a View Resolver pattern and explain its significance in Spring MVC?
40. What is the @Controller annotation used for?
41. Can you create a controller without using @Controller or @RestController annotations?
42. What is ContextLoaderListener and what does it do?
43. What are the differences between @RequestParam and @PathVariable annotations?
44. What is the Model in Spring MVC?
45. What is the use of @Autowired annotation?
46. What is the role of @ModelAttribute annotation?
47. What is the importance of the web.xml in Spring MVC?
48. What are the types of Spring MVC Dependency Injection?
49. What is the importance of session scope?
50. What is the importance of @Required annotation?
51. Differentiate between the @Autowired and the @Inject annotations.
52. Are singleton beans thread-safe?
53. How can you achieve thread-safety in beans?
54. What is the significance of @Repository annotation?
55. How is the dispatcher servlet instantiated?

Spring MVC Interview Questions

(.....Continued)

56. How is the root application context in Spring MVC loaded?
57. How does the Spring MVC flow look like? In other words, How does a DispatcherServlet know what Controller needs to be called when there is an incoming request to the Spring MVC?
58. Where does the access to the model from the view come from?
59. Why do we need BindingResults?
60. What are Spring Interceptors?
61. Is there any need to keep spring-mvc.jar on the classpath or is it already present as part of spring-core?
62. What are the differences between the <vs> tags?
63. How is the form data validation done in Spring Web MVC Framework?
64. How to get ServletConfig and ServletContext objects in spring bean?
65. Differentiate between a Bean Factory and an Application Context.
66. How are i18n and localization supported in Spring MVC?
67. What do you understand by MultipartResolver?
68. How is it possible to use the Tomcat JNDI DataSource in the Spring applications?
69. What will be the selection state of a checkbox input if the user first checks the checkbox and gets validation errors in other fields and then unchecks the checkbox after getting the errors?

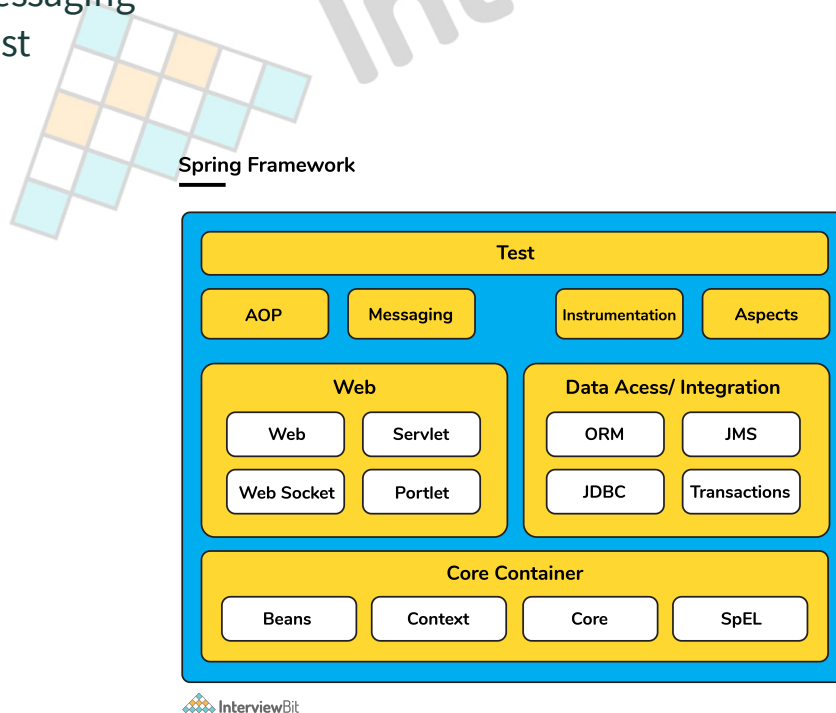
Let's get Started

The Spring Framework was first developed by Rod Johnson in 2003. It was developed to make the development of Java applications quicker, easier and safer for developers. Spring Framework is an open-source, lightweight, easy-to-build framework that can also be considered as a framework of frameworks as it houses various frameworks like Dependency Injection, Spring MVC, Spring JDBC, Spring Hibernate, Spring AOP, EJB, JSF, etc. A key element of Spring is the feature of the application's infrastructural support. Spring focuses on providing the “plumbing” of enterprise-level applications and ensures that the developers can focus purely on application-level business logic, without unnecessary ties to specific deployment environments. Applications developed in Spring are more reliable, scalable, and very simple to build and maintain. Spring was developed as means to help developers manage the business objects of the application. Due to its vast features and flexibilities, Spring became the most loved framework for developing enterprise-level Java-based applications. In the following section, we will see what are the most commonly asked interview questions and answers to prepare you for Spring-based interviews.

Spring, Spring Core, Spring IoC Interview Questions

1. What is Spring Framework?

- Spring is a powerful open-source, loosely coupled, lightweight, [java framework](#) meant for reducing the complexity of developing enterprise-level applications. This framework is also called the “framework of frameworks” as spring provides support to various other important frameworks like JSF, Hibernate, Struts, EJB, etc.
- There are around 20 modules which are generalized into the following types:
 - Core Container
 - Data Access/Integration
 - Web
 - AOP (Aspect Oriented Programming)
 - Instrumentation
 - Messaging
 - Test



Spring handles all the infrastructure-related aspects which lets the programmer to focus mostly on application development.

2. What are the features of Spring Framework?

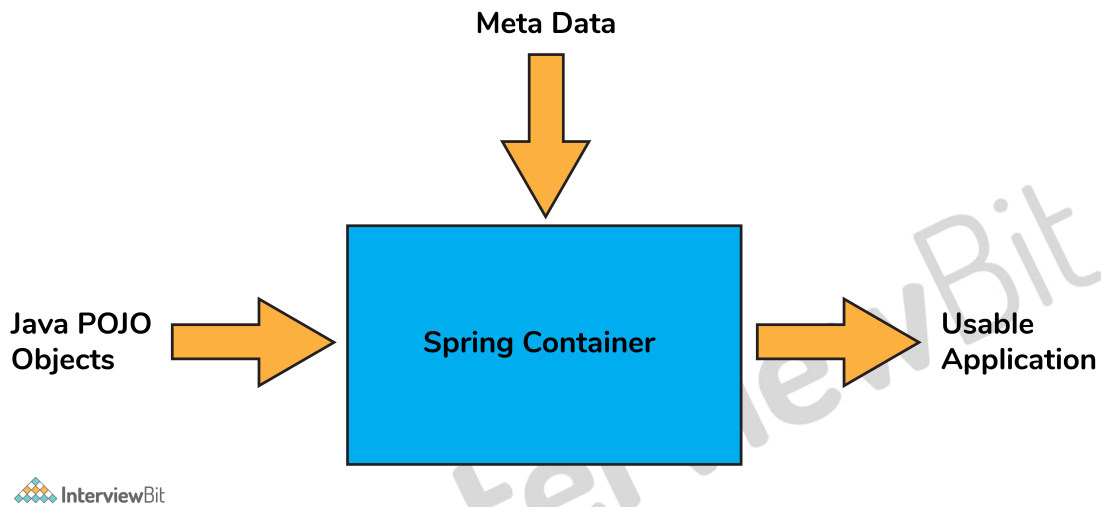
- Spring framework follows **layered architecture** pattern that helps in the necessary components selection along with providing a robust and cohesive framework for J2EE applications development.
- The AOP (Aspect Oriented Programming) part of Spring supports unified development by ensuring **separation of application's business logic** from other system services.
- Spring provides **highly configurable** MVC web application framework which has the ability to switch to other frameworks easily.
- Provides provision of **creation and management** of the configurations and defining the lifecycle of application objects.
- Spring has a special design principle which is known as IoC (**Inversion of Control**) that supports objects to give their dependencies rather than looking for creating dependent objects.
- Spring is a **lightweight, java based, loosely coupled** framework.
- Spring provides generic **abstraction layer for transaction management** that is also very useful for container-less environments.
- Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate or other frameworks) into **consistent, unchecked exceptions**. This introduces abstraction and greatly simplifies exception handling.

3. What is a Spring configuration file?

A Spring configuration file is basically an XML file that mainly contains the classes information and describes how those classes are configured and linked to each other. The XML configuration files are verbose and cleaner.

4. What do you mean by IoC (Inversion of Control) Container?

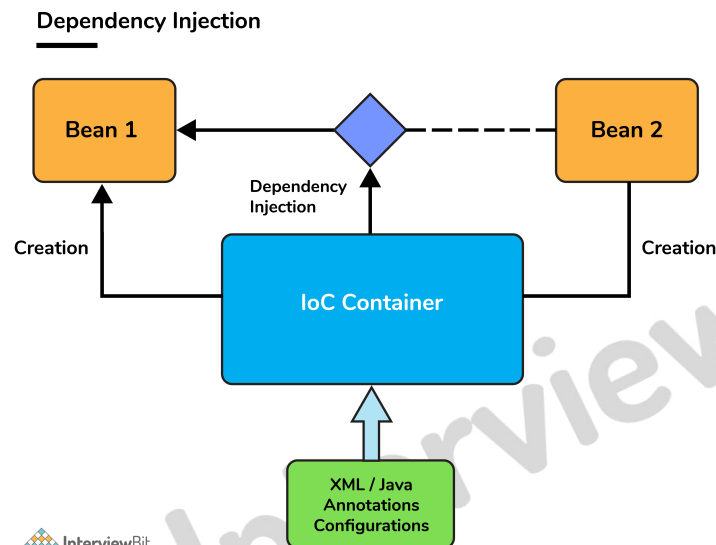
Spring container forms the core of the Spring Framework. The Spring container uses Dependency Injection (DI) for managing the application components by creating objects, wiring them together along with configuring and managing their overall life cycles. The instructions for the spring container to do the tasks can be provided either by XML configuration, Java annotations, or Java code.



5. What do you understand by Dependency Injection?

The main idea in Dependency Injection is that you don't have to create your objects but you just have to describe how they should be created.

- The components and services need not be connected by us in the code directly. We have to describe which services are needed by which components in the configuration file. The IoC container present in Spring will wire them up together.



- In Java, the 2 major ways of achieving dependency injection are:
 - Constructor injection: Here, the IoC container invokes the class constructor with a number of arguments where each argument represents a dependency on the other class.
 - Setter injection: Here, the spring container calls the setter methods on the beans after invoking a no-argument static factory method or default constructor to instantiate the bean.

6. Explain the difference between constructor and setter injection?

- In constructor injection, partial injection is not allowed whereas it is allowed in setter injection.
- The constructor injection doesn't override the setter property whereas the same is not true for setter injection.
- Constructor injection creates a new instance if any modification is done. The creation of a new instance is not possible in setter injection.
- In case the bean has many properties, then constructor injection is preferred. If it has few properties, then setter injection is preferred.

7. What are Spring Beans?

- They are the objects forming the backbone of the user's application and are managed by the Spring IoC container.
- Spring beans are instantiated, configured, wired, and managed by IoC container.
- Beans are created with the configuration metadata that the users supply to the container (by means of XML or java annotations configurations.)

8. How is the configuration meta data provided to the spring container?

There are 3 ways of providing the configuration metadata. They are as follows:

- **XML-Based configuration:** The bean configurations and their dependencies are specified in XML configuration files. This starts with a bean tag as shown below:

```
<bean id="interviewBitBean" class="org.intervuewBit.firstSpring.InterviewBitBean">
  <property name="name" value="InterviewBit"></property>
</bean>
```

- **Annotation-Based configuration:** Instead of the XML approach, the beans can be configured into the component class itself by using annotations on the relevant class, method, or field declaration.
 - Annotation wiring is not active in the Spring container by default. This has to be enabled in the Spring XML configuration file as shown below

```
<beans>
<context:annotation-config/>
<!-- bean definitions go here -->
</beans>
```

- **Java-based configuration:** Spring Framework introduced key features as part of new Java configuration support. This makes use of the **@Configuration** annotated classes and **@Bean** annotated methods. **Note that:**
 - @Bean annotation has the same role as the <bean/> element.
 - Classes annotated with @Configuration allow to define inter-bean dependencies by simply calling other @Bean methods in the same class.

9. What are the bean scopes available in Spring?

The Spring Framework has five scope supports. They are:

- **Singleton:** The scope of bean definition while using this would be a single instance per IoC container.
- **Prototype:** Here, the scope for a single bean definition can be any number of object instances.
- **Request:** The scope of the bean definition is an HTTP request.
- **Session:** Here, the scope of the bean definition is HTTP-session.
- **Global-session:** The scope of the bean definition here is a Global HTTP session.

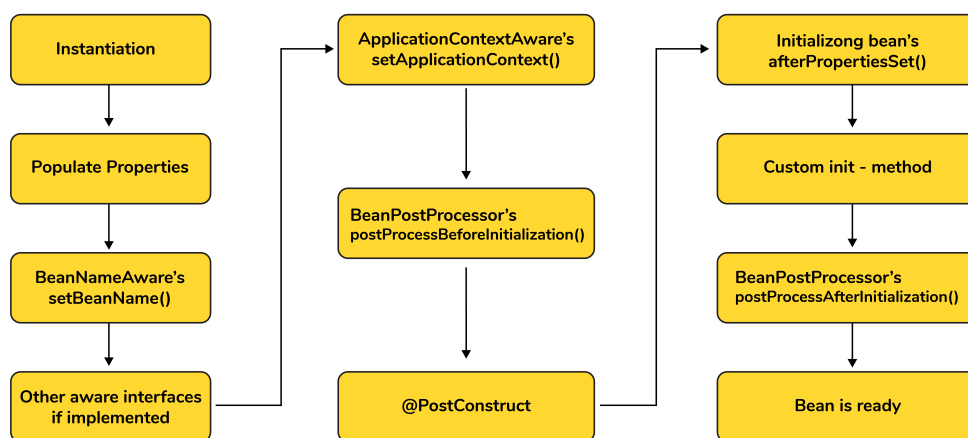
Note: The last three scopes are available only if the users use web-aware ApplicationContext containers.

10. Explain Bean life cycle in Spring Bean Factory Container.

The Bean life cycle is as follows:

- The IoC container instantiates the bean from the bean's definition in the XML file.
- Spring then populates all of the properties using the dependency injection as specified in the bean definition.
- The bean factory container calls `setBeanName()` which take the bean ID and the corresponding bean has to implement `BeanNameAware` interface.
- The factory then calls `setBeanFactory()` by passing an instance of itself (if `BeanFactoryAware` interface is implemented in the bean).
- If `BeanPostProcessors` is associated with a bean, then the `preProcessBeforeInitialization()` methods are invoked.
- If an init-method is specified, then it will be called.
- Lastly, `postProcessAfterInitialization()` methods will be called if there are any `BeanPostProcessors` associated with the bean that needs to be run post creation.

Spring Bean Life Cycle



InterviewBit

11. What do you understand by Bean Wiring.

- When beans are combined together within the Spring container, they are said to be wired or the phenomenon is called bean wiring.
- The Spring container should know what beans are needed and how the beans are dependent on each other while wiring beans. This is given by means of XML / Annotations / Java code-based configuration.

12. What is autowiring and name the different modes of it?

The IoC container autowires relationships between the application beans. Spring lets collaborators resolve which bean has to be wired automatically by inspecting the contents of the BeanFactory.

Different modes of this process are:

- **no**: This means **no autowiring** and is the default setting. An explicit bean reference should be used for wiring.
- **byName**: The bean dependency is injected according to the **name of the bean**. This matches and wires its properties with the beans defined by the same names as per the configuration.
- **byType**: This injects the bean dependency based on **type**.
- **constructor**: Here, it injects the bean dependency **by calling the constructor** of the class. It has a large number of parameters.
- **autodetect**: First the container tries to wire using autowire by the constructor, if it isn't possible then it tries to autowire by byType.

