**Interview Questions**

**SPRING:**

1. **What is Spring?**

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

1. **What are the advantages of Spring framework?**

* Predefined Templates
* Loose Coupling
* Easy to test
* Lightweight
* Fast Development
* Powerful Abstraction
* Declarative support

1. **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.

1. **What are the benefits of IOC?**

* It minimizes the amount of code in your application.
* It makes your application easy to test as it doesn't require any singletons or JNDI lookup mechanisms in your unit test cases.
* Loose coupling is promoted with minimal effort and least intrusive mechanism.
* IOC containers support eager instantiation and lazy loading of services.

1. **What is the role of IOC container in Spring?**

* create the instance
* configure the instance, and
* assemble the dependencies
* The Spring IoC creates the objects, wire them together, configure them, and manage their complete lifecycle from creation till destruction. The Spring container uses dependency injection (DI) to manage the components that make up an application.

1. **What are types of IOC container in spring?**

* BeanFactory
* ApplicationContext

1. **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 etc.
* **Bean Factory container:** This is the simplest container providing basic support for DI .The BeanFactory is usually preferred where the resources are limited like mobile devices or applet based applications
* **Spring ApplicationContext Container:** This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners.

1. **What is the difference between constructor injection and setter injection?**

* Constructor Injection - No Partial Injection
* Constructor Injection – doesn’t override the setter property
* Constructor Injection – Creates new instance if any modification occurs
* Constructor Injection – Better for too many properties
* **Constructor-based dependency injection:** Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.
* Setter Injection – Partial Injection
* Setter Injection – Overrides the constructor property if both are defined.
* Setter Injection – Doesn’t create new instance if you change the property value.
* Setter Injection – Better for few properties.
* **Setter-based dependency injection:** Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

1. **What is autowiring in Spring? What are the autowiring mode?**

* Autowiring enables the programmer to inject the bean automatically. We don't need to write explicit injection logic.
* Ex. <bean id="emp" **class**="com.javatpoint.Employee" autowire="byName" />
* **4 autowiring modes:**
  + **no** - this is the default mode, it means autowiring is not enabled. This is default setting which means no autowiring and you should use explicit bean reference for wiring. You have nothing to do special for this wiring.
  + **byname** - injects the bean based on the property name. It uses setter method. Autowiring by property name. Spring container looks at the properties of the beans on which autowire attribute is set to byName in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file. (ex. For annotations @Autowired and @Qualifier in the property)
  + **byType** - injects the bean based on the property type. It uses setter method. Autowiring by property datatype. Spring container looks at the properties of the beans on which autowire attribute is set to byType in the XML configuration file. It then tries to match and wire a property if its type matches with exactly one of the beans name in configuration file. If more than one such beans exist, a fatal exception is thrown. (ex. For annotations @Autowired to instance variables)
  + **constructor** - It injects the bean using constructor. Similar to byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised.
  + **autodetect** - Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by byType.

1. **What are Spring beans?**

* The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container, for example, in the form of XML <bean/> definitions.

1. **What are the 3 Types of configuration metadata of Spring Container?**

* XML based configuration file.
* Annotation-based configuration
* Java-based configuration (Also uses Component Scan using the annotation @ComponentScan)

1. **What are the different bean scopes in Spring?**

* **singleton** - The bean instance will be only once and same instance will be returned by the IOC container. It is the default scope.
* **prototype** - The bean instance will be created each time when requested.
* **request** - The bean instance will be created per HTTP request.
* **session** - The bean instance will be created per HTTP session.
* **globalsession** - The bean instance will be created per HTTP global session. It can be used in portlet context only.

1. **What is the default scope of bean in Spring framework?**

* The default scope of bean is Singleton for Spring framework.

1. **Is Singleton beans thread safe in Spring framework?**

* No, singleton beans are not thread-safe in Spring framework.

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

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

1. **How can you inject Java Collection in Spring?**

* **<list>**: This helps in wiring i.e. injecting a list of values, allowing duplicates.
* **<set>**: This helps in wiring a set of values but without any duplicates.
* **<map>**: This can be used to inject a collection of name-value pairs where name and value can be of any type.
* **<props>**: This can be used to inject a collection of name-value pairs where the name and value are both Strings.

1. **In Annotation-based container configuration, how do you turn on the annotation wiring?**

* Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file by configuring <context:annotation-config/>

1. **What does @Required annotation mean?**

* This annotation simply indicates that the affected bean property must be populated at configuration time, through an explicit property value in a bean definition or through autowiring. The container throws BeanInitializationException if the affected bean property has not been populated.

