**What is a Spring Framework?**

The Spring Framework is a comprehensive and modular framework for building enterprise-level applications in Java.

It provides a configuration model programming that simplify Java development.

Spring's primary features are dependency injection, aspect-oriented programming, and transaction management.

**What are Features of the Spring Framework?**

The Spring Framework has Spring IoC container. It initializes, configures, and manages the lifecycle of beans.

The IoC container uses configuration metadata, which can be provided in various forms such as XML configuration files, annotations, or Java configuration classes.

Inversion of Control (IoC) is a design principle that transfers the control of object creation and management from the application code to a container or framework.

Spring's implementation of IoC through Dependency Injection promotes loose coupling, enhances testability, and simplifies the management of dependencies.

Spring Framework allows dependency injection. Creates dependent objects outside of a class and provides those objects to a class in various ways (constructor injection, setter injection, etc.). This promotes loose coupling between the classes.

Spring Framework allows AOP (Aspect-Oriented Programming) that allows the separation of cross-cutting concerns (like logging, transaction management, security, etc.) from the business logic. This helps in reducing code duplication and increasing modularity.

Spring provides a consistent programming model for transaction management, which can be used in a declarative manner. This can be applied across various transaction management APIs such as JTA, JDBC, Hibernate, and JPA.

Spring provides Spring MVC that is a web framework built on the core concepts of Spring. It follows the Model-View-Controller design pattern and helps in building robust, scalable web applications.

Spring Boot is an extension of the Spring Framework that simplifies the setup and development of new Spring applications. It provides pre-configured templates and reduces the amount of boilerplate code, making it easier to get started with Spring development.

**4. What are the different 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 dependent objects. This is called Inversion of Control.
* Aspect oriented Programming (AOP): Spring supports cohesive development by separating application business logic from system services.
* Container: Spring Framework creates, manages and configures application objects.
* MVC Framework: Spring Framework’s MVC web application framework is highly configurable.
* Transaction Management:  Spring’s transaction support used in container less environments.
* JDBC Exception Handling: Spring offers an exception hierarchy, which simplifies the error handling.

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

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

1. Spring Core
2. Spring Bean
3. SpEL (Spring Expression Language)
4. Spring Context

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

1. JDBC (Java DataBase Connectivity)
2. ORM (Object Relational Mapping)
3. OXM (Object XML Mappers)
4. JMS (Java Messaging Service)
5. Transaction

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

1. Web
2. Web – MVC
3. Web – Socket
4. 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.

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.

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

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?

1. Use as a Full-fledged Spring web application.
2. Use as a third-party web framework, using Spring Frameworks middle-tier.
3. Use for remote usage.
4. Use as Enterprise Java Bean which can wrap existing POJOs (Plain Old Java Objects).

9. What is Spring IOC Container?

The container creates the object, wires them together, configures them and manages their complete life cycle. 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, we do not have to create objects but have to describe how they should be created. we don’t connect 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. How many ways can Dependency Injection be done?

Dependency injection can be done in three ways :

* 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 | 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?

1. BeanFactory: BeanFactory is like a factory class that contains a collection of beans. It instantiates the bean whenever asked for by clients.
2. 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 | 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.

* 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 unit test cases.
* It promotes loose coupling with minimal effort.
* It supports eager instantiation and lazy loading of the services.

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

XML-Based configuration:  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. They generally start with a bean tag. For example:

|  |
| --- |
| <bean id="studentbean" class="org.edureka.firstSpring.StudentBean">   <property name="name" value="Edureka"></property>  </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, we need to enable it in your Spring configuration file before using it. For example:

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

Java-based configuration: Java-configuration support @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:

|  |
| --- |
| @Configuration  public class StudentConfig  {  @Bean  public StudentBean myStudent()  { return new StudentBean();  }  } |

17. 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 ApplicationContext.

18. 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.

19. 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

|  |
| --- |
| public class Student  {  private Person person;  //Setters and Getters  }  public class Person  {  private String name;  private String address;  //Setters and Getters  } |

studentbean.xml

|  |
| --- |
| <bean id=&ldquo;StudentBean" class="com.edureka.Student">  <property name="person">  <!--This is inner bean -->  <bean class="com.edureka.Person">  <property name="name" value=&ldquo;Scott"></property>  <property name="address" value=&ldquo;Bangalore"></property>  </bean>  </property>  </bean> |

20. Define Bean Wiring.

When beans are combined together within the Spring container, it’s called wiring or bean wiring. The Spring container needs to know what beans are needed and how the container should use dependency injection to tie the beans together, while wiring beans.