13. What are the limitations of autowiring?

- **Overriding possibility**: Dependencies are specified using `<constructor-arg>` and `<property>` settings that override autowiring.
- **Data types restriction**: Primitive data types, Strings, and Classes can't be autowired.

Spring Boot Interview Questions

14. What do you understand by the term 'Spring Boot'?

Spring Boot is an open-source, java-based framework that provides support for Rapid Application Development and gives a platform for developing stand-alone and production-ready spring applications with a need for very few configurations.

15. Explain the advantages of using Spring Boot for application development.

- Spring Boot helps to create stand-alone applications which can be started using java.jar (Doesn't require configuring WAR files).
- Spring Boot also offers pinpointed 'started' POMs to Maven configuration.
- Has provision to embed Undertow, Tomcat, Jetty, or other web servers directly.
- Auto-Configuration: Provides a way to automatically configure an application based on the dependencies present on the classpath.
- Spring Boot was developed with the intention of lessening the lines of code.
- It offers production-ready support like monitoring and apps developed using spring boot are easier to launch.

16. Differentiate between Spring and Spring Boot.

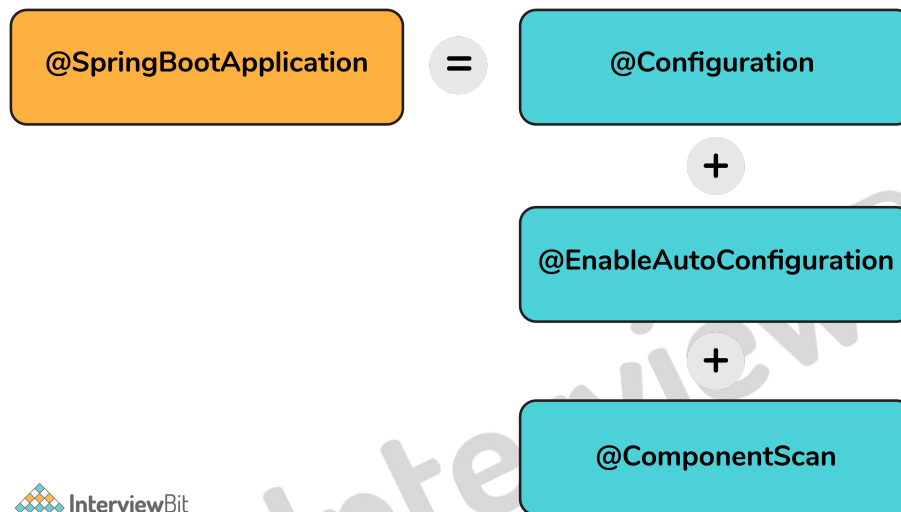
- The Spring Framework provides multiple features like dependency injection, data binding, aspect-oriented programming (AOP), data access, and many more that help easier development of web applications whereas Spring Boot helps in easier usage of the Spring Framework by simplifying or managing various loosely coupled blocks of Spring which are tedious and have a potential of becoming messy.
- Spring boot simplifies commonly used spring dependencies and runs applications straight from a command line. It also doesn't require an application container and it helps in monitoring several components and configures them externally.

17. What are the features of Spring Boot?

- **Spring Boot CLI** – This allows you to Groovy / Maven for writing Spring boot application and avoids boilerplate code.
- **Starter Dependency** – With the help of this feature, Spring Boot aggregates common dependencies together and eventually improves productivity and reduces the burden on
- **Spring Initializer** – This is a web application that helps a developer in creating an internal project structure. The developer does not have to manually set up the structure of the project while making use of this feature.
- **Auto-Configuration** – This helps in loading the default configurations according to the project you are working on. In this way, unnecessary WAR files can be avoided.
- **Spring Actuator** – Spring boot uses actuator to provide “Management EndPoints” which helps the developer in going through the Application Internals, Metrics etc.
- **Logging and Security** – This ensures that all the applications made using Spring Boot are properly secured without any hassle.

18. What does @SpringBootApplication annotation do internally?

As per the Spring Boot documentation, the @SpringBootApplication annotation is one point replacement for using @Configuration, @EnableAutoConfiguration and @ComponentScan annotations alongside their default attributes.



This enables the developer to use a single annotation instead of using multiple annotations thus lessening the lines of code. However, Spring provides loosely coupled features which is why we can use these annotations as per our project needs.

19. What are the effects of running Spring Boot Application as “Java Application”?

- The application automatically launches the tomcat server as soon as it sees that we are running a web application.

20. What is Spring Boot dependency management system?

- It is basically used to manage dependencies and configuration automatically without the need of specifying the version for any of that dependencies.

21. What are the possible sources of external configuration?

- Spring Boot allows the developers to run the same application in different environments by making use of its feature of external configuration. This uses environment variables, properties files, command-line arguments, YAML files, and system properties to mention the required configuration properties for its corresponding environments. Following are the sources of external configuration:
 - **Command-line properties** – Spring Boot provides support for command-line arguments and converts these arguments to properties and then adds them to the set of environment properties.
 - **Application Properties** – By default, Spring Boot searches for the application properties file or its YAML file in the current directory of the application, classpath root, or config directory to load the properties.
 - **Profile-specific properties** – Properties are loaded from the `application-{profile}.properties` file or its YAML file. This file resides in the same location as that of the non-specific property files and the `{profile}` placeholder refers to an active profile or an environment.

22. Can we change the default port of the embedded Tomcat server in Spring boot?

- Yes, we can change it by using the application properties file by adding a property of `server.port` and assigning it to any port you wish to.
- For example, if you want the port to be 8081, then you have to mention `server.port=8081`. Once the port number is mentioned, the application properties file will be automatically loaded by Spring Boot and the specified configurations will be applied to the application.

23. Can you tell how to exclude any package without using the basePackages filter?

We can use the `exclude` attribute while using the annotation `@SpringBootApplication` as follows:

```
@SpringBootApplication(exclude= {Student.class})
public class InterviewBitAppConfiguration {}
```

24. How to disable specific auto-configuration class?

- You can use the `exclude` attribute of `@EnableAutoConfiguration` for this purpose as shown below:

```
@EnableAutoConfiguration(exclude = {InterviewBitAutoConfiguration.class})
```

If the class is not specified on the classpath, we can specify the fully qualified name as the value for the `excludeName` .

```
//By using "excludeName"
@EnableAutoConfiguration(excludeName={Foo.class})
```

- You can add into the application.properties and multiple classes can be added by keeping it comma-separated.

25. Can the default web server in the Spring Boot application be disabled?

Yes! `application.properties` is used to configure the web application type, by mentioning `spring.main.web-application-type=none` .

26. What are the uses of @RequestMapping and @RestController annotations in Spring Boot?

- **@RequestMapping:**

- This provides the routing information and informs Spring that any HTTP request matching the URL must be mapped to the respective method.
- `org.springframework.web.bind.annotation.RequestMapping` has to be imported to use this annotation.

- **@RestController:**

- This is applied to a class to mark it as a request handler thereby creating RESTful web services using Spring MVC. This annotation adds the `@ResponseBody` and `@Controller` annotation to the class.
- `org.springframework.web.bind.annotation.RestController` has to be imported to use this annotation.

Check out more Interview Questions on Spring Boot [here](#).

Spring AOP, Spring JDBC, Spring Hibernate Interview Questions

27. What is Spring AOP?

- Spring AOP (Aspect Oriented Programming) is similar to OOPs (Object Oriented Programming) as it also provides modularity.
- In AOP key unit is **aspects** or **concerns** which are nothing but stand-alone modules in the application. Some aspects have centralized code but other aspects may be scattered or tangled code like in the case of logging or transactions. These scattered aspects are called **cross-cutting concern**.
 - A cross-cutting concern such as transaction management, authentication, logging, security etc is a concern that could affect the whole application and should be centralized in one location in code as much as possible for security and modularity purposes.
- AOP provides platform to dynamically add these cross-cutting concerns before, after or around the actual logic by using simple pluggable configurations.
- This results in easy maintainance of code. Concerns can be added or removed simply by modifying configuration files and therefore without the need for recompiling complete sourcecode.
- There are 2 types of implementing Spring AOP:
 - Using XML configuration files
 - Using AspectJ annotation style

28. What is an advice? Explain its types in spring.

An advice is the implementation of cross-cutting concerns can be applied to other modules of the spring application. Advices are of mainly 5 types:

- **Before:**
 - This advice executes **before** a join point, but it does not have the ability to prevent execution flow from proceeding to the join point (unless it throws an exception).
 - To use this, use @Before annotation.
- **AfterReturning:**
 - This advice is to be executed **after** a join point **completes** normally i.e if a method returns without throwing an exception.
 - To use this, use @AfterReturning annotation.
- **AfterThrowing:**
 - This advice is to be executed if a method exits by **throwing an exception**.
 - To use this, use @AfterThrowing annotation.
- **After:**
 - This advice is to be executed **regardless** of the means by which a join point exits (normal return or exception encounter).
 - To use this, use @After annotation.
- **Around:**
 - This is the most powerful advice surrounds a join point such as a method invocation.
 - To use this, use @Around annotation.

29. What is Spring AOP Proxy pattern?

- A proxy pattern is a well-used design pattern where a proxy is an object that looks like another object but adds special functionality to it behind the scenes.
- Spring AOP follows proxy-based pattern and this is created by the AOP framework to implement the aspect contracts in runtime.
- The standard JDK dynamic proxies are default AOP proxies that enables any interface(s) to be proxied. Spring AOP can also use CGLIB proxies that are required to proxy classes, rather than interfaces. In case a business object does not implement an interface, then CGLIB proxies are used by default.

30. What are some of the classes for Spring JDBC API?

- Following are the classes
 - JdbcTemplate
 - SimpleJdbcTemplate
 - NamedParameterJdbcTemplate
 - SimpleJdbcInsert
 - SimpleJdbcCall
- The most commonly used one is JdbcTemplate. This internally uses the JDBC API and has the advantage that we don't need to create connection, statement, start transaction, commit transaction, and close connection to execute different queries. All these are handled by JdbcTemplate itself. The developer can focus on executing the query directly.

31. How can you fetch records by Spring JdbcTemplate?

This can be done by using the query method of JdbcTemplate. There are two interfaces that help to do this:

- **ResultSetExtractor:**

- It defines only one method `extractData` that accepts `ResultSet` instance as a parameter and returns the list.
- Syntax:

```
public T extractData(ResultSet rs) throws SQLException, DataAccessException;
```

- **RowMapper:**

- This is an enhanced version of `ResultSetExtractor` that saves a lot of code.
- It allows to map a row of the relations with the instance of the user-defined class.
- It iterates the `ResultSet` internally and adds it into the result collection thereby saving a lot of code to fetch records.

32. What is Hibernate ORM Framework?

- Object-relational mapping (ORM) is the phenomenon of mapping application domain model objects to the relational database tables and vice versa.
- Hibernate is the most commonly used java based ORM framework.

33. What are the two ways of accessing Hibernate by using Spring.

- Inversion of Control approach by using Hibernate Template and Callback.
- Extending `HibernateDAOSupport` and Applying an AOP Interceptor node.

34. What is Hibernate Validator Framework?

- Data validation is a crucial part of any application. We can find data validation in:
 - UI layer before sending objects to the server
 - At the server-side before processing it
 - Before persisting data into the database
- Validation is a cross-cutting concern/task, so as good practice, we should try to keep it apart from our business logic. JSR303 and JSR349 provide specifications for bean validation by using annotations.
- This framework provides the reference implementation for JSR303 and JSR349 specifications.

35. What is HibernateTemplate class?

- Prior to Hibernate 3.0.1, Spring provided 2 classes namely:
 - `HibernateDaoSupport` to get the Session from Hibernate and
 - `HibernateTemplate` for Spring transaction management purposes.
- However, from Hibernate 3.0.1 onwards, by using `HibernateTemplate` class we can use `SessionFactory.getCurrentSession()` method to get the current session and then use it to get the transaction management benefits.
- `HibernateTemplate` has the benefit of exception translation but that can be achieved easily by using `@Repository` annotation with service classes.

Spring MVC Interview Questions

36. What is the Spring MVC framework?

- Spring MVC is request driven framework and one of the core components of the Spring framework.
- It comes with ready to use loosely coupled components and elements that greatly aids developers in building flexible and robust web applications.
- The MVC (Model - View - Controller) architecture separates and provides loose coupling between the different aspects of the application – input logic (Model), business logic (Controller), and UI logic (View).

37. What are the benefits of Spring MVC framework over other MVC frameworks?

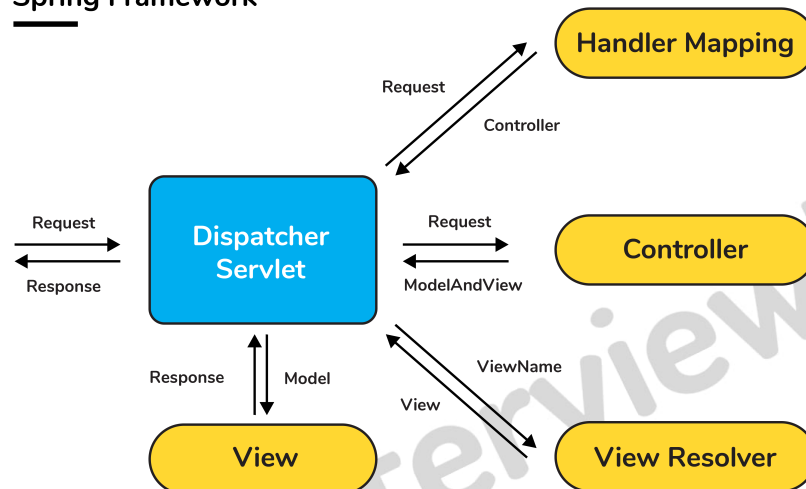
- Clear separation of roles – There is a specialised dedicated object for every role.
- Reusable business code logic – With Spring MVC, there is no need for duplicating the code. Existing objects can be used as commands instead of replicating them in order to extend a particular framework base class.
- Spring MVC framework provides customizable binding and validation.
- Also provides customizable locale and theme resolution.
- Spring MVC supports customizable handler mapping and view resolution too.

38. What is DispatcherServlet in Spring MVC?

Spring MVC framework is built around a central servlet called DispatcherServlet that handles all the HTTP requests and responses. The DispatcherServlet does a lot more than that:

- It seamlessly integrates with the IoC container and allows you to use each feature of Spring in an easier manner.
- The DispatcherServlet contacts HandlerMapping to call the appropriate Controller for processing the request on receiving it. Then, the controller calls appropriate service methods to set or process the Model data. The service processes the data and returns the view name to DispatcherServlet. DispatcherServlet then takes the help of ViewResolver and picks up the defined view for the request. Once the view is decided, the DispatcherServlet passes the Model data to View where it is finally rendered on the browser.

Spring Framework



39. What is a View Resolver pattern and explain its significance in Spring MVC?

- It is a J2EE pattern that allows the applications to dynamically choose technology for rendering the data on the browser (View).
 - Any technology like HTML, JSP, XSLT, JSF, or any other such technology can be used as View.
- The View Resolver has the information of different views. The Controller returns the name of the View which is then passed to View Resolver by the DispatcherServlet for selecting the appropriate View technology and then the data is displayed.
- The default ViewResolver used in Spring MVC is `InternalResourceViewResolver`.

40. What is the @Controller annotation used for?

- The @Controller is a stereotype Spring MVC annotation to define a Controller.

41. Can you create a controller without using @Controller or @RestController annotations?

- Yes! You can create a controller without `@Controller` or `@RestController` annotations by annotating the Spring MVC Controller classes using the `@Component` annotation. In this case, the real job of request mapping to handler method is done using the `@RequestMapping` annotation.