1. **What does @Autowired annotation mean?**

* This annotation provides more fine-grained control over where and how autowiring should be accomplished. The @Autowired annotation can be used to autowire bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

1. **What does @Qualifier annotation mean?**

* There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property, in such case you can use @Qualifier annotation along with @Autowired to remove the confusion by specifying which exact bean will be wired.

1. **What are JSR-250 Annotations?**

* Spring has JSR-250 based annotations which include @PostConstruct, @PreDestroy and @Resource annotations.
  + **@PostConstruct**: This annotation can be used as an alternate of initialization callback.
  + **@PreDestroy**: This annotation can be used as an alternate of destruction callback.
  + **@Resource**: This annotation can be used on fields or setter methods. The @Resource annotation takes a 'name' attribute which will be interpreted as the bean name to be injected. You can say, it follows by-name autowiring semantics.

1. **What is Spring Java-based configuration? Give some annotation example.**

* Java based configuration option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations.
* For example: Annotation **@Configuration** indicates that the class can be used by the Spring IoC container as a source of bean definitions. The **@Bean** annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context.

1. **How JDBC can be used more efficiently in Spring framework?**

* JDBC can be used more efficiently with the help of a template class provided by spring framework called as JdbcTemplate.

1. **How JdbcTemplate can be used?**

* With use of Spring JDBC framework the burden of resource management and error handling is reduced a lot. So it leaves developers to write the statements and queries to get the data to and from the database. JdbcTemplate provides many convenience methods for doing things such as converting database data into primitives or objects, executing prepared and callable statements, and providing custom database error handling.

**SPRING MVC**

1. **What is Spring MVC framework?**

* The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

1. **What are the advantages of Spring MVC framework?**

* Clear separation of roles -- controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver, etc. Each role can be fulfilled by a specialized object.
* Powerful and straightforward configuration of both framework and application classes as JavaBeans.
* Reusable business code -- no need for duplication. You can use existing business objects as command or form objects instead of mirroring them in order to extend a particular framework base class.
* Customizable binding and validation
* Customizable handler mapping and view resolution
* Customizable locale and theme resolution
* A JSP form tag library, introduced in Spring 2.0, that makes writing forms in JSP pages much easier. etc.

1. **What is a DispatcherServlet and ContextLoaderListener?**

* The Spring Web MVC framework is designed around a DispatcherServlet that handles all the HTTP requests and responses.
* By default, DispatcherServlet loads its configuration file using <servlet\_name>-servlet.xml.
* ContextLoaderListener reads the spring configuration file (with value given against “**contextConfigLocation**” in web.xml), parse it and loads the beans defined in that config file

1. **What is WebApplicationContext?**

* The *WebApplicationContext* is an extension of the plain *ApplicationContext* that has some extra features necessary for web applications. It differs from a normal *ApplicationContext* in that it is capable of resolving themes, and that it knows which servlet it is associated with.

1. **What is the front controller class of Spring MVC?**

* A front controller is defined as “a controller which handles all requests for a Web Application.” **DispatcherServlet (actually a servlet) is the front controller in Spring MVC that intercepts every request and then dispatches/forwards requests to an appropriate controller.**
* When a web request is sent to a Spring MVC application, dispatcher servlet first receives the request. Then it organizes the different components configured in Spring’s web application context (e.g. actual request handler controller and view resolvers) or annotations present in the controller itself, all needed to handle the request.

1. **What is @Controller annotation?**

* The *@Controller* annotation indicates that a particular class serves the role of a controller. Spring does not require you to extend any controller base class or reference the Servlet API.

1. **What is @RequestMapping annotation?**

* *@RequestMapping* annotation is used to map a URL to either an entire class or a particular handler method.

1. **How can we use Spring to create Restful Web Service returning JSON response?**

* For adding JSON support to your spring application, you will need to **add Jackson dependency** in first step.
* All you have to do is return JAXB annotated object from method and use @ResponseBody annotation on this return type.
* Alternatively, you can use @RestController annotation in place of @Controller annotation. This will remove the need to using @ResponseBody. **@RestController = @Controller + @ResponseBody**

1. **Can we have multiple Spring configuration files?**

* YES. **You can have multiple spring context files**. There are two ways to make spring read and configure them.
  + Specify all files in web.xml file using **contextConfigLocation** init parameter.
  + OR, you can **import them into existing configuration file** you have already configured.