21. What is auto wiring and name the different modes of it?

The Spring container is able to autowire relationships between the collaborating beans.   
Different modes of bean auto-wiring are:

1. no: This is default setting which means no autowiring. Explicit bean reference should be used for wiring.
2. byName: It injects the object dependency according to name of the bean. It matches and wires its properties with the beans defined by the same names in the XML file.
3. byType: It injects the object dependency according to type. It matches and wires a property if its type matches with exactly one of the beans name in XML file.
4. constructor: It injects the dependency by calling the constructor of the class. It has a large number of parameters.
5. autodetect: First the container tries to wire using autowire by *constructor*, if it can’t then it tries to autowire by *byType*.

22. What are the limitations with auto wiring?

* Overriding possibility: You can always specify dependencies using <constructor-arg> and <property> settings which will override autowiring.
* Primitive data type: Simple properties such as primitives, Strings and Classes can’t be autowired.
* Confusing nature: Always prefer using explicit wiring because autowiring is less precise.

23. What do you mean by  Annotation-based container configuration?

Instead of using XML to describe a bean wiring, the developer moves the configuration into the component class itself by using annotations on the relevant class, method, or field declaration. It acts as an alternative to XML setups. For example:

|  |
| --- |
| @Configuration  public class AnnotationConfig  {  @Bean  public MyDemo myDemo()   { return new MyDemoImpll(); }  } |

24. How annotation wiring can be turned on in Spring?

By default, Annotation wiring is not turned on in the Spring container. Thus, to use annotation based wiring we must enable it in our Spring configuration file by configuring <context:annotation-config/> element.

For example:

|  |
| --- |
| <beans xmlns="<http://www.springframework.org/schema/beans>"  xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"  xmlns:context="<http://www.springframework.org/schema/context>">  <context:annotation-config/>  <beans &hellip;&hellip;&hellip;&hellip; />  </beans> |

25. What’s the difference between @Component, @Controller, @Repository & @Service annotations in Spring?

@Component: This marks a java class as a bean. It is a generic stereotype for any Spring-managed component. The component-scanning mechanism of spring now can pick it up and pull it into the application context.

@Controller: This marks a class as a Spring Web MVC controller. Beans marked with it are automatically imported into the Dependency Injection container.

@Service: This annotation is a specialization of the component annotation. It doesn’t provide any additional behavior over the @Component annotation. You can use @Service over @Component in service-layer classes as it specifies intent in a better way.

@Repository: This annotation is a specialization of the @Component annotation with similar use and functionality. It provides additional benefits specifically for DAOs. It imports the DAOs into the DI container and makes the unchecked exceptions eligible for translation into Spring DataAccessException.

26. What is @Required annotation?

@Required is applied to bean property setter methods. This annotation simply indicates that the affected bean property must be populated at the configuration time with the help of an explicit property value in a bean definition or with autowiring. If the affected bean property has not been populated, the container will throw BeanInitializationException.

For example:

|  |
| --- |
| public class Employee  {  private String name;  @Required  public void setName(String name)  {this.name=name; }  public string getName()  { return name; }  } |

27. What is @Qualifier annotation?

When you create more than one bean of the same type and want to wire only one of them with a property  you can use the @Qualifier annotation along with @Autowired to remove the ambiguity by specifying which exact bean should be wired.

For example, here we have two classes, Employee and EmpAccount respectively. In EmpAccount, using @Qualifier its specified that bean with id emp1 must be wired.

Employee.java

|  |
| --- |
| public class Employee  {  private String name;  @Autowired  public void setName(String name)  { this.name=name; }  public string getName()  { return name; }  } |

EmpAccount.java

|  |
| --- |
| public class EmpAccount  {  private Employee emp;  @Autowired  @Qualifier(emp1)  public void showName()  {  System.out.println(&ldquo;Employee name : &rdquo;+emp.getName);  }  } |

28.  What is @RequestMapping annotation?

@RequestMapping annotation is used for mapping a particular HTTP request method to a specific class/ method in controller that will be handling the respective request. This annotation can be applied at both levels:

* Class level : Maps the URL of the request
* Method level: Maps the URL as well as HTTP request method

29. Describe Spring DAO support?

The Data Access Object (DAO) support in Spring makes it easy to work with data access technologies like JDBC, Hibernate or JDO. It allows you to code without worrying about catching exceptions that are specific to each of these technology.