42. What is ContextLoaderListener and what does it do?

- The ContextLoaderListener loads and creates the ApplicationContext, so a developer need not write explicit code to do create it. In short, it is a listener that aids to bootstrap Spring MVC.
 - The application context is where Spring bean resides. For a web application, there is a subclass called WebApplicationContext.
- The lifecycle of the ApplicationContext is tied to the lifecycle of the ServletContext by using ContextLoaderListener. The ServletContext from the WebApplicationContext can be obtained using the `getServletContext()` method.

43. What are the differences between @RequestParam and @PathVariable annotations?

- Even though both these annotations are used to extract some data from URL, there is a key difference between them.
 - The `@RequestParam` is used to extract **query parameters** that is anything after “?” in the URL.
 - The `@PathVariable` is used to extract the data present as part of the URI itself.]
 - For example, if the given URL is `http://localhost:8080/InterviewBit/Spring/SpringMVC/?format=json`, then you can access the query parameter “format” using the `@RequestParam` annotation and `/Spring/{type}` using the `@PathVariable`, which will give you SpringMVC.

```
@RequestMapping("/Spring/{type}")
public void getQuestions(@PathVariable("type") String type,
                        @RequestParam(value = "format", required = false) String format) {
    /* Some code */
}
```

44. What is the Model in Spring MVC?

- Model is a reference to have the data for rendering.
- It is always created and passed to the view in Spring MVC. If a mapped controller method has Model as a parameter, then that model instance is automatically injected to that method.
- Any attributes set on the injected model would be preserved and passed to the View.

45. What is the use of @Autowired annotation?

`@Autowired` annotation is meant for the injection of a bean by means of its type along with methods and fields. This helps the Spring framework to resolve dependencies by injecting and collaborating the beans into another bean. For example, consider the below code snippet:

```
import org.springframework.beans.factory.annotation.Autowired;
import java.util.*;
public class InterviewBit {
    // Autowiring/Injecting FormatterUtil as dependency to InterviewBit class
    @Autowired
    private FormatterUtil formatterUtil;

    public Date something( String value ){
        Date dateFormatted = formatterUtil.formatDate(value);
        return dateFormatted
    }
}
/**
 * Util class to format any string value to valid date format
 */
public class FormatterUtil {

    public Date formatDate(String value){
        //code to format date
    }
}
```

46. What is the role of @ModelAttribute annotation?

The annotation plays a very important role in binding method parameters to the respective attribute that corresponds to a model. Then it reflects the same on the presentation page. The role of the annotation also depends on what the developer is using that for. In case, it is used at the method level, then that method is responsible for adding attributes to it. When used at a parameter level, it represents that the parameter value is meant to be retrieved from the model layer.

47. What is the importance of the web.xml in Spring MVC?

`web.xml` is also known as the Deployment Descriptor which has definitions of the servlets and their mappings, filters, and lifecycle listeners. It is also used for configuring the ContextLoaderListener. Whenever the application is deployed, a ContextLoaderListener instance is created by Servlet container which leads to a load of WebApplicationContext.

48. What are the types of Spring MVC Dependency Injection?

There are two types of DI (Dependency Injection):

- Construction-Based:

- This type of DI is accomplished when the Spring IoC (Inversion of Control) container invokes parameterized constructor having a dependency on other classes.
- This cannot instantiate the values partially and ensures that the dependency injection is done fully.
- There are two possible ways of achieving this:

Annotation Configuration: This approach uses POJO objects and annotations for configuration. For example, consider the below code snippet:

```
@Configuration
@ComponentScan("com.interviewbit.constructordi")
public class SpringAppConfig {
    @Bean
    public Shape shapes() {
        return new Shapes("Rectangle");
    }
    @Bean
    public Dimension dimensions() {
        return new Dimension(4,3);
    }
}
```

Here, the annotations are used for notifying the Spring runtime that the class specified with `@Bean` annotation is the provider of beans and the process of context scan needs to be performed on the package `com.interviewbit.constructordi` by means of `@ComponentScan` annotation. Next, we will be defining a Figure class component as below:

```
@Component
public class Figure {
    private Shape shape;
    private Dimension dimension;

    @Autowired
    public Figure(Shape shape, Dimension dimension) {
        this.shape = shape;
        this.dimension = dimension;
    }
}
```

Spring encounters this Figure class while performing context scan and it initializes the instance of this class by invoking the constructor annotated with `@Autowired`. The Shape and Dimension instances are obtained by calling the methods annotated with `@Bean` in the `SpringAppConfig` class. Instances of Engine and Transmission will be obtained by calling `@Bean` annotated methods of the Config class. Finally, we need to bootstrap an `ApplicationContext` using our POJO configuration:

```
ApplicationContext context = new AnnotationConfigApplicationContext(SpringAppConfig.class);
Figure figure = context.getBean(Figure.class);
```

XML Configuration: This is another way of configuring Spring runtime by using the XML configuration file. For example, consider the below code snippet in the `springAppConfig.xml` file:

```
<bean id="toyota" class="com.interviewbit.constructordi.Figure">
    <constructor-arg index="0" ref="shape"/>
    <constructor-arg index="1" ref="dimension"/>
</bean>
<bean id="shape" class="com.interviewbit.constructordi.Shape">
    <constructor-arg index="0" value="Rectangle"/>
</bean>
<bean id="dimension" class="com.interviewbit.constructordi.Dimension">
    <constructor-arg index="0" value="4"/>
    <constructor-arg index="1" value="3"/>
</bean>
```

The `constructor-arg` tag can accept either literal value or another bean's reference and explicit index and type. The index and type arguments are used for resolving conflicts in cases of ambiguity.

While bootstrapping this class, the Spring `ApplicationContext` needs to use `ClassPathXmlApplicationContext` as shown below:

```
ApplicationContext context = new ClassPathXmlApplicationContext("springAppConfig.xml");
Figure figure = context.getBean(Figure.class);
```

- **Setter-Based:**

- This form of DI is achieved when the Spring IoC container calls the bean's setter method after a non-parameterized constructor is called to perform bean instantiation.
- It is possible to achieve circular dependency using setter injection.
- For achieving this type of DI, we need to configure it through the configuration file under the `<property>` tag. For example, consider a class `InterviewBit` that sets the property `articles` as shown below:

```
package com.interviewbit.model;
import com.interviewbit.model.Article;
public class InterviewBit {
    // Object of the Article interface
    Article article;
    public void setArticle(Article article)
    {
        this.article = article;
    }
}
```

In the bean configuration file, we will be setting as below:


```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans.xsd">
  <bean id="InterviewBit" class="com.interviewbit.model.InterviewBit">
    <property name="article">
      <ref bean="JsonArticle" />
    </property>
  </bean>
  <bean id="JsonArticle" class="com.interviewbit.bean.JsonArticle" />
</beans>
```

The 'JsonArticle' bean is injected into the InterviewBit class object by means of the `setArticle` method.

In cases where both types of dependencies are used, then the setter dependency injection has more preference by considering the specificity nature.

49. What is the importance of session scope?

Session scopes are used to create bean instances for HTTP sessions. This would mean that a single bean can be used for serving multiple HTTP requests. The scope of the bean can be defined by means of using scope attribute or using `@Scope` or `@SessionScope` annotations.

- Using scope attribute:

```
<bean id="userBean" class="com.interviewbit.UserBean" scope="session"/>
```

- Using `@Scope` annotation:

```
@Component
@Scope("session")
public class UserBean {
    //some methods and properties
}
```

- Using `@SessionScope`:

```
@Component
@SessionScope
public class UserBean {
    //some methods and properties
}
```

50. What is the importance of @Required annotation?

The annotation is used for indicating that the property of the bean should be populated via autowiring or any explicit value during the bean definition at the configuration time. For example, consider a code snippet below where we need to have the values of age and the name:

```
import org.springframework.beans.factory.annotation.Required;
public class User {
    private int age;
    private String name;

    @Required
    public void setAge(int age) {
        this.age = age;
    }
    public Integer getAge() {
        return this.age;
    }

    @Required
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}
```

51. Differentiate between the @Autowired and the @Inject annotations.

@Autowired	@Inject
This annotation is part of the Spring framework.	This annotation is part of Java CDI.
Has required attribute.	Does not have the required attribute.
Singleton is the default scope for autowired beans.	Prototype is the default scope of inject beans.
In case of ambiguity, then @Qualifier annotation is to be used.	In case of ambiguity, then @Named qualifier needs to be used.
Since this annotation is provided by the Spring framework, in case you shift to another Dependency injection framework, there would be a lot of refactoring needed.	Since this annotation is part of Java CDI, it is not framework dependent and hence less code refactoring when there are framework changes.

52. Are singleton beans thread-safe?

No, the singleton beans are not thread-safe because the concept of thread-safety essentially deals with the execution of the program and the singleton is simply a design pattern meant for the creation of objects. Thread safety nature of a bean depends on the nature of its implementation.

53. How can you achieve thread-safety in beans?

The thread safety can be achieved by changing the scope of the bean to request, session or prototype but at the cost of performance. This is purely based on the project requirements.

54. What is the significance of @Repository annotation?

@Repository annotation indicates that a component is used as the repository that acts as a means to store, search or retrieve data. These can be added to the DAO classes.

55. How is the dispatcher servlet instantiated?

The dispatcher servlet is instantiated by means of servlet containers such as Tomcat. The Dispatcher Servlet should be defined in web.xml The DispatcherServlet is instantiated by Servlet containers like Tomcat. The Dispatcher Servlet can be defined in web.xml as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javae

<!-- Define Dispatcher Servlet -->
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>InterviewBitServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>
```

Here, the load-on-startup tag is 1 which indicates that the DispatcherServlet is instantiated whenever the Spring MVC application is loaded into the servlet container. During this process, it looks for the servlet-name-context.xml file and initializes beans that are defined in the file.

56. How is the root application context in Spring MVC loaded?

The root application context is loaded using the ContextLoaderListener that belongs to the entire application. Spring MVC allows instantiating multiple DispatcherServlet and each of them have multiple contexts specific to them. They can have the same root context too.

57. How does the Spring MVC flow look like? In other words, How does a DispatcherServlet know what Controller needs to be called when there is an incoming request to the Spring MVC?

A Dispatcher Servlet knows which controller to call by means of handler mappings. These mappings have the mapping between the controller and the requests.

`BeanNameUrlHandlerMapping` and `SimpleUrlHandlerMapping` are the two most commonly used handler mappings.

- `BeanNameUrlHandlerMapping`: When the URL request matches the bean name, the class corresponding to the bean definition is the actual controller that is responsible for processing the request.
- `SimpleUrlHandlerMapping`: Here, the mapping is very explicit. The number of URLs can be specified here and each URL is associated explicitly with a controller.

If the Spring MVC is configured using annotations, then `@RequestMapping` annotations are used for this purpose. The `@RequestMapping` annotation is configured by making use of the URI path, HTTP methods, query parameters, and the HTTP Headers.

58. Where does the access to the model from the view come from?

The view requires access to the model to render the output as the model contains the required data meant for rendering. The model is associated with the controller that processes the client requests and finally encapsulates the response into the Model object.

59. Why do we need BindingResults?

BindingResults is an important Spring interface that is within the `org.springframework.validation` package. This interface has a very simple and easy process of invocation and plays a vital role in detecting errors in the submitted forms. However, care has to be taken by the developer to use the BindingResult parameter just after the object that needs validation. For example:

```
@PostMapping("/interviewbit")
public String registerCourse(@Valid RegisterUser registerUser,
    BindingResult bindingResult, Model model) {
    if (bindingResult.hasErrors()) {
        return "home";
    }
    model.addAttribute("message", "Valid inputs");
    return "home";
}
```

The Spring will understand to find the corresponding validators by checking the @Valid annotation on the parameter.

60. What are Spring Interceptors?

Spring Interceptors are used to pre-handle and post-handle the web requests in Spring MVC which are handled by Spring Controllers. This can be achieved by the `HandlerInterceptor` interface. These handlers are used for manipulating the model attributes that are passed to the controllers or the views.

The Spring handler interceptor can be registered for specific URL mappings so that it can intercept only those requests. The custom handler interceptor must implement the `HandlerInterceptor` interface that has 3 callback methods that can be implemented:

- `preHandle()`
- `postHandle()`
- `afterCompletion()`

The only problem with this interface is that all the methods of this interface need to be implemented irrespective of its requirements. This can be avoided if our handler class extends the `HandlerInterceptorAdapter` class that internally implements the `HandlerInterceptor` interface and provides default blank implementations.

61. Is there any need to keep `spring-mvc.jar` on the classpath or is it already present as part of `spring-core`?

The `spring-mvc.jar` does not belong to the `spring-core`. This means that the jar has to be included in the project's classpath if we have to use the Spring MVC framework in our project. For Java applications, the `spring-mvc.jar` is placed inside `/WEB-INF/lib` folder.

62. What are the differences between the `<context:annotation-config>` vs `<context:component-scan>` tags?

`<context:annotation-config>` is used for activating applied annotations in pre-registered beans in the application context. It also registers the beans defined in the config file and it scans the annotations within the beans and activates them.

The `<context:component-scan>` tag does the task of `<context:annotation-config>` along with scanning the packages and registering the beans in the application context.

`<context:annotation-config>` = Scan and activate annotations in pre-registered beans.

`<context:component-scan>` = Register Bean + Scan and activate annotations in package.

63. How is the form data validation done in Spring Web MVC Framework?

Spring MVC does the task of data validation using the validator object which implements the Validator interface. In the custom validator class that we have created, we can use the utility methods of the ValidationUtils class like

`rejectIfEmptyOrWhitespace()` or `rejectIfEmpty()` to perform validation of the form fields.

```
@Component
public class UserValidator implements Validator
{
    public boolean supports(Class clazz) {
        return UserVO.class.isAssignableFrom(clazz);
    }

    public void validate(Object target, Errors errors)
    {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "error.name", "Name is required");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "age", "error.age", "Age is required");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "phone", "error.phone", "Phone number is required");
    }
}
```

In the fields that are subject to validation, in case of errors, the validator methods would create field error and bind that to the field.

To activate the custom validator as spring bean, then:

- We have to add the `@Component` annotation on the custom validator class and initiate the component scanning of the package containing the validator declarations by adding the below change:

```
<context:component-scan base-package="com.interviewbit.validators"/>
```

OR

The validator class can be registered in the context file directly as a bean as shown:

```
<bean id="userValidator" class="com.interviewbit.validators.UserValidator" />
```


64. How to get ServletConfig and ServletContext objects in spring bean?

This can be done by either implementing the spring-aware interfaces or by using the @Autowired annotation.