1. **Difference between <context:annotation-config> and <context:component-scan>?**

* First big difference between both tags is that <context:annotation-config> is **used to activate applied annotations in already registered beans in application context**. Note that it simply does not matter whether bean was registered by which mechanism e.g. using <context:component-scan> or it was defined in application-context.xml file itself
* Second difference is driven from first difference itself. It **registers the beans defined in config file into context + it also scans the annotations inside beans and activate them**. So <context:component-scan> does what<context:annotation-config> does, but additionally it scan the packages and register the beans in application context.
* **<context:annotation-config> = Scanning and activating annotations in “already registered beans”.**
* **<context:component-scan> = Bean Registration + Scanning and activating annotations**

1. **Difference between @Component, @Controller, @Repository, and @Service annotations?**

* The @Component annotation marks a java class as a bean so the component-scanning mechanism of spring can pick it up and pull it into the application context
* The @Repository annotation is a specialization of the @Component annotation with similar use and functionality. In addition to importing the DAOs into the DI container, it also makes the unchecked exceptions (thrown from DAO methods) eligible for translation into Spring DataAccessException.
* The @Service annotation is also a specialization of the component annotation. It doesn’t currently provide any additional behavior over the @Component annotation, but it’s a good idea to use @Service over @Component in service-layer classes because it specifies intent better.
* @Controller annotation marks a class as a Spring Web MVC controller. It too is a @Component specialization, so beans marked with it are automatically imported into the DI container. When you add the @Controller annotation to a class, you can use another annotation i.e. @RequestMapping; to map URLs to instance methods of a class.

1. **What does the ViewResolver class?**

* ViewResolver is an interface to be implemented by objects that can resolve views by name. There are plenty of ways using which you can resolve view names. These ways are supported by various in-built implementations of this interface. Most commonly used implementation is InternalResourceViewResolver class. It defines **prefix** and**suffix** properties to resolve the view component.

**SPRING INTEGRATION TEST**

1. **What is SpringJUnit4ClassRunner?**

* A custom extension of [JUnit4ClassRunner](http://junit.sourceforge.net/javadoc_40/org/junit/internal.runners.JUnit4ClassRunner.html?is-external=true) which provides functionality of the Spring TestContext Framework to standard JUnit 4.4+ tests by means of the [TestContextManager](http://docs.spring.io/spring-framework/docs/2.5.6/api/org/springframework/test/context/TestContextManager.html" \o "class in org.springframework.test.context) and associated support classes and annotations
* If placed inside @RunWith annotation in a JUnit class, it indicates that the class should use Spring's JUnit facilities

1. **What is MockMvc?**

* **Main entry point for server-side Spring MVC test support.**
* Will mock the entire Spring MVC infrastructure. We just need the Spring application context to create it.

1. **What is MockMvcBuilders.webAppContextSetup(WebApplicationContext context)?**

* Build a MockMvc instance using the given, fully initialized (i.e., refreshed) WebApplicationContext.
* The DispatcherServlet will use the context to discover Spring MVC infrastructure and application controllers in it. The context must have been configured with a ServletContext.

1. **What is MockMvcRequestBuilders and MockMvcResultMatchers?**

* **MockMvcRequestBuilders** are Static factory methods for RequestBuilders.
* **MockMvcResultMatchers** are Static factory methods for ResultMatcher-based result actions.

*@Test public void helloWorld() throws Exception {*

*mockMvc.perform(*

*MockMvcRequestBuilders.get("/hello").accept(MediaType.TEXT\_PLAIN))*

*.andExpect(MockMvcResultMatchers.status().isOk())*

*.andExpect(MockMvcResultMatchers.content().contentType(MediaType.TEXT\_PLAIN))*

*.andExpect(MockMvcResultMatchers.content().string("Hello World!"));*

*}*

is the same as,

*@Test public void helloWorld() throws Exception {*

*mockMvc.perform(get("/hello").accept(MediaType.TEXT\_PLAIN))*

*.andExpect(status().isOk())*

*.andExpect(content().contentType(MediaType.TEXT\_PLAIN))*

*.andExpect(content().string("Hello World!"));*

*}*

1. **What is MvcResult?**

* Provides access to the result of an executed request.

**SPRINGBOOT**

1. **What is the key benefit of using Spring boot?**

* The key benefit is that you can “build a production ready application from scratch in a matter of minutes”.

1. **How does Spring boot enable you to build a production ready application from scratch so fast?**

* It takes the approach of “convention over configuration”
* The Spring jars dependency management and versioning are simplified
* Spring Boot’s main benefit is its ability to configure resources based on what it finds in your classpath. If your Maven POM includes JPA dependencies and a PostgreSQL driver, then Spring Boot will setup a persistence unit based on PostgreSQL. If you’ve added a web dependency, then you get Spring MVC configured with sensible defaults.
* Spring boot is based on an HTTP server. Spring Boot has an embedded version of Tomcat by default, but gives you a way to opt for Jetty server if you wish.