30. Name the exceptions thrown by the Spring DAO classes.

DataAccessResourceFailureException, InvalidDataAccessApiUsageException, DataIntegrityViolationException, UncategorizedDataAccessException, InvalidDataAccessResourceUsageException, DataRetrievalFailureException.

31.  Which classes are present in spring JDBC API?

Classes present in JDBC API are as follows:

1. JdbcTemplate
2. SimpleJdbcTemplate
3. NamedParameterJdbcTemplate
4. SimpleJdbcInsert
5. SimpleJdbcCall

32. What are the ways by which Hibernate can be accessed using Spring?

There are two ways by which we can access Hibernate using Spring:

1. Inversion of Control with a Hibernate Template and Callback
2. Extending HibernateDAOSupport and Applying an AOP Interceptor node

33. Name the types of transaction management that Spring supports.

Two types of transaction management are supported by Spring. They are:

1. Programmatic transaction management: In this, the transaction is managed with the help of programming. It provides you extreme flexibility, but it is very difficult to maintain.
2. Declarative transaction management: In this, the transaction management is separated from the business code. Only annotations or XML based configurations are used to manage the transactions.

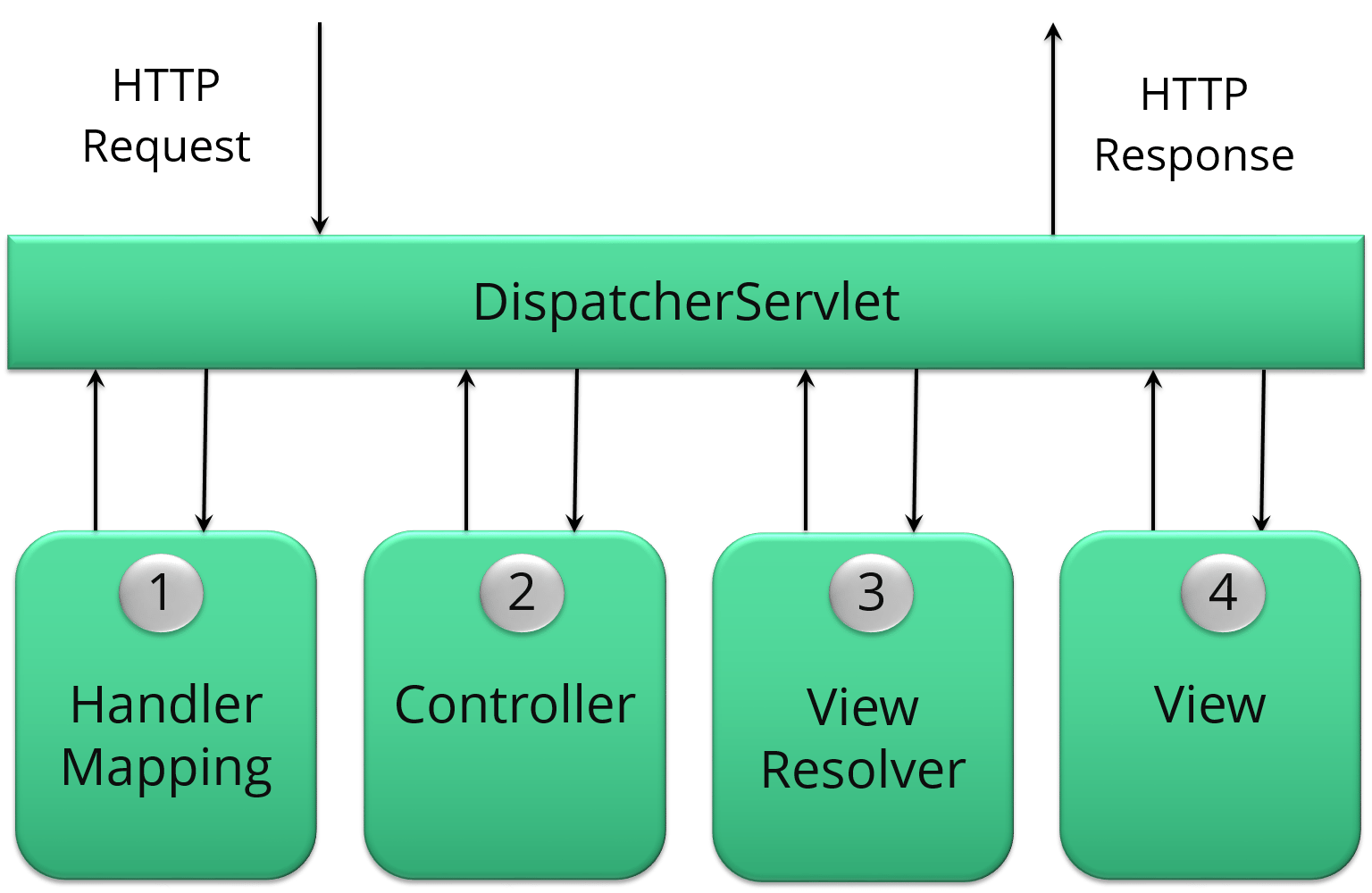
34. What are the different ORM’s supported by Spring?