```
@Autowired
private ServletContext servletContext;
@Autowired
private ServletConfig servletConfig;
```

65. Differentiate between a Bean Factory and an Application Context.

BeanFactory and the ApplicationContext are both Java interfaces. The difference is that the ApplicationContext extends the BeanFactory. BeanFactory provides both IoC and DI basic features whereas the ApplicationContext provides more advanced features. Following are the differences between these two:

Category	BeanFactory	Applications
Internationalization (i18n)	Does not provide support for i18n.	Provides support for i18n.
Event Publishing	Provides the ability to publish events to listener beans by using ContextStartedEvent and ContextStoppedEvent to publish context when it is started and stopped respectively.	ApplicationContext event handling, ApplicationListener and ApplicationEvent
Implementations	XMLBeanFactory is a popular implementation of BeanFactory.	ClassPathXmlApplicationContext is a popular implementation of ApplicationContext. It uses WebApplicationContext that extends ApplicationContext and adds getServletContext() method.
Autowiring	For autowiring, beans have to be registered in the AutoWiredBeanPostProcessor API.	Here, XML configuration is done to achieve autowiring.

66. How are i18n and localization supported in Spring MVC?

Spring MVC has `LocaleResolver` that supports i18n and localization. For supporting both internationalization and localization. The following beans need to be configured in the application:

- **SessionLocaleResolver:** This bean plays a vital role to get and resolve the locales from the pre-defined attributes in the user session.

Syntax:

```
<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />
<property name="defaultLocale" value="en" />
</bean>
```

- **LocaleChangeInterceptor:** This bean is useful to resolve the parameter from the incoming request.

Syntax:

```
<bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor" />
<property name="paramName" value="lang" />
</bean>
```

- **DefaultAnnotationHandlerMapping:** This refers to the HandlerMapping interface implementation which maps the handlers/interceptors based on the HTTP paths specified in the @RequestMapping at type or method level.

Syntax:

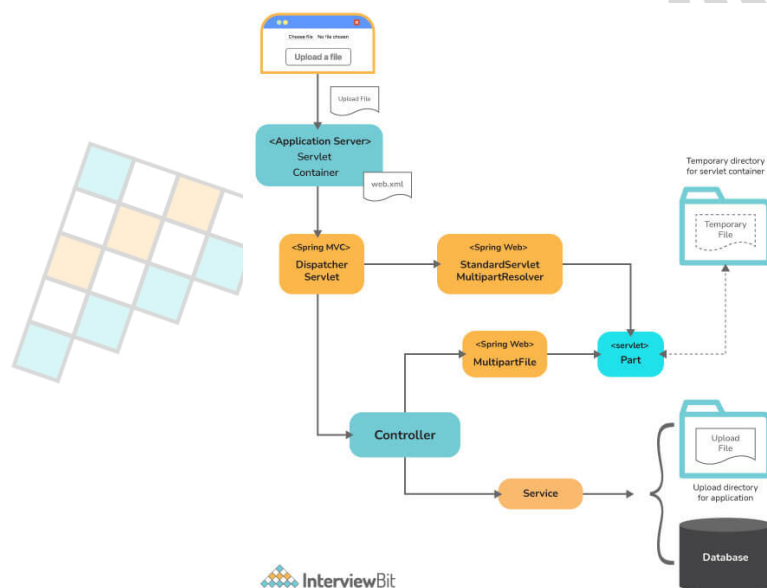
```
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerMapping" />
<property name="interceptors">
  <list>
    <ref bean="localeChangeInterceptor" />
  </list>
</property>
</bean>
```

67. What do you understand by MultipartResolver?

The MultipartResolver is used for handling the file upload scenarios in the Spring web application. There are 2 concrete implementations of this in Spring, they are:

- CommonsMultipartResolver meant for Jakarta Commons FileUpload
- StandardServletMultipartResolver meant for Servlet 3.0 Part API

To implement this, we need to create a bean with id="multipartResolver" in the application context of DispatcherServlet. Doing this ensures that all the requests handled by the DispatcherServlet have this resolver applied whenever a multipart request is detected. If a multipart request is detected by the DispatcherServlet, it resolves the request by means of the already configured MultipartResolver, and the request is passed on as a wrapped/abstract HttpServletRequest. Controllers then cast this request as the `MultipartHttpServletRequest` interface to get access to the Multipart files. The following diagram illustrates the flow clearly:



68. How is it possible to use the Tomcat JNDI DataSource in the Spring applications?

To use the servlet container which is configured in the JNDI (Java Naming and Directory Interface) DataSource, the DataSource bean has to be configured in the spring bean config file and then injected into the beans as dependencies. Post this, the DataSource bean can be used for performing database operations by means of the JdbcTemplate. The syntax for registering a MySQL DataSource bean:

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
  <property name="jndiName" value="java:comp/env/jdbc/MySQLDB"/>
</bean>
```

69. What will be the selection state of a checkbox input if the user first checks the checkbox and gets validation errors in other fields and then unchecks the checkbox after getting the errors?

The validation is generally performed during HTTP POST requests. During HTTP requests, if the state of the checkbox is unchecked, then HTTP includes the request parameter for the checkbox thereby not picking up the updated selection. This can be fixed by making use of a hidden form field that starts with `_` in the Spring MVC.

Conclusion:

In this article, we have seen the most commonly asked Spring Interview Questions during an interview. Spring is a very powerful framework that allows building enterprise-level web applications. Applications developed using Spring are generally quick, scalable, and transparent. Due to this, Spring has been embraced by a huge Java Developer's community thereby making it an inevitable part of any Java Developer's Job Role. Knowing Spring ensures that an added advantage is with the developers to progress steadily in their careers too.

Tip: We also recommend reading guides posted [here](#).

Links to More Interview Questions

[C Interview Questions](#)

[Php Interview Questions](#)

[C Sharp Interview Questions](#)

[Web Api Interview Questions](#)

[Hibernate Interview Questions](#)

[Node Js Interview Questions](#)

[Cpp Interview Questions](#)

[Oops Interview Questions](#)

[Devops Interview Questions](#)

[Machine Learning Interview Questions](#)

[Docker Interview Questions](#)

[Mysql Interview Questions](#)

[Css Interview Questions](#)

[Laravel Interview Questions](#)

[Asp Net Interview Questions](#)

[Django Interview Questions](#)

[Dot Net Interview Questions](#)

[Kubernetes Interview Questions](#)

[Operating System Interview Questions](#)

[React Native Interview Questions](#)

[Aws Interview Questions](#)

[Git Interview Questions](#)

[Java 8 Interview Questions](#)

[Mongodb Interview Questions](#)

[Dbms Interview Questions](#)

[Spring Boot Interview Questions](#)

[Power Bi Interview Questions](#)

[Pl Sql Interview Questions](#)

[Tableau Interview Questions](#)

[Linux Interview Questions](#)

[Ansible Interview Questions](#)




[Java Interview Questions](#)

[Jenkins Interview Questions](#)

Let's begin with the first section of Spring interview questions that is the General Questions.

General Questions – Spring Interview Questions

1. What are the major features in different versions of Spring Framework?

Version	Logo	Feature
Spring 2.5		This version was released in 2007. It was the first version which supported annotations.
Spring 3.0		This version was released in 2009. It made full-fledged use of improvements in Java5 and also provided support to JEE6.
Spring 4.0		This version was released in 2013. This was the first version to provide full support to Java 8.

Features of Spring Framework

2. What is a Spring Framework?

- Spring is a powerful open source, application framework created to reduce the complexity of enterprise application development.
- It is light-weighted and loosely coupled.
- It has layered architecture, which allows you to select the components to use, while also providing a cohesive framework for J2EE application development.
- Spring framework is also called the framework of frameworks as it provides support to various other frameworks such as Struts, Hibernate, Tapestry, EJB, JSF etc.



3. List the advantages of Spring Framework.

- Because of Spring Frameworks layered architecture, you can use what you need and leave which you don't.
- Spring Framework enables POJO (Plain Old Java Object) Programming which in turn enables continuous integration and testability.
- JDBC is simplified due to Dependency Injection and Inversion of Control.
- It is open-source and has no vendor lock-in.

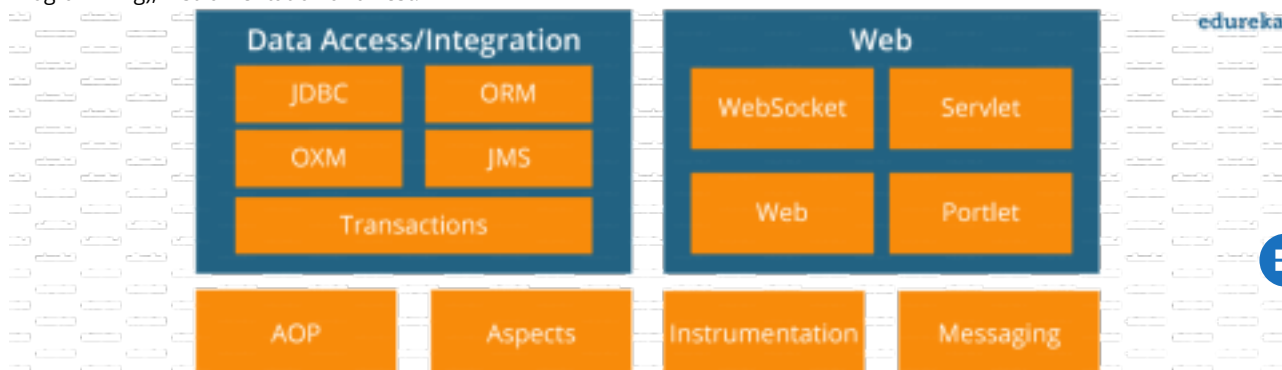
4. What are the different features of Spring Framework?

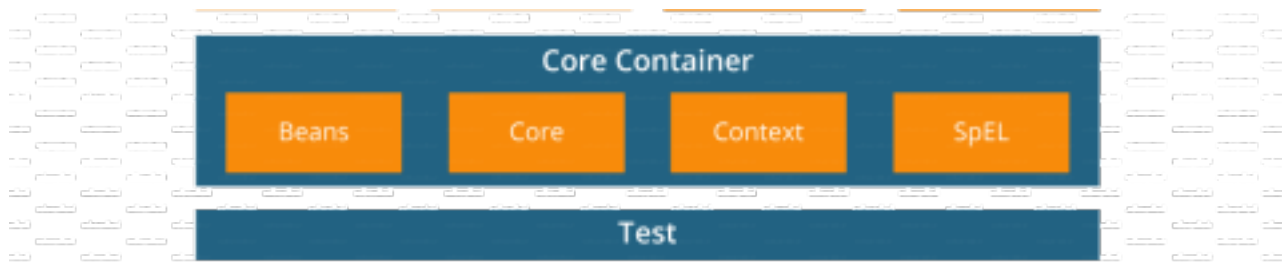
Following are some of the major features of Spring Framework :

- **Lightweight:** Spring is lightweight when it comes to size and transparency.
- **Inversion of control (IOC):** The objects give their dependencies instead of creating or looking for dependent objects. This is called Inversion Of Control.
- **Aspect oriented Programming (AOP):** Aspect oriented programming in Spring supports cohesive development by separating application business logic from system services.
- **Container:** Spring Framework creates and manages the life cycle and configuration of the application objects.
- **MVC Framework:** Spring Framework's MVC web application framework is highly configurable. Other frameworks can also be used easily instead of Spring MVC Framework.
- **Transaction Management:** Generic abstraction layer for transaction management is provided by the Spring Framework. Spring's transaction support can be also used in container less environments.
- **JDBC Exception Handling:** The JDBC abstraction layer of the Spring offers an exception hierarchy, which simplifies the error handling strategy.

5. How many modules are there in Spring Framework and what are they?

There are around 20 modules which are generalized into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation and Test.





- **Spring Core Container** – This layer is basically the core of Spring Framework. It contains the following modules :

- Spring Core
- Spring Bean
- SpEL (Spring Expression Language)
- Spring Context

- **Data Access/Integration** – This layer provides support to interact with the database. It contains the following modules :

- JDBC (Java DataBase Connectivity)
- ORM (Object Relational Mapping)
- OXM (Object XML Mappers)
- JMS (Java Messaging Service)
- Transaction

- **Web** – This layer provides support to create web application. It contains the following modules :

- Web
- Web – MVC
- Web – Socket
- Web – Portlet

- **Aspect Oriented Programming (AOP)** – In this layer you can use Advices, Pointcuts etc., to decouple the code.

- **Instrumentation** – This layer provides support to class instrumentation and classloader implementations.

- **Test** – This layer provides support to testing with JUnit and TestNG.

Few Miscellaneous modules are given below:

- **Messaging** – This module provides support for STOMP. It also supports an annotation programming model that is used for routing and processing STOMP messages from WebSocket clients.
- **Aspects** – This module provides support to integration with AspectJ.

6. What is a Spring configuration file?

A Spring configuration file is an XML file. This file mainly contains the classes information. It describes how those classes are configured as well as introduced to each other. The XML configuration files, however, are verbose and more clean. If it's not planned and written correctly, it becomes very difficult to manage in big projects.



7. What are the different components of a Spring application?

A Spring application, generally consists of following components:

- **Interface**: It defines the functions.
- **Bean class**: It contains properties, its setter and getter methods, functions etc.
- **Spring Aspect Oriented Programming (AOP)**: Provides the functionality of cross-cutting concerns.
- **Bean Configuration File**: Contains the information of classes and how to configure them.
- **User program**: It uses the function.

8. What are the various ways of using Spring Framework?

Spring Framework can be used in various ways. They are listed as follows:

- As a Full-fledged Spring web application.
- As a third-party web framework, using Spring Frameworks middle-tier.
- For remote usage.
- As Enterprise Java Bean which can wrap existing POJOs (Plain Old Java Objects).



The next section of Spring Interview Questions is on *Dependency Injection and IoC container*.

Dependency Injection/ IoC Container – Spring Interview Questions

9. What is Spring IOC Container?

At the core of the Spring Framework, lies the Spring container. The container creates the object, wires them together, configures them and manages their complete life cycle. The Spring container makes use of Dependency Injection to manage the components that make up an application. The container receives instructions for which objects to instantiate, configure, and assemble by reading the configuration metadata provided. This metadata can be provided either by XML, Java annotations or Java code.



10. What do you mean by Dependency Injection?

In Dependency Injection, you do not have to create your objects but have to describe how they should be created. You don't connect your components and services together in the code directly, but describe which services are needed by which components in the configuration file. The IoC container will wire them up together.

11. In how many ways can Dependency Injection be done?

In general, dependency injection can be done in three ways, namely :

- Constructor Injection
- Setter Injection
- Interface Injection

In Spring Framework, only constructor and setter injections are used.

12. Differentiate between constructor injection and setter injection.

Constructor Injection vs Setter Injection

Constructor Injection	Setter Injection
There is no partial injection.	There can be partial injection.
It doesn't override the setter property.	It overrides the constructor property.
It will create a new instance if any modification is done.	It will not create new instance if any modification is done.
It works better for many properties.	It works better for few properties.

13. How many types of IOC containers are there in spring?

- BeanFactory:** BeanFactory is like a factory class that contains a collection of beans. It instantiates the bean whenever asked for by clients.
- ApplicationContext:** The ApplicationContext interface is built on top of the BeanFactory interface. It provides some extra functionality on top BeanFactory.

14. Differentiate between BeanFactory and ApplicationContext.

BeanFactory vs ApplicationContext

BeanFactory	ApplicationContext
It is an interface defined in org.springframework.beans.factory. BeanFactory	It is an interface defined in org.springframework.context. ApplicationContext
It uses Lazy initialization	It uses Eager/ Aggressive initialization
It explicitly provides a resource object using the syntax	It creates and manages resource objects on its own
It doesn't supports internationalization	It supports internationalization
It doesn't supports annotation based dependency	It supports annotation based dependency

15. List some of the benefits of IoC.

Some of the benefits of IoC are:

- It will minimize the amount of code in your application.
- It will make your application easy to test because it doesn't require any singletons or JNDI lookup mechanisms in your test cases.
- It promotes loose coupling with minimal effort and least intrusive mechanism.
- It supports eager instantiation and lazy loading of the services.



Let's move on to the next section of Spring Interview Questions, that is *Spring Beans Interview Questions*.

Spring Beans – Spring Interview Questions

16. Explain Spring Beans?

- They are the objects that form the backbone of the user's application.
- Beans are managed by the Spring IoC container.
- They are instantiated, configured, wired and managed by a Spring IoC container
- Beans are created with the configuration metadata that the users supply to the container.



17. How configuration metadata is provided to the Spring container?

Configuration metadata can be provided to Spring container in following ways:

- **XML-Based configuration:** In Spring Framework, the dependencies and the services needed by beans are specified in configuration files which are in XML format. These configuration files usually contain a lot of bean definitions and application specific configuration options. They generally start with a bean tag. For example:

```

1 <bean id="studentbean" class="org.edureka.firstSpring.StudentBean">
2   <property name="name" value="Eduureka"></property>
3 </bean>
  
```

- **Annotation-Based configuration:** Instead of using XML to describe a bean wiring, you can configure the bean into the component class itself by using annotations on the relevant class, method, or field declaration. By default, annotation wiring is not turned on in the Spring container. So, you need to enable it in your Spring configuration file before using it. For example:

```

1 <beans>
2   <context:annotation-config/>
3   <!-- bean definitions go here -->
4 </beans>
  
```

- **Java-based configuration:** The key features in Spring Framework's new Java-configuration support are @Configuration annotated classes and @Bean annotated methods.