1. **How do you specify dependencies in Spring boot?**

* Via spring-boot-starter-<name of the dependencies>

1. **How will you get Spring boot to use Jetty server instead of Tomcat, which is the default?**

* By adding the jetty server dependency “**spring-boot-starter-jetty**” in the pom.xml file.

1. **What is Spring boot CLI?**

* CLI stands for **C**ommand **L**ine **I**nterface, which is a Spring Boot software to run and test Spring Boot applications from command prompt. When you run Spring Boot applications using CLI, then it internally uses **Spring Boot Starter** and S**pring Boot AutoConfigurate**components to resolve all dependencies and execute the application. It internally contains Groovy and Grape (JAR Dependency Manager) to add Spring Boot Defaults and resolve all dependencies automatically.

1. **What are the 4 key components of Spring Boot framework?**

* **Spring Boot Starter**: is responsible for combining a group of common or related dependencies. E.g. spring-boot-starter-actuator, spring-boot-starter-web, spring-boot-starter-data-rest, spring-boot-starter-hateoas, spring-boot-starter-jdbc, spring-boot-starter-tomcat, etc
* **Spring Boot AutoConfigurator**: is responsible for simplifying the wiring up of Spring components. One of the common criticisms of Spring IO framework is that it requires lot of XML or Java based configurations. The Spring Boot AutoConfigurator component will take the burden of wiring up the Spring components. It also reduces the number of annotations. For example, @SpringBootAnnotation = @Configuration + @ComponentScan + @EnableAutoConfiguration.
* **Spring Boot CLI**: is responsible for running & testing a Spring Boot application from a command prompt. It internally uses the components “Spring Boot Starters” and “Spring Boot AutoConfigurator”. You can also run Spring Web Applications from a command prompt.
* **Spring Boot Actuator**: is responsible for providing production-ready features to a Spring Boot application without having to actually implement these things yourself. it exposes different types of information about the running application – health, metrics, info, env etc. This is not a replacement for a production-grade monitoring solution, but is a good starting point from a development & testing perspective.

1. **What are the benefits of using Spring Boot in your next micro-service application?**

* Lesser dependency management effort.
* Lesser boiler plate code to wire up Spring components.
* Easier to integrate within Spring ecosystems like spring-jdbc, spring-web, spring-orm, spring-data, spring-security, etc.

**HIBERNATE**

1. **What is Hibernate?**

* Hibernate is an open-source and lightweight ORM tool that is used to store, manipulate and retrieve data from the database.

1. **What is ORM?**

* ORM is an acronym for Object/Relational mapping. It is a programming strategy to map object with the data stored in the database. It simplifies data creation, data manipulation and data access.

1. **What are the core interfaces of Hibernate?**

* Configuration
* SessionFactory
* Session
* Query
* Criteria
* Transaction

1. **What is SessionFactory?**

* SessionFactory provides the instance of Session. It is a factory of Session. It holds the data of second level cache that is not enabled by default.

1. **Is SessionFactory a thread-safe object?**

* Yes, SessionFactory is a thread-safe object, many threads cannot access it simultaneously

1. **What is Session?**

* It maintains a connection between hibernate application and database.
* It provides methods to store, update, delete or fetch data from the database such as persist(), update(), delete(), load(), get() etc.
* It is a factory of Query, Criteria and Transaction i.e. it provides factory methods to return these instances.

1. **Is Session a thread-safe object?**

* No, Session is not a thread-safe object, many threads can access it simultaneously. In other words, you can share it between threads.

1. **What are the 4 types of association mapping are possible in hibernate?**

* One to One
* One to Many
* Many to One
* Many to Many

1. **What is lazy loading in hibernate?**

* Lazy loading in hibernate improves the performance. It loads the child objects on demand.
* Since Hibernate 3, lazy loading is enabled by default, you don't need to do lazy="true". It means not to load the child objects when parent is loaded.

1. **What is HQL (Hibernate Query Language)?**

* Hibernate Query Language is known as an object oriented query language. It is like structured query language (SQL).
* The main advantage of HQL over SQL is:
  + You don’t need to learn SQL
  + Database independent
  + Simple to write query

1. **What is the difference between first level cache and second level cache?**

* First Level Cache is **associated with Session**. It is **enabled** by default.
* Second Level Cache is associated with **SessionFactory**. It is **not enabled** by default.