JPA(Java Persistence API), JDO(Java Data Objects), Hibernate, iBatis, OJB etc

35. What do you mean by Spring MVC framework?

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

36. Describe DispatcherServlet.

The DispatcherServlet is the core of Spring Web MVC framework. It handles all the HTTP requests and responses. The DispatcherServlet receives the entry of handler mapping from the configuration file and forwards the request to the controller. The controller then returns an object of Model And View. The DispatcherServlet checks the entry of view resolver in the configuration file and calls the specified view component.

### **37. Difference between Spring and Spring Boot ?**

|  |  |
| --- | --- |
| Spring | Spring Boot |
| Spring is an open-source lightweight framework widely used to develop enterprise applications. | Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs. |
| Spring Framework is dependency injection. | Spring Boot is Autoconfiguration. |
| It helps to create a loosely coupled application. | It helps to create a tightly coupled application. |
| To run the Spring application, we need to set the server explicitly. | Spring Boot provides embedded servers such as Tomcat and Jetty etc. |
| To run the Spring application, a deployment descriptor is required. | There is no requirement for a deployment descriptor. |
| To create a Spring application, the developers write lots of code. | It reduces the lines of code. |
| It doesn’t provide support for the in-memory database. | It provides support for the in-memory database such as H2. |
| Developers need to write boilerplate code for smaller tasks. | In Spring Boot, there is reduction in boilerplate code. |

### **38. Difference between Spring MVC and Spring Boot?**

|  |  |
| --- | --- |
| SPRING MVC | SPRING BOOT |
| Spring MVC is a Model View, and Controller based web framework widely used to develop web applications. | Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs. |
| If we are using Spring MVC, we need to build the configuration manually. | If we are using Spring Boot, there is no need to build the configuration manually. |
| In the Spring MVC, a deployment descriptor is required. | In the Spring Boot, there is no need for a deployment descriptor. |
| Spring MVC specifies each dependency separately. | It wraps the dependencies together in a single unit. |
| Spring MVC framework consists of four components : Model, View, Controller, and Front Controller. | There are four main layers in Spring Boot: Presentation Layer, Data Access Layer, Service Layer, and Integration Layer. |
| It takes more time in development. | It reduces development time and increases productivity. |
| Spring MVC do not provide powerful batch processing. | Powerful batch processing is provided by Spring Boot. |

### 39. Explain Bean life cycle in spring?

### Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed. For this we write code in init() and destroy() methods. So life cycle flow is

### Container started -> Bean Instantiated -> Dependencies Injected -> Custom init() Method -> Custom utility method ->Custom destroy() method.