“

1. @Bean annotation plays the same role as the <bean/> element. ”

“

2.@Configuration classes allows to define inter-bean dependencies by simply calling other @Bean methods in the same class. ”

For example:

```

1 @Configuration
2 public class StudentConfig
3 {
4   @Bean
5   public StudentBean myStudent()
6   { return new StudentBean(); }
7 }
  
```

18. How many bean scopes are supported by Spring?

The Spring Framework supports five scopes. They are:

- **Singleton:** This provides scope for the bean definition to single instance per Spring IoC container.
- **Prototype:** This provides scope for a single bean definition to have any number of object instances.
- **Request:** This provides scope for a bean definition to an HTTP-request.
- **Session:** This provides scope for a bean definition to an HTTP-session.
- **Global-session:** This provides scope for a bean definition to an Global HTTP-session.

The last three are available only if the users use a web-aware ApplicationContext.





Spring Framework Certification Course

[Instructor-led Sessions](#)

[Real-life Case Studies](#)

[Assignments](#)

[Lifetime Access](#)

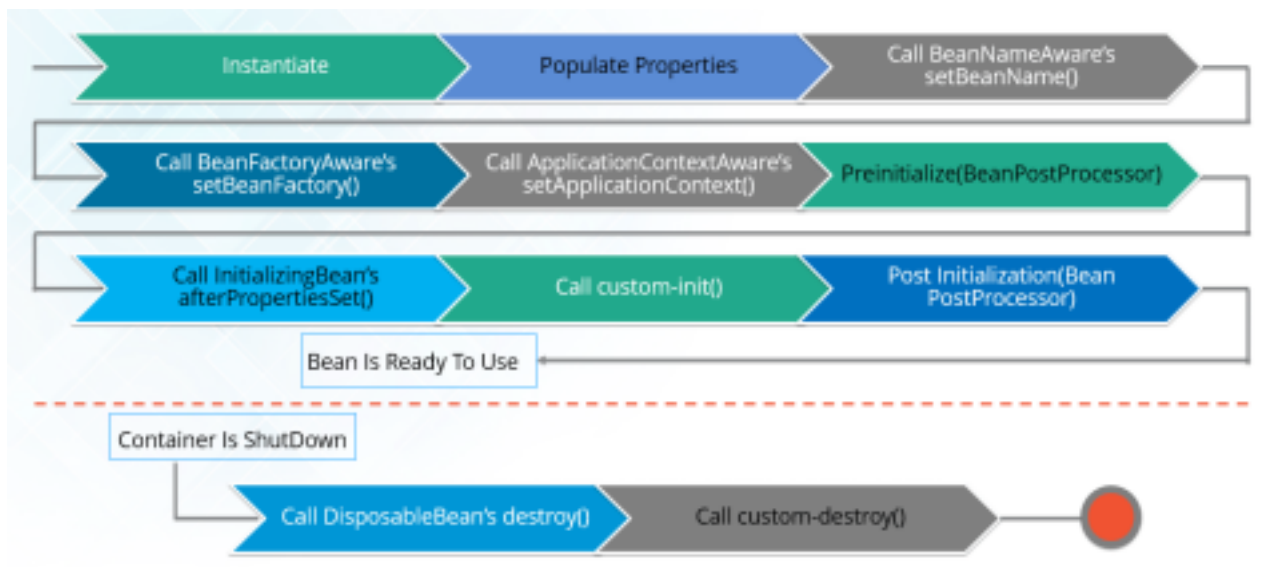
[Explore Curriculum](#)

19. What is the Bean life cycle in Spring Bean Factory Container?

Bean life cycle in Spring Bean Factory Container is as follows:

1. The Spring container instantiates the bean from the bean's definition in the XML file.
2. Spring populates all of the properties using the dependency injection, as specified in the bean definition.
3. The factory calls `setBeanName()` by passing the bean's ID, if the bean implements the `BeanNameAware` interface.
4. The factory calls `setBeanFactory()` by passing an instance of itself, if the bean implements the `BeanFactoryAware` interface.
5. `preProcessBeforeInitialization()` methods are called if there are any `BeanPostProcessors` associated with the bean.
6. If an `init-method` is specified for the bean, then it will be called.
7. Finally, `postProcessAfterInitialization()` methods will be called if there are any `BeanPostProcessors` associated with the bean.

To understand it in better way check the below diagram:



20. Explain inner beans in Spring.

A bean can be declared as an inner bean only when it is used as a property of another bean. For defining a bean, the Spring's XML based configuration metadata provides the use of `<bean>` element inside the `<property>` or `<constructor-arg>`. Inner beans are always anonymous and they are always scoped as prototypes. For example, let's say we have one `Student` class having reference of `Person` class. Here we will be creating only one instance of `Person` class and use it inside `Student`.

Here's a `Student` class followed by bean configuration file:

`Student.java`



Spring Interview Questions

Spring interview questions and answers are frequently asked because it is now widely used framework to develop enterprise application in java. There are given a list of top 40 frequently asked spring interview questions.

1) What is Spring?

It is a lightweight, loosely coupled and integrated framework for developing enterprise applications in java.

2) What are the advantages of spring framework?

1. Predefined Templates
2. Loose Coupling
3. Easy to test
4. Lightweight
5. Fast Development
6. Powerful Abstraction
7. Declarative support

[More details...](#)

3) What are the modules of spring framework?

1. Test
2. Spring Core Container
3. AOP, Aspects and Instrumentation
4. Data Access/Integration
5. Web

[More details...](#)

4) What is IOC and DI?

IOC (Inversion of Control) and DI (Dependency Injection) is a design pattern to provide loose coupling. It removes the dependency from the program.

Let's write a code without following IOC and DI.



```
public class Employee{  
    Address address;  
    Employee(){  
        address=new Address();//creating instance  
    }  
}
```

Now, there is dependency between Employee and Address because Employee is forced to use the same address instance.

Let's write the IOC or DI code.

```
public class Employee{  
    Address address;  
    Employee(Address address){  
        this.address=address;//not creating instance  
    }  
}
```

Now, there is no dependency between Employee and Address because Employee is not forced to use the same address instance. It can use any address instance.

5) What is the role of IOC container in spring?

IOC container is responsible to:

- create the instance
- configure the instance, and
- assemble the dependencies

[More details...](#)

6) What are the types of IOC container in spring?

There are two types of IOC containers in spring framework.



1. BeanFactory
2. ApplicationContext

[More details...](#)

7) What is the difference between BeanFactory and ApplicationContext?

BeanFactory is the **basic container** whereas ApplicationContext is the **advanced container**. ApplicationContext extends the BeanFactory interface. ApplicationContext provides more facilities than BeanFactory such as integration with spring AOP, message resource handling for i18n etc.

8) What is the difference between constructor injection and setter injection?

No.	Constructor Injection	Setter Injection
1)	No Partial Injection	Partial Injection
2)	Doesn't override the setter property	Overrides the constructor property if both are defined.
3)	Creates new instance if any modification occurs	Doesn't create new instance if you change the property value
4)	Better for too many properties	Better for few properties.

[More details...](#)

9) What is autowiring in spring? What are the autowiring modes?

Autowiring enables the programmer to inject the bean automatically. We don't need to write explicit injection logic.

Let's see the code to inject bean using dependency injection.

```
<bean id="emp" class="com.javatpoint.Employee" autowire="byName" />
```

The autowiring modes are given below:

No.	Mode	Description
1)	no	this is the default mode, it means autowiring is not enabled.
2)	byName	injects the bean based on the property name. It uses setter method.
3)	byType	injects the bean based on the property type. It uses
4)	constructor	It injects the bean using constructor

The "autodetect" mode is deprecated since spring 3.

10) What are the different bean scopes in spring?

There are 5 bean scopes in spring framework.

No.	Scope	Description
1)	singleton	The bean instance will be only once and same instance will be returned by the IOC container. It is the default scope.
2)	prototype	The bean instance will be created each time when requested.
3)	request	The bean instance will be created per HTTP request.
4)	session	The bean instance will be created per HTTP session.
5)	globalsession	The bean instance will be created per HTTP global session. It can be used in portlet context only.

11) In which scenario, you will use singleton and prototype scope?

Singleton scope should be used with EJB **stateless session bean** and prototype scope with EJB **stateful session bean**.

12) What are the transaction management supports provided by spring?

Spring framework provides two type of transaction management supports:

1. **Programmatic Transaction Management**: should be used for few transaction operations.
2. **Declarative Transaction Management**: should be used for many transaction operations.

» Spring JDBC Interview Questions

13) What are the advantages of JdbcTemplate in spring?

Less code: By using the JdbcTemplate class, you don't need to create connection, close connection to execute different queries. You can execute the query directly.

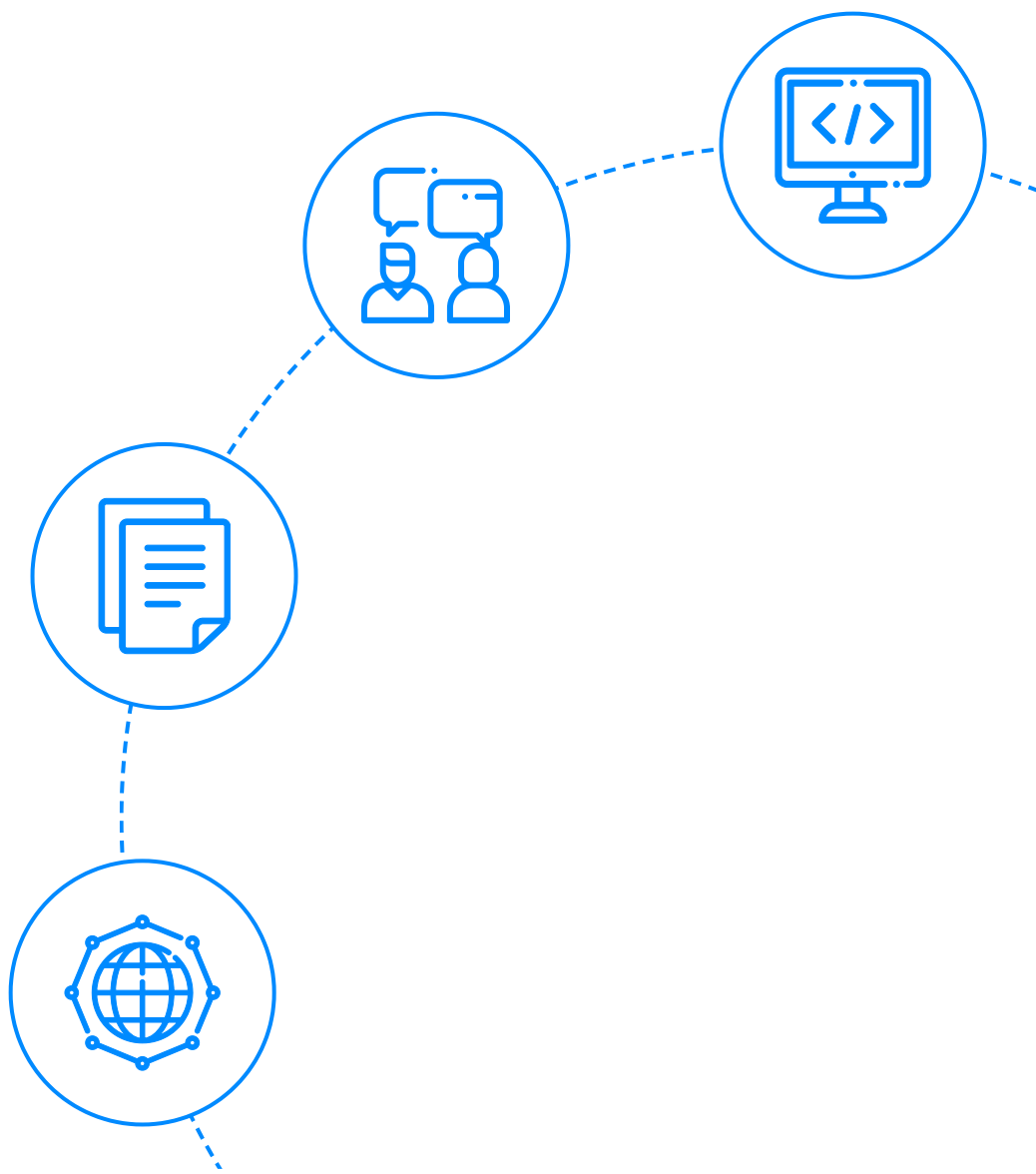
[More details...](#)

14) What are classes for spring JDBC API?



InterviewBit

Spring Security Interview Questions



To view the live version of the page, [click here.](#)

© Copyright by Interviewbit

Contents

Spring Security Interview Questions for Freshers

1. What are some essential features of Spring Security?
2. What is Spring security authentication and authorization?
3. What do you mean by basic authentication?
4. What do you mean by digest authentication?
5. What do you mean by session management in Spring Security?
6. Explain SecurityContext and SecurityContext Holder in Spring security.
7. Explain spring security OAuth2.
8. What do you mean by OAuth2 Authorization code grant type?
9. What is method security and why do we need it?
10. What do you mean by HASHING in spring security?
11. Explain salting and its usage.
12. What is PasswordEncoder?
13. Explain AbstractSecurityInterceptor in spring security?
14. Is security a cross-cutting concern?

Spring Security Interview Questions for Experienced

15. What is SpEL (Spring Expression Language)?
16. Name security annotations that are allowed to use SpEL.
17. Explain what is AuthenticationManager in Spring security.
18. Explain what is ProviderManager in Spring security.

Spring Security Interview Questions for Experienced

(.....Continued)

19. What is JWT?
20. What is Spring Security Filter Chain?
21. Explain how the security filter chain works.
22. Name some predefined filters used in spring security and write their functions.
23. What do you mean by principal in Spring security?
24. Can you explain what is DelegatingFilterProxy in spring security?
25. Can you explain what is FilterChainProxy in spring security?
26. What is the intercept-url pattern and why do we need it?
27. Does order matter in the intercept-url pattern? If yes, then in which order should we write it?
28. State the difference between ROLE_USER and ROLE_ANONYMOUS in a spring intercept-url configuration.
29. State the difference between @PreAuthorize and @Secured in Spring security.
30. State the difference between @Secured and @RolesAllowed.

Let's get Started

Anything on the web like web applications is exposed to the open world of the Internet, they are vulnerable to security threats. Only authorized personnel should have access to Web pages, files, and other classified resources. There are often several layers of security, such as firewalls, proxy servers, JVM security, etc., but, if access is to be controlled, application-level security should also be applied. Hence, Spring Security, a part of the [Spring Framework](#), provides a means for applying a layer of security to Java applications.

What is Spring Security?



Spring Security



Spring Security is essentially just a bunch of servlet filters that enable Java applications to include authentication and authorization functionality. It is one of the most powerful, and highly customizable access-control frameworks (security framework) that provide authentication, authorization, and other security features for Java EE (Enterprise edition) based enterprise applications. The real power of Spring Security lies in its ability to be extended to meet custom needs. Its main responsibility is to authenticate and authorize incoming requests for accessing any resource, including rest API endpoints, MVC (Model-View-Controller) URLs, static resources, etc.

Spring Security Interview Questions for Freshers

1. What are some essential features of Spring Security?

Some essential **features** of Spring Security include:

- Supports authentication and authorization in a flexible and comprehensive manner.
- Detection and prevention of attacks including session fixation, clickjacking, cross-site request forgery, etc.
- Integrate with Servlet API.
- Offers optional integration with Spring Web MVC (Model-View-Controller).
- Java Authentication and Authorization Service (JAAS) is used for authentication purposes.
- Allows Single Sign-On so that users can access multiple applications with just one account (username and password).