# 40. Difference Between ApplicationContext and WebApplicationContext in Spring MVC

| ApplicationContext | WebApplicationContext |
| --- | --- |
| ApplicationContext is used to create standalone applications. | WebApplicationContext is used to create web applications. |
| ApplicationContext is the parent of the WebApplicationContext interface. | WebApplicationContext is the child of the ApplicationContext interface. |
| In the case of ApplicationContext, we have to create and destroy the container objects manually. | But in the case of WebApplicationContext, we don’t need to create and destroy the container object. The container object will be created automatically. |
| There is always a single ApplicationContext in an application. | There can be multiple WebApplicationContexts for each of the dispatcher servlets. |
| ApplicationContext represents the Spring IoC Containers and it is the sub-interface of BeanFactory. | WebApplicationContext in Spring is a web-aware ApplicationContext i.e it has Servlet Context information. |
| ApplicationContext is used to inject all the middle-tier beans (Services, DAOs) which are instantiated using the “ContextLoaderListener” class configured in web.xml. | WebApplicationContext is used to deal with the web-related components such as controllers and view resolvers, which is configured using “DispatcherServlet”. |

41. What are the differences between Spring JDBC & Spring Data JDBC?

| Spring JDBC | Spring Data JDBC |
| --- | --- |
| Spring JDBC is a Model class. | Spring Data JDBC is a POJO class. |
| Getter and setters are mandatory. | Getter and setters are not mandatory. |
| The parameterized constructor will be helpful. | The parameterized constructor may not be helpful. |
| There is no specific annotation is required. The only thing is we should have equal attributes match with the DB table and each attribute should have a getter and setter. | @Table, @ID, and @Column annotations are helpful to mention for direct connection with the database. |
| Data Access Layer is specified with the interface and its implementation. | Data Access Layer is simple and it omits lazy loading, cache implementation, etc., which is there in JPA(Java Persistence API). |

42. What are the different methods to create spring Bean?

1. Creating Bean Inside an XML Configuration File (beans.xml)
2. Using @Component Annotation
3. Using @Bean Annotation

### 43. Differences between Bean factory and Application Context?

| BeanFactory | ApplicationContext |
| --- | --- |
| It is a fundamental container that provides the basic functionality for managing beans. | It is an advanced container that extends the BeanFactory that provides all basic functionality and adds some advanced features. |
| It is suitable to build standalone applications. | It is suitable to build Web applications, integration with AOP modules, ORM and distributed applications. |
| It supports only Singleton and Prototype bean scopes. | It supports all types of bean scopes such as Singleton, Prototype, Request, Session etc. |
| It does not support Annotations. In Bean Autowiring, we need to configure the properties in XML file only. | It supports Annotation based configuration in Bean Autowiring. |
| This interface does not provides messaging (i18n or internationalization) functionality. | ApplicationContext interface extends MessageSource interface, thus it provides messaging (i18n or internationalization) functionality. |
| BeanFactory does not support Event publication functionality. | Event handling in the ApplicationContext is provided through the ApplicationEvent class and ApplicationListener interface. |
| In BeanFactory, we need to manually register BeanPostProcessors and BeanFactoryPostProcessors. | The ApplicationContext automatically registers BeanFactoryPostProcessor and BeanPostProcessor at startup. |
| BeanFactory will create a bean object when the getBean() method is called thus making it Lazy initialization. | ApplicationContext loads all the beans and creates objects at the time of startup only thus making it Eager initialization. |
| BeanFactory interface provides basic features only thus requires less memory. For standalone applications where the basic features are enough and when memory consumption is critical, we can use BeanFactory. | ApplicationContext provides all the basic features and advanced features, including several that are geared towards enterprise applications thus requires more memory. |

44. Explain WebApplicationContext.

The WebApplicationContext is an extension of the plain ApplicationContext. It has some extra features that are necessary for web applications. It differs from a normal ApplicationContext in terms of its capability of resolving themes and in deciding which servlet it is associated with.

45. 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.

# 46. What are Bean Scopes in Java Spring?

**Bean Scopes** refers to the lifecycle of Bean that means when the object of Bean will be instantiated, how long does that object live, and how many objects will be created for that bean throughout. it is managed by the spring container.

The [spring framework](https://www.geeksforgeeks.org/introduction-to-spring-framework/) provides five scopes for a bean. We can use three of them only in the context of web-aware **Spring ApplicationContext** and the rest of the two is available for both **IoC container and Spring-MVC container.**

1. Singleton: Only one instance will be created for a single bean definition per Spring IoC container and the same object will be shared for each request made for that bean.
2. Prototype: A new instance will be created for a single bean definition every time a request is made for that bean.
3. Request: A new instance will be created for a single bean definition every time an HTTP request is made for that bean. But Only valid in the context of a web-aware Spring ApplicationContext.
4. Session: Scopes a single bean definition to the lifecycle of an HTTP Session. But Only valid in the context of a web-aware Spring ApplicationContext.
5. Global-Session: Scopes a single bean definition to the lifecycle of a global HTTP Session. It is also only valid in the context of a web-aware Spring ApplicationContext.