2. What is Spring security authentication and authorization?



- **Authentication:** This refers to the process of verifying the identity of the user, using the credentials provided when accessing certain restricted resources. Two steps are involved in authenticating a user, namely identification and verification. An example is logging into a website with a username and a password. This is like answering the question Who are you?
- **Authorization:** It is the ability to determine a user's authority to perform an action or to view data, assuming they have successfully logged in. This ensures that users can only access the parts of a resource that they are authorized to access. It could be thought of as an answer to the question Can a user do/read this?

3. What do you mean by basic authentication?

RESTful web services can be authenticated in many ways, but the most basic one is basic authentication. For basic authentication, we send a username and password using the HTTP [Authorization] header to enable us to access the resource. Usernames and passwords are encoded using base64 encoding (not encryption) in Basic Authentication. The encoding is not secure since it can be easily decoded.

Syntax:

```
Value = username:password
Encoded Value = base64(Value)
Authorization Value = Basic <Encoded Value>
//Example: Authorization: Basic VGVzdFVzZXI6dGVzdDEyMw==
//Decode it'll give back the original username:password UserName:user123
```

4. What do you mean by digest authentication?

RESTful web services can be authenticated in many ways, but advanced authentication methods include digest authentication. It applies a hash function to username, password, HTTP method, and URI in order to send credentials in encrypted form. It generates more complex cryptographic results by using the hashing technique which is not easy to decode.

Syntax:

```
Hash1=MD5(username:realm:password)
Hash2=MD5(method:digestURI)
response=MD5(Hash1:nonce:nonceCount:cnonce:qop:Hash2)
//Example, this got generated by running this example
Authorization: Digest username="TestAdmin", realm="admin-digest-realm", nonce="MTYwMDEv"
```

5. What do you mean by session management in Spring Security?

As far as security is concerned, session management relates to securing and managing multiple users' sessions against their request. It facilitates secure interactions between a user and a service/application and pertains to a sequence of requests and responses associated with a particular user. Session Management is one of the most critical aspects of Spring security as if sessions are not managed properly, the security of data will suffer. To control HTTP sessions, Spring security uses the following options:

- SessionManagementFilter.
- SessionAuthenticationStrategy

With these two, spring-security can manage the following security session options:

- Session timeouts (amount of time a user can remain inactive on a website before the site ends the session.)
- Concurrent sessions (the number of sessions that an authenticated user can have open at once).
- Session-fixation (an attack that permits an attacker to hijack a valid user session).

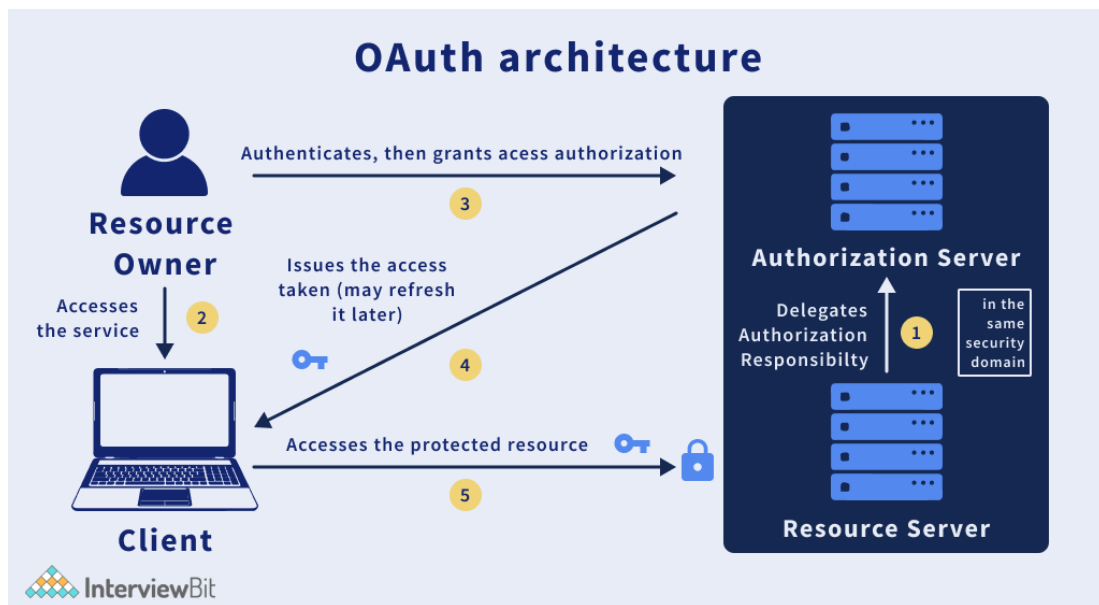
6. Explain SecurityContext and SecurityContextHolder in Spring security.

There are two fundamental classes of Spring Security: SecurityContext and SecurityContextHolder.

- **SecurityContext:** In this, information/data about the currently authenticated user (also known as the principal) is stored. So, in order to obtain a username or any other information about the user, you must first obtain the SecurityContext.
- **SecurityContextHolder:** Retrieving the currently authenticated principal is easiest via a static call to the SecurityContextHolder. As a helper class, it provides access to the security context. By default, it uses a ThreadLocal object to store SecurityContext, so SecurityContext is always accessible to methods in the same thread of execution, even if SecurityContext isn't passed around.

7. Explain spring security OAuth2.

A simple authorization framework, OAuth 2.0, permits client applications to access protected resources via an authorization server. Using it, a client application (third party) can gain limited access to an HTTP service on behalf of the resource owner or on its own behalf.



In OAuth2, four roles are available as shown below:

- **Resource Owner/User:** The owner of a resource, i.e., the individual who holds the rights to that resource.
- **Client:** The application requests an access token (represents a user's permission for the client to access their data/resources), then accesses the protected resource server after receiving the access token.
- **Authorization Server:** After successfully authenticating the resource owner and obtaining authorization, the server issues access tokens to the client.
- **Resource Server:** It provides access to requested resources. Initially, it validates the access tokens, then it provides authorization.

8. What do you mean by OAuth2 Authorization code grant type?

The term "grant type" in OAuth 2.0 refers to the way an application gets an access token. The authorization code flow is one of several types of grants defined by OAuth 2.0. This grant is used by both web applications and native applications to obtain an access token after a user authorizes the application. As opposed to most other grant types, it requires the application to first launch a browser to begin the process/flow. The process involves the following steps:

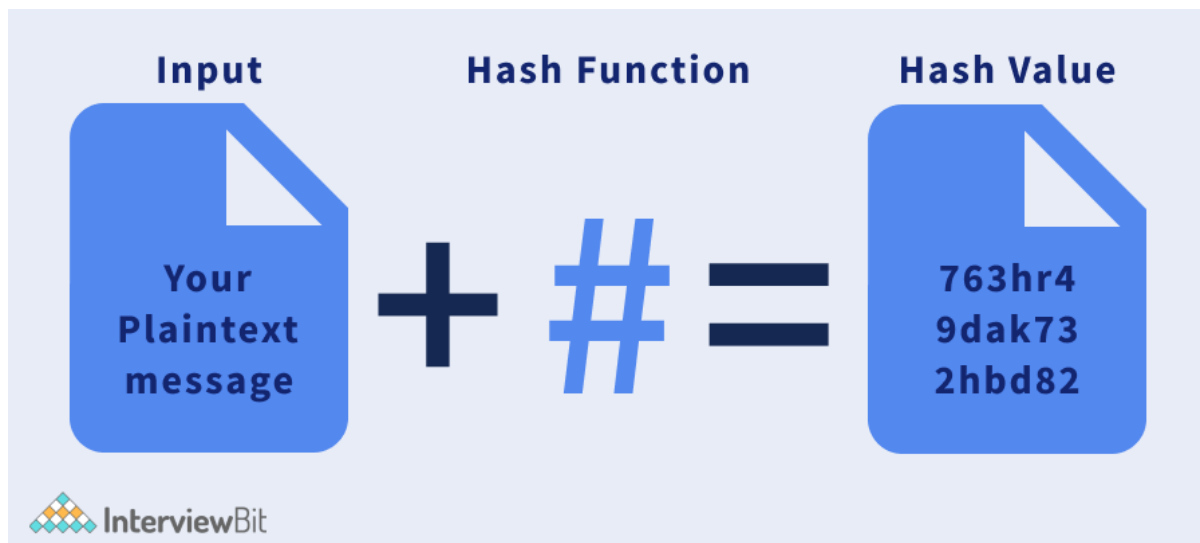
- The application opens a browser to direct the user to an OAuth server.
- Upon seeing the authorization prompt, the user approves the application's request.
- Upon approval, the user is redirected back to the application with an authorization code in the query string.
- Application exchange authorization codes for access tokens.

9. What is method security and why do we need it?

Simply put, Spring method security lets us add or support authorization at the method level. Spring security checks the authorization of the logged-in user in addition to authentication. Upon login, the ROLE of the user is used to determine which user is authorized to access the resource. When creating a new user in WebSecurityConfig, we can specify his ROLE as well. A security measure applied to a method prevents unauthorized users and only allows authentic users. The purpose of method level security is not to facilitate users who have access but to prevent unauthorized users from performing activities beyond their privileges and roles. Method level security is implemented using AOP (**Aspect-Oriented Programming**).

10. What do you mean by HASHING in spring security?

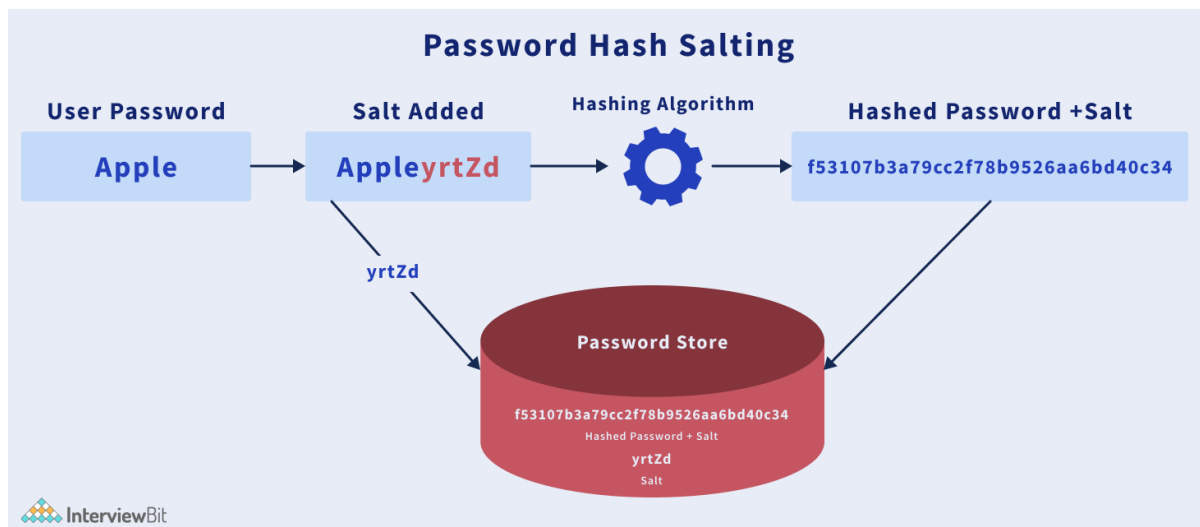
Databases often suffer from security problems when storing passwords. Plain text passwords cannot be stored in your database because then anyone who has access to the database would know the passwords of every user. The solution to this problem is to store encrypted passwords in a database. This is called password hashing.



As part of a general security concept, Hashing involves encoding a string according to the hashing algorithm used. MD4, MD5, SHA (Security Hashing Algorithm) like SHA256 SHA128, etc., are some of the hashing algorithms that can be applied. The hashing method should take the password as input and return a hashed string, which should be stored in a database rather than plain text.

11. Explain salting and its usage.

Spring Security automatically applies salting since version 3.1. Salting is the process of combining random data with a password before password hashing. Salt improves hashing by increasing its uniqueness and complexity without increasing the requirements for users, thereby reducing password attacks. Hashed passwords are then stored in a database, along with salt. Your application will be protected from Dictionary-Attack by using salting. With Salt, you can add an extra string to the password to make it more difficult for hackers to crack it.



12. What is PasswordEncoder?

Password encoding is provided by Spring Security using the PasswordEncoder interface. This interface defines two methods:

- **encode():** It converts a plain password into an encoded form.
- **matches():** It compares an encoded password from the database with a plain password (input by the user) that's been encoded using the same salting and hashing algorithm as the encoded password.

13. Explain AbstractSecurityInterceptor in spring security?

In Spring Security, the AbstractSecurityInterceptor handles the initial authorization of incoming requests. AbstractSecurityInterceptor has two concrete implementations:

- **FilterSecurityInterceptor:** It will authorize all authenticated user requests.
- **MethodSecurityInterceptor:** This is crucial for implementing method-level security. It allows us to secure our program at the method level.

14. Is security a cross-cutting concern?

Spring Security is indeed a cross-cutting concern. Spring security is also using Spring AOP (Aspect Oriented Programming) internally. A cross-cutting concern is one that applies throughout the whole application and affects it all. Below are some cross-cutting concerns related to the enterprise application.

- Logging and tracing
- Transaction management
- Security
- Caching
- Error handling
- Performance monitoring
- Custom Business Rules

Spring Security Interview Questions for Experienced

15. What is SpEL (Spring Expression Language)?

Spring Framework 3.0 introduced Expression Language/ SpEL. In Spring Expression Language (SpEL), queries and manipulations of object graphs are possible at runtime. You can use it with XML and annotation-based Spring configurations. JSP EL, OGNL, MVEL and JBoss EL are some of the expression languages available, but SpEL provides additional features including string template functionality and method invocation.

Example:

```
import org.springframework.expression.Expression;
import org.springframework.expression.ExpressionParser;
import org.springframework.expression.spel.standard.SpelExpressionParser;
public class WelcomeTest
{
    public static void main(String[] args)
    {
        ExpressionParser parser = new SpelExpressionParser();
        Expression exp = parser.parseExpression("'WELCOMetoSPEL'");
        String message = (String) exp.getValue();
        System.out.println(message);
        //OR
        //System.out.println(parser.parseExpression("'Hello SPEL'").getValue());
    }
}
```

Output:

WELCOMetoSPEL

16. Name security annotations that are allowed to use SpEL.

Some security annotations that are allowed to use SpEL include:

- @PreAuthorize
- @PreFilter
- @PostAuthorize
- @PostFilter

These provide expression-based access control. In Spring Security, @PreAuthorize is one of the most powerful annotations that allows you to use SpEL. But the old @Secured annotation cannot use it, for example you cannot write @Secured("hasRole('ROLEADMIN')"), but you can do @PreAuthorize("hasRole('ROLEADMIN')").

17. Explain what is AuthenticationManager in Spring security.

A Spring Security component called `AuthenticationManager` tells "How authentication will happen". Because the how part of this question depends on which authentication provider we are using for our application, an `AuthenticationManager` contains references to all the `AuthenticationProviders`. `AuthenticationManager` is the strategy interface for authentication, which has only one method:

```
public interface AuthenticationManager {  
    Authentication authenticate(Authentication authentication)  
        throws AuthenticationException;  
}
```

`AuthenticationManagers` can perform one of three actions in their `authenticate()` method:

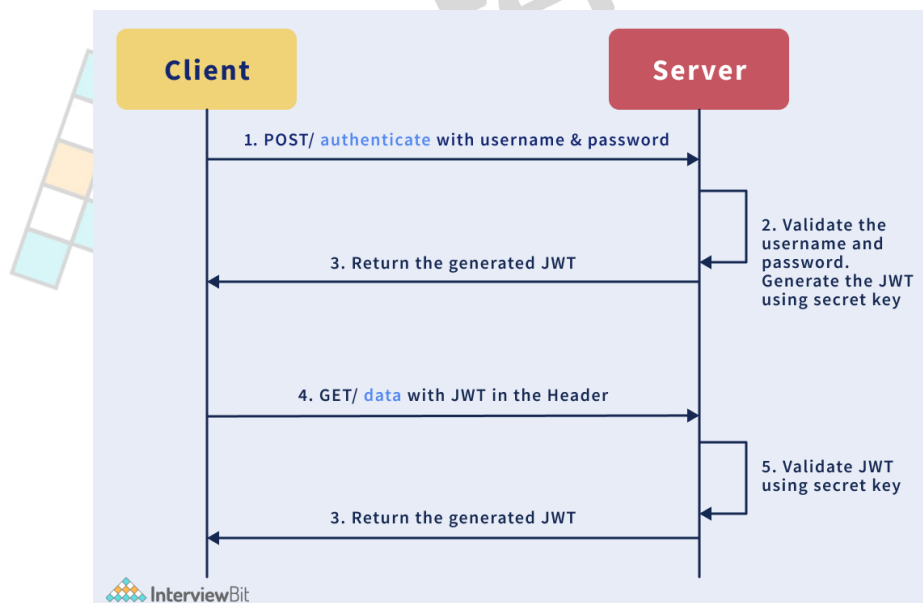
- If it can verify that the input represents a valid principal, it will return an `Authentication` (normally `authenticated=true`).
- If the input is believed to represent an invalid principal, it will throw an `AuthenticationException`.
- If it is unable to decide, it will return `null`.

18. Explain what is `ProviderManager` in Spring security.

The default implementation of `AuthenticationManager` is `ProviderManager`. It does not handle the authentication request itself, rather delegates the authentication process to a list of configured `AuthenticationProviders`. Each authenticationprovider in turn is queried to see if it can handle the authentication request.

19. What is JWT?

JWT (JSON Web Tokens) are tokens that are generated by a server upon user authentication in a web application and are then sent to the client (normally a browser). As a result, these tokens are sent on every HTTP request, allowing the server to verify or authenticate the user's identity. This method is used for authorizing transactions or requests between client and server. The use of JWT does not intend to hide data, but rather ensure its authenticity. JWTs are signed and encoded, instead of encrypted. A cryptographic algorithm is used to digitally sign JWTs in order to ensure that they cannot be altered after they are issued. Information contained in the token is signed by the server's private key in order to ensure integrity.



- Login credentials are sent by the user. When successful, JWT tokens (signed by private key/secret key) are sent back by the server to the client.
- The client takes JWT and inserts it in the Authorization header to make data requests for the user.
- Upon receiving the token from the client, the server simply needs to compare the signature sent by the client to the one it generated with its private key/secret key. The token will be valid once the signatures match.

Three parts make up JSON Web Tokens, separated by a dot (.). The first two (the header and the payload) contain Base64-URL encoded JSON, while the third is a cryptographic signature.

For example:

```
eyJhbGciOiJI1NiJ9.eyJ1IjogdGdfEENvZGVyIn0.5d1p7GmziL2dfecege4mtaqv0_xX4oFUuTDh14K
```

Take a look at each of the sections:

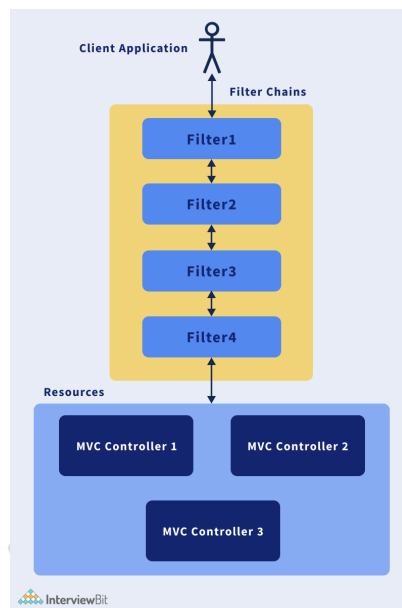
```
eyJhbGciOiJI1NiJ9    #header  
eyJ1IjogdGdfEENvZGVyIn0    #payload  
5d1p7GmziL2dfecege4mtaqv0_xX4oFUuTDh14KuF    #signature
```

20. What is Spring Security Filter Chain?

Spring Security executes most of its security features using the filter chain. Spring security is driven through servlet filters in web applications. A servlet filter intercepts requests before they reach the protected resource (e.g., a Spring controller). As a result, every request for a protected resource will be processed through a spring security filter chain for completing authentication and authorization purposes.

21. Explain how the security filter chain works.

Here's how filters work in a web application:



- **Step 1:** The client first sends a request for a resource (MVC controller). The application container creates a filter chain for handling and processing incoming requests.
- **Step 2:** Each `HttpServletRequest` passes through the filter chain depending upon the request URI. (We can configure whether the filter chains should be applied to all requests or to the specific request URI).
- **Step 3:** For most web applications, filters perform the following functions:
 - Modify or Change the `HttpServletRequest/HttpServletResponse` before it reaches the Spring MVC controller.
 - Can stop the processing of the request and send a response to the client, such as Servlets not allowing requests to specific URI's.

22. Name some predefined filters used in spring security and write their functions.

Filter chains in Spring Security are very complex and flexible. They use services such as UserDetailsService and AuthenticationManager to accomplish their tasks. It is also important to consider their orders since you might want to verify their authenticity before authorizing them. A few of the important security filters from Spring's filter chain are listed below in the order they occur:

- **SecurityContextPersistenceFilter:** Stores the SecurityContext contents between HTTP requests. It also clears SecurityContextHolder when a request is finished.
- **ConcurrentSessionFilter:** It is responsible for handling concurrent sessions. Its purpose is to refresh the last modified time of the request's session and to ensure the session hasn't expired.
- **UsernamePasswordAuthenticationFilter:** It's the most popular authentication filter and is the one that's most often customized.
- **ExceptionTranslationFilter:** This filter resides above FilterSecurityInterceptor in the security filter stack. Although it doesn't perform actual security enforcement, it handles exceptions thrown by the security interceptors and returns valid and suitable HTTP responses.
- **FilterSecurityInterceptor:** It is responsible for securing HTTP resources (web URIs), and raising or throwing authentication and authorization exceptions when access is denied.

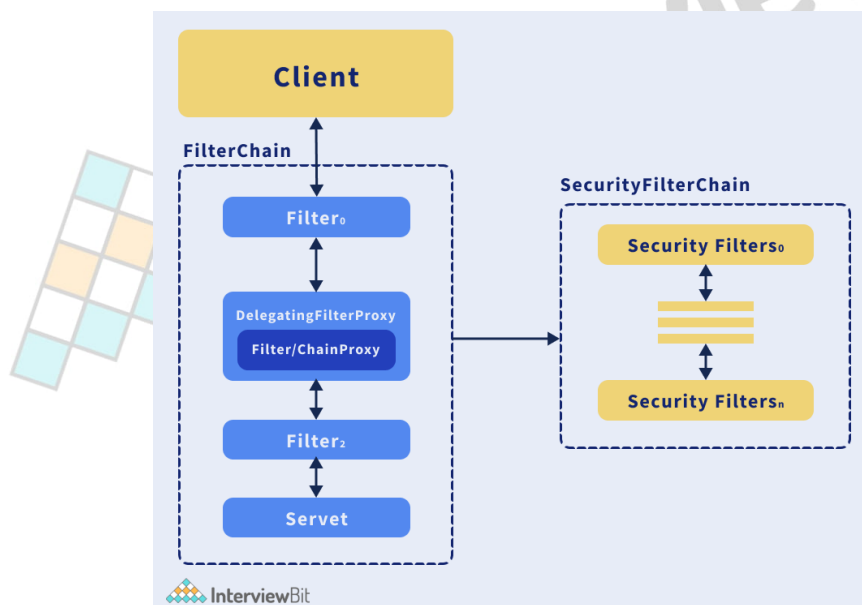
23. What do you mean by principal in Spring security?

The principal is actually the currently logged in user that is using the application. Information/data about the principal (currently authenticated user) is stored in the SecurityContext of the application. As a helper class, SecurityContextHolder provides access to the security context. By default, it uses a ThreadLocal object to store SecurityContext, so SecurityContext is always accessible to methods in the same thread of execution, even if SecurityContext isn't passed around explicitly.

24. Can you explain what is DelegatingFilterProxy in spring security?

A servlet filter must be declared in the web.xml file so that it can be invoked before the request is passed on to the actual Servlet class. DelegatingFilterProxy is a servlet filter embedded in the spring context. It acts as a bridge between web.xml (web application) and the application context (Spring IoC Container).

DelegatingFilterProxy is a proxy that delegates an incoming request to a group of filters (which are not managed as spring beans) provided by the Spring web framework. It provides full access to the Spring context's life cycle machinery and dependency injection.



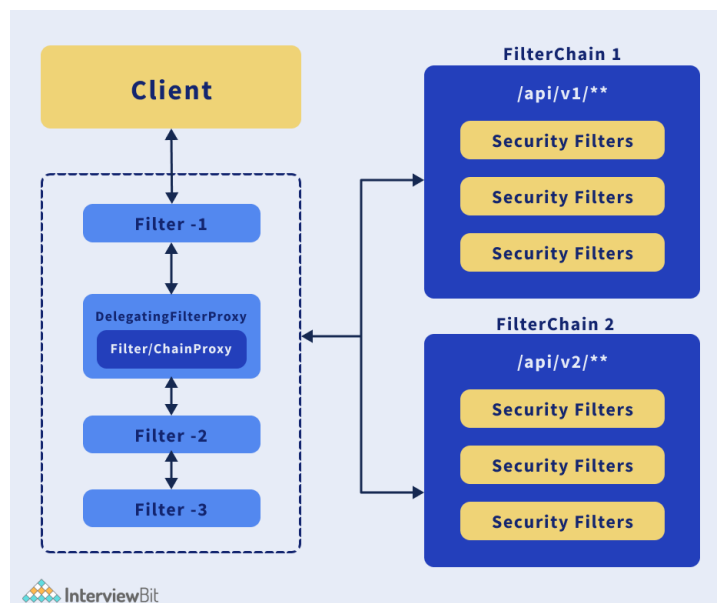
Whenever a request reaches the web application, the proxy ensures that the request is delegated to Spring Security, and, if everything goes smoothly, it will ensure that the request is directed to the right resource within the web application. The following example demonstrates how to configure the DelegatingProxyFilter in web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>
      org.springframework.web.filter.DelegatingFilterProxy
    </filter-class>
  </filter>

  <filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

25. Can you explain what is FilterChainProxy in spring security?

FilterChainProxy is another servlet filter designed to invoke the appropriate filters based on the path of the incoming request. It contains information about the security filters that make up the security filter chain. It is not directly executed, but it is started by the DelegatingFilterProxy.



26. What is the intercept-url pattern and why do we need it?

<Intercept-url> is used to configure authorizations or access-controls in a Spring Security application. It is used to restrict access to a particular URL. The majority of web applications using Spring Security usually have just a few intercept-URLs because their security needs are quite less.

Example: Basic Spring security using intercept URL

```
<http realm="Example" use-expressions="false">
  <intercept-url pattern="/index.jsp" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
  <intercept-url pattern="/login.jsp*" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
  <intercept-url pattern="/admin/*" access="ROLE_ADMIN"/>
  <intercept-url pattern="/trade/*" access="ROLE_TRADER"/>
  <intercept-url pattern="/*" access="ROLE_USER,ROLE_ADMIN,ROLE_TRADER"/>
</http-basic>
```

In this case, index.jsp and admin.jsp can be accessed without authentication. Anything with admin in the URL requires ROLE_ADMIN access, and anything with trade in the URL requires ROLE_TRADER access.

27. Does order matter in the intercept-url pattern? If yes, then in which order should we write it?

Yes, ordering is crucial when we have multiple intercept-URL patterns. Multiple intercept URLs should be written from more specific to less specific. As intercept-URL patterns are processed in the order they appear in a spring security configuration file, the URL must match the right pattern.

28. State the difference between ROLE_USER and ROLE_ANONYMOUS in a spring intercept-url configuration.

- **ROLE_USER:** It has no relevance unless you assign it to your users as soon as they are authenticated. You are responsible for loading the roles (authorities) for each authenticated user.
- **ROLE_ANONYMOUS:** When a configuration uses Spring Security's "anonymous authentication" filter, ROLE_ANONYMOUS is the default role assigned to an anonymous (unauthenticated) user. ROLE_ANONYMOUS is enabled by default. However, it would be better if you used the expression `isAnonymous()` instead, which has the same meaning.

29. State the difference between @PreAuthorize and @Secured in Spring security.

A variety of security options are available with Spring Framework. This framework offers many useful tools or methods for securing applications. In order to provide method-level security, @Secured and @PreAuthorize are the most commonly used annotations. Compared to @Secured, @PreAuthorize is quite new but becoming well known very fast. There aren't many differences between @Secured and @PreAuthorize; they're nearly identical. However, @PreAuthorize is considerably more powerful than @Secured.

@PreAuthorize	@Secured
We can access the methods and properties of SecurityExpressionRoot while using @PreAuthorize.	We can access the methods and properties of SecurityExpressionRoot while using @Secured.
It can work with Spring EL.	It cannot work with Spring EL.
It supports multiple roles in conjunction with AND operator. For example: <pre>@PreAuthorize("hasRole('ROLE_role1') and hasRole('ROLE_role2')")</pre>	It does not support multiple roles in conjunction with AND operator. If more than one role is specified, the OR operator is used. For example: <pre>@Secured("ROLE_role1,ROLE_role2")</pre>
Add the following line to spring-security.xml and spring boot to enable @PreAuthorize and @PostAuthorize annotations in your code: <pre>XML: <global-method-security pre-post-annotations="enabled"/></pre> <pre>Spring boot: @EnableGlobalMethodSecurity(prePostEnabled = true)</pre>	Add the following line to spring-security.xml and spring boot to enable @Secured and @RolesAllowed annotations in your code: <pre>XML: <global-method-security secured-annotations="enabled"/></pre> <pre>Spring boot: @EnableGlobalMethodSecurity(securedEnabled = true)</pre>

30. State the difference between @Secured and @RolesAllowed.

@RolesAllowed: It is a Java standard annotation (JSR250) (i.e., not only spring security). Because this annotation only supports role-based security, it is more limited than the @PreAuthorize annotation. To enable the @RolesAllowed annotation in your code, add the following line to spring-security.xml and spring boot.

```
XML: <global-method-security jsr250-annotations="enabled"/>
```

```
Spring boot: @EnableGlobalMethodSecurity(jsr250Enabled = true)
```

@Secured: It is a Spring specific annotation. There is more to it than just role-based security. It secures methods implemented by beans (objects whose life-cycle is managed by the Spring IoC). However, Spring Expression Language (SpEL) is not supported for defining security constraints. To enable the @Secured annotation in your code, add the following line to spring-security.xml and spring boot.

```
XML: global-method-security secured-annotations="enabled"/>
```

```
Spring boot: @EnableGlobalMethodSecurity(securedEnabled=true)
```

Conclusion

Spring Security is one of the most popular, powerful, and highly customizable access-control frameworks (security framework) that provide authentication, authorization, and other security features for enterprise applications. In this article, we have compiled a comprehensive list of Spring Security Interview questions, which are typically asked during interviews. In addition to checking your existing Spring Security skills, these questions serve as a good resource for reviewing some important concepts before you appear for an interview. It is suitable for both freshers as well as experienced developers and tech leads.