# 47. What are spring boot layers?

1. Presentation Layer – Authentication & Json Translation
2. Business Layer – Business Logic, Validation & Authorization
3. Persistence Layer – Storage Logic
4. Database Layer – Actual Database

The presentation layer is the top layer of the spring boot architecture. It consists of Views. i.e., the front-end part of the application. It handles the HTTP requests and performs authentication. It is responsible for converting the JSON field’s parameter to Java Objects and vice-versa. Once it performs the authentication of the request it passes it to the next layer. i.e., the business layer.

The business layer contains all the business logic. It consists of services classes. It is responsible for validation and authorization.

The persistence layer contains all the database storage logic. It is responsible for converting business objects to the database row and vice-versa.

The database layer contains all the databases such as [MySql](https://www.geeksforgeeks.org/sql-tutorial/), [MongoDB](https://www.geeksforgeeks.org/mongodb-an-introduction/), etc. This layer responsible for performing the [CRUD](https://www.geeksforgeeks.org/spring-boot-crud-operations/) operations.

### 48. How can you fetch records by spring JdbcTemplate?

You can fetch records from the database by the query method of JdbcTemplate. There are two interfaces to do this:

1. [ResultSetExtractor](https://www.javatpoint.com/ResultSetExtractor-example)
2. [RowMapper](https://www.javatpoint.com/RowMapper-example)

### 49. What is the advantage of NamedParameterJdbcTemplate?

NamedParameterJdbcTemplate class is used to pass value to the named parameter. A named parameter is better than ? (question mark of PreparedStatement).

### 50.What does the ViewResolver class?

The View Resolver class resolves the view component to be invoked for the request. It defines prefix and suffix properties to resolve the view component.

### 51. How to exclude any package without using the basePackages filter?

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

@SpringBootApplication(exclude= {Student.class})

### 52. How to disable specific auto-configuration class?

You can use the exclude attribute of @EnableAutoConfiguration

@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 @EnableAutoConfiguration(excludeName={Foo.class})

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

### 53. 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

54. What is servlet life cycle?

The entire life cycle of a Servlet is managed by the **Servlet container** which uses the **javax.servlet.Servlet** interface to understand the Servlet object and manage it.

The Servlet life cycle mainly goes through four stages.

* Loading a Servlet.
* Initializing the Servlet.
* Request handling.
* Destroying the Servlet.

The Web container or Servlet Container can load the Servlet by Initializing the context and instantiated.

After that The container initializes the Servlet object by invoking the **Servlet.init(ServletConfig)** method which accepts ServletConfig object reference as parameter.

After initialization, the Servlet instance is ready to serve the client requests. It creates the **ServletRequest** and **ServletResponse** objects.

After creating the request and response objects it invokes the Servlet.service method by passing the request and response objects.

When a Servlet container decides to destroy the Servlet then calls the **destroy()** method on the Servlet instance.

There are three life cycle methods of a Servlet init(), service(), destroy()

55. What is the life cycle of JSP?

* Translation of JSP page to Servlet
* Compilation of JSP page(Compilation of JSP into test.java)
* Classloading (test.java to test.class)
* Instantiation(Object of the generated Servlet is created)
* Initialization(jspInit() method is invoked by the container)
* Request processing(\_jspService()is invoked by the container)
* JSP Cleanup (jspDestroy() method is invoked by the container)