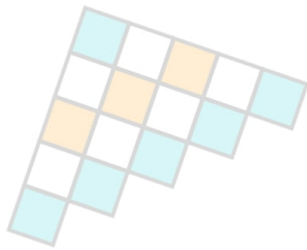
Additional Useful Resources

[Interview questions on Java](#)

[Interview questions on Spring Boot](#)

[Spring MVC vs Spring Boot](#)

[Spring vs Spring Boot](#)



InterviewBit

Links to More Interview Questions

[C Interview Questions](#)

[Php Interview Questions](#)

[C Sharp Interview Questions](#)

[Web Api Interview Questions](#)

[Hibernate Interview Questions](#)

[Node Js Interview Questions](#)

[Cpp Interview Questions](#)

[Oops Interview Questions](#)

[Devops Interview Questions](#)

[Machine Learning Interview Questions](#)

[Docker Interview Questions](#)

[Mysql Interview Questions](#)

[Css Interview Questions](#)

[Laravel Interview Questions](#)

[Asp Net Interview Questions](#)

[Django Interview Questions](#)

[Dot Net Interview Questions](#)

[Kubernetes Interview Questions](#)

[Operating System Interview Questions](#)

[React Native Interview Questions](#)

[Aws Interview Questions](#)

[Git Interview Questions](#)

[Java 8 Interview Questions](#)

[Mongodb Interview Questions](#)

[Dbms Interview Questions](#)

[Spring Boot Interview Questions](#)

[Power Bi Interview Questions](#)

[Pl Sql Interview Questions](#)

[Tableau Interview Questions](#)

[Linux Interview Questions](#)

[Ansible Interview Questions](#)

[Java Interview Questions](#)

[Jenkins Interview Questions](#)

20 Spring REST Web Service Interview Questions

Here are a couple of frequently asked questions about using REST web services in the Spring Framework.

1. What does REST stand for?

REST stands for the REpresentational State Transfer, which uses the HTTP protocol to send data from the client to the server, e.g. a book in the server can be delivered to the client using JSON or XML. However, if you are not familiar with REST, I suggest you to first check out the [REST API design and development](#) to better understand it.

2. What is a resource?

A resource is how data is represented in the REST architecture. By exposing entities as the resource, it allows a client to read, write, modify, and create resources using HTTP methods, for example, [GET](#), [POST](#), [PUT](#), DELETE, etc.

3. What are safe REST operations?

REST API uses HTTP methods to perform operations. Some of the HTTP operations, which doesn't modify the resource at the server, are known as safe operations, including GET and HEAD. On the other hand, [PUT](#), POST, and DELETE are unsafe, because they modify the resource on the server.

4. What are idempotent operations? Why is idempotency important?

There are some HTTP methods — like GET — that produce the same response no matter how many times you use them, sending multiple GET request to the same [URI](#) will result in same response without any side-effect. Hence, this is known as idempotent.

On the other hand, the [POST is not idempotent](#), because if you send multiple POST request, it will result in multiple resource creation on the server, but, again, PUT is idempotent, if you are using it to update the resource.

Even multiple PUT requests can be used to update a resource on a server and will give the same end result. You can take a [HTTP Fundamentals](#) course by Pluralsight to learn more about idempotent methods of HTTP protocol and HTTP in general.

5. Is REST scalable and/or interoperable?

Yes, [REST](#) is scalable and interoperable. It doesn't mandate a specific choice of technology either at client or server end. You can use [Java](#), [C++](#), [Python](#), or [JavaScript](#) to create RESTful web services and consume them at the client end. I suggest you read a good book on REST API, like [RESTful Web Services](#) to learn more about REST.

6. What are the advantages of the RestTemplate? (answer)

The `RestTemplate` class is an implementation of [the Template method pattern](#) in the Spring framework. Similar to other popular template classes, like the `JdbcTemplate` or `JmsTemplate`, it also simplifies the interaction with RESTful web services on the client side. You can use it to consume a RESTful web service very easily, as shown in this `RestTemplate` example.

7. Which HTTP methods does REST use?

REST can use any HTTP methods, but the most popular ones are GET for retrieving a resource, POST for creating a resource, [PUT for updating resource](#), and DELETE for removing a resource from the server.

8. What is an HttpMessageConverter in Spring REST?

An `HttpMessageConverter` is a [strategy interface](#) that specifies a converter that can convert from and to HTTP requests and responses. Spring REST uses this interface to convert HTTP responses to various formats, for example, JSON or XML.

Each `HttpMessageConverter` implementation has one or several MIME Types associated with it. Spring uses the "Accept" header to determine the content type that the client is expecting.

It will then try to find a registered [HttpMessageConverter](#) that is capable of handling that specific content-type and use it to convert the response into that format before sending it to the client. If you are new to Spring MVC, see this [Spring 5: Beginner to Guru](#) resource to learn the basics.

9. How to create a custom implementation of the `HttpMessageConverter` to support a new type of request/responses? ([answer](#))

You just need to create an implementation of the `AbstractHttpMessageConverter` and register it using the `WebMvcConfigurerAdapter#extendMessageConverters()` method with the classes that generate a new type of request/response.

10. Is REST normally stateless? ([answer](#))

Yes, REST API should be stateless, because it is based on HTTP, which is also stateless. A request in REST API should contain all the details required to process it. It should not rely on previous or next requests or some data maintained at the server end, like sessions. The REST specification puts a constraint to make it stateless, and you should keep that in mind while designing your REST API.

11. What does `@RequestMapping` annotation do? ([answer](#))

The `@RequestMapping` annotation is used to map web requests to Spring Controller methods. You can map a request based upon HTTP methods, e.g. GET, POST, and various other parameters.

For example, if you are developing a RESTful web service using Spring, then you can use, produce, and consume property along with media type annotations to indicate that this method is only used to produce or consume JSON, as shown below:

```
@RequestMapping (method = RequestMethod.POST, consumes="application/json")
public Book save(@RequestBody Book aBook) {
    return bookRepository.save(aBook);
}
```

Similarly, you can create other handler methods to produce JSON or XML. If you are not familiar with these annotations, then I suggest you join this [Spring MVC For Beginners](#) course on Udemy to learn the basics.

12. Is `@Controller` a stereotype? Is `@RestController` a stereotype? ([answer](#))

Yes, both `@Controller` and `@RestController` are stereotypes. The `@Controller` is actually a specialization of

Spring's `@Component` stereotype annotation. This means that the class annotated with the `@Controller` will also be automatically detected by the Spring container, as part of the container's component scanning process.

And, the `@RestController` is a specialization of the `@Controller` for the RESTful web service. It not only combines the `@ResponseBody` and `@Controller` annotations, but it also gives more meaning to your controller class to clearly indicate that it deals with RESTful requests.

Your Spring Framework may also use this annotation to provide some more useful features related to REST API development in future.

13. What is the difference between `@Controller` and `@RestController`? ([answer](#))

There are many differences between the `@Controller` and `@RestController` annotations, as discussed in my earlier article (see the answer for more!), but the most important one is that with the `@RestController` you get the `@ResponseBody` annotation automatically, which means you don't need to separately annotate your handler methods with the `@ResponseBody` annotation.

This makes the development of RESTful web services easier using Spring. You can see here to learn more about [Spring Boot](#) and how it can help you to create Spring MVC based web applications.

```
@RestController
@EnableAutoConfiguration
public class Example {

    public static void main(String[] args) {
        SpringApplication.run(Example.class, args);
    }

    @RequestMapping("/")
    public String home() {
        return "Hello World!";
    }
}
```

14. When do you need `@ResponseBody` annotation in Spring MVC? ([answer](#))

The `@ResponseBody` annotation can be put on a method to indicate that the return type should be written directly to the HTTP response body (and not placed in a Model, or interpreted as a view name).

For example:

```
@RequestMapping(path = "/hello", method = RequestMethod.PUT)

@ResponseBody

public String helloWorld() {

    return "Hello World";

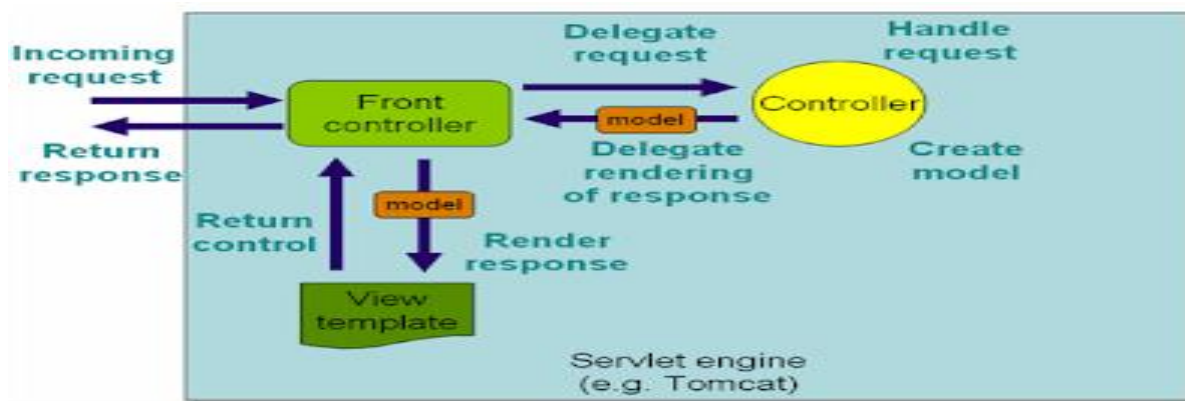
}
```

Alternatively, you can also use the `@RestController` annotation instead of the `@Controller` annotation. This will remove the need for using `@ResponseBody` because, as discussed in the previous answer, it comes automatically with the `@RestController` annotation.

15. What does `@PathVariable` do in Spring MVC? Why it's useful in REST with Spring? ([answer](#))

This is one of the useful annotations from Spring MVC that allows you to read values from the URI, like query parameter. It's particularly useful in case of creating RESTful web service using Spring, because, in REST, resource identifiers are part of the URI. This question is normally asked by experienced Spring MVC developers with 4 to 6 years of experience.

For example, this [URL](#) can be helpful if you want to learn how to extract the id, then you can use the `@PathVariable` annotation of Spring MVC. If you are not familiar with Spring MVC annotations, then [Spring MVC For Beginners: Build Java Web App in 25 Steps](#) is a good place to start.



16. What is the HTTP status return code for a successful DELETE statement? ([answer](#))

There is no strict rule about what status code your REST API should return to after a successful DELETE. It can return 200 Ok or 204 No Content.

In general, if the DELETE operation is successful, the response body is empty, return 204. If the DELETE request is successful and the response body is NOT empty, return 200.

17. What does CRUD mean? ([answer](#))

CRUD is a short form of Create, Read, Update, and Delete. In REST API, the POST is used to create a resource, GET is used to read a resource, [PUT](#) is used to updated a resource, and DELETE is used to remove a resource from the server. This one is another beginner level Spring MVC question common amongst 1 to 3 years as an experienced programmer.

18. Where do you need @EnableWebMVC? ([answer](#))

The `@EnableWebMvc` annotation is required to enable Spring MVC when Java configuration is used to configure Spring MVC instead of XML. It is equivalent to `<mvc: annotation-driven>` in an XML configuration.

It enables support for the `@Controller`-annotated classes that use `@RequestMapping` to map incoming requests to handler methods that are not already familiar with Spring's support for Java configuration. The [Spring Master Class](#) on Udemy is a good place to start.

19. When do you need `@ResponseStatus` annotation in Spring MVC? ([answer](#))

This is a good question for 3 to 5 years as an experienced Spring developer. The `@ResponseStatus` annotation is required during error handling in [Spring MVC](#) and REST. Normally, when an error or exception is thrown at the server side, the web server returns a blanket HTTP status code 500 — Internal server error.

This may work for a human user but not for REST clients. You need to send them the proper status code, like 404, if the resource is not found. That's where you can use the `@ResponseStatus` annotation, which allows you to send custom HTTP status codes along with proper error message in case of an exception.

In order to use it, you can create custom exceptions and annotate them using the `@ResponseStatus` annotation and proper HTTP status code and reason.

When such exceptions are thrown from the controller's handler methods and not handled anywhere else, then the appropriate HTTP response with the proper HTTP status code is sent to the client.

For example, if you are writing a [RESTful web service](#) for a library that provides book information, then you can use `@ResponseStatus` to create an exception that returns the HTTP response code 404 when a book is not found instead of the Internal Server Error (500), as shown below:

```
@ResponseStatus(value=HttpStatus.NOT_FOUND, reason="No such Book") // 404

public class BookNotFoundException extends RuntimeException {

    // ...

}
```

If this exception is thrown from any handler method, then the HTTP error code 404 with the reason "No such Book" will be returned to the client.

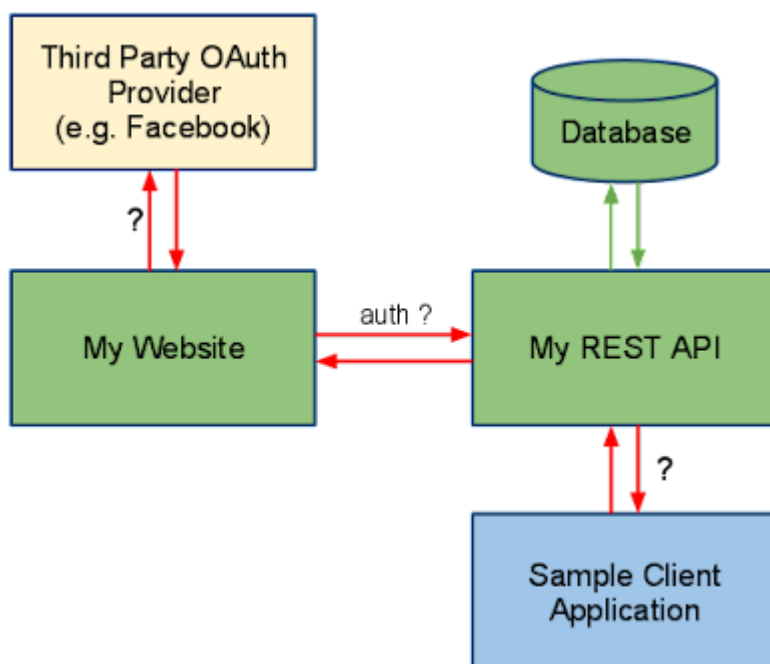
If you are not familiar with the basics concepts of Spring MVC, Security, and REST, I suggest you go through these [REST with Spring](#) and [Learn Spring Security](#) courses to gain some experience before your next job interview. These two courses are specially designed to provide you with

some real-world experience to boost both your knowledge and experience with Spring MVC, REST, and Spring Security.

20. Is REST secure? What can you do to secure it? ([answer](#))

This question is mostly asked by experienced Java programmers with about 2 to 5 years of experience with both REST and Spring. Security is a broad term; it could mean security of message, which is provided by encryption or access restriction that are provided using authentication and authorization. REST is normally not secure, but you can secure it by using Spring Security.

At the very least, you can enable the HTTP basic authentication by using HTTP in your Spring Security configuration file. Similarly, you can expose your REST API using [HTTPS](#), if the underlying server supports HTTPS.



21. Does REST work with transport layer security (TLS)? ([answer](#))

Transport Layer Security (TLS) is used for secure communication between the client and server. It is the successor of SSL (Secure Socket Layer). Since HTTPS can work with both SSL and TLS, REST can also work with TLS.

Actually, in REST, it is up to the server to implement security protocols. The same RESTful web service can be accessed using HTTP and HTTPS, if the server supports [SSL](#).

If you are using Tomcat, you can learn more about how to enable SSL in Tomcat.

22. Do you need Spring MVC in your classpath for developing RESTful Web Service? ([answer](#))

This question is often asked by Java programmers with 1 to 2 years of experience in Spring. The short answer is: **yes** — you need Spring MVC in your Java application's classpath to develop RESTful web services using the Spring framework.

It's actually Spring MVC that provides all useful annotations, like `@RestController`, `@ResponseCode`, `@ResponseBody`, `@RequestBody`, and `@PathVariable` (see [REST with Spring](#)). Hence, you must use `spring-mvc.jar` or the appropriate Maven entry in your pom.xml