# 56. Difference Between Spring DAO vs Spring ORM vs Spring JDBC

| Spring DAO | Spring ORM | Spring JDBC |
| --- | --- | --- |
| @Repository annotation are mandatory. | Easy integration with Spring with the following   * SessionFactory for Hibernate * EntityManagerFactory for JPA * SqlSessionFactory for MyBatis | For plain JDBC calls. |
| Data access implementation is totally separated and hence it is independent of the database. | Multi-technology implementation is possible by integrating with the required tools. | If the application is not complex and lies on a single database |
| An additional layer and its dependencies need to be specified. Hence it may take some time to start if the application is complex. | An additional layer and its dependencies need to be specified. Hence it may take some time to start if the application is complex. | As this is straightforward, no complex dependencies are required but portability will become less if we use this. |
| Maintenance issues will be there because of the complexity of the additional layer. | Maintenance issues will be there because of the complexity of the additional layer. | Here less maintenance only. |
| Design patterns like Factory classes, and Data Transfer Object(DTO) are needed to get implemented along with DAO. | Got the support for multiple technologies like Hibernate, JPA, and iBatis. | Implementation is simple. If relying on a single database and direct query purpose means can depend on this. |

57.What is JDBC API?

Java Database Connectivity (JDBC) is an application programming interface **(API)** is a data access technology used for Java database connectivity. It provides methods to query and update data in a database. There are 4 Types of JDBC Drivers:

1. JDBC-ODBC Bridge Driver
2. Native API Driver (partially java driver)
3. Network Protocol Driver (fully java driver)
4. Thin Driver (fully java driver)

58.What are the advantages and disadvantages of JDBC API?

Advantages:

1. Automatically creates the XML format of data from the database.
2. It supports query and stored procedures.
3. Almost any database for which ODBC driver is installed can be accessed.

Disadvantages:

1. Writing a lot of codes before and after executing the query, such as creating connection, creating a statement, closing result-set, closing connection, etc.
2. Writing exception handling code on the database logic.
3. Repetition of these codes from one to another database logic is time-consuming.

These problems of **JDBC API** are eliminated by **Spring JDBC-Template**.

59.What is Spring JDBC?

Spring framework provides the following approaches for JDBC database access:

* JdbcTemplate
* NamedParameterJdbcTemplate
* SimpleJdbcTemplate
* SimpleJdbcInsert and SimpleJdbcCall

60.What is JDBC Template?

**JdbcTemplate**is simplifies the use of **JDBC.** It executes SQL queries or updates, initiating iteration over ResultSets and catching **JDBC**exceptions and translating them to the generic. It handles the exception and provides the informative exception messages with the help of exception classes defined in the **org.springframework.dao** package.

61.What are the common methods of Spring JDBC Template class?

| JDBC Template Class Methods | Description |
| --- | --- |
| public int update(String query) | Used to insert, update and delete records. |
| public int update(String query, Object… args) | Used to insert, update and delete records using PreparedStatement using given arguments. |
| public T execute(String sql, PreparedStatementCallback action) | Executes the query by using PreparedStatementCallback. |
| public void execute(String query) | Used to execute DDL query. |
| public T query(String sql, ResultSetExtractor result) | Used to fetch records using ResultSetExtractor. |

### 62.Write JDBC Template queries?

### Select query example:

int result = jdbcTemplate.queryForObject("SELECT COUNT(\*) FROM STUDENT", Integer.class);

Insert or Update query example:

return jdbcTemplate.update("INSERT INTO STUDENT VALUES (?, ?, ?)", id, "megan", "India");

### 63.How to use spring prepared statement JDBC Template?

### Spring provides **PreparedStatementSetter** interface in **org.springframework.jdbc.core** package used by the JdbcTemplate class. The JdbcTemplate will create the PreparedStatement and with the callback only being responsible for setting parameter values.

### 64.Explain Spring Data Repository Interface?

### Repository, CrudRepository and PagingAndSortingRepository belong to Spring Data Commons whereas JpaRepository belongs to Spring Data JPA.

65.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

# 66.What are the differences between Spring Boot Starter Web and Spring Boot Starter Tomcat?

| Spring Boot Starter Web | Spring Boot Starter Tomcat |
| --- | --- |
| Spring Boot Starter Web is used for building RESTful applications using Spring MVC. | Spring Boot Starter Tomcat is the default embedded container for Spring Boot Starter Web. |
| We cannot exclude it while using web services. | We can exclude it when we want to use another embedded container. |
| It also supports Jetty Server and Undertow Server. | It acts as an embedded web server. |
| It auto-configures Dispatcher Servlet, Error Page, Embedded servlet container, and Web JARs for managing the static dependencies for web development. | It has a core, el. logging, WebSocket. |
| It contains spring web dependencies. | It contains everything related to an embedded tomcat server. |
| It auto-configures the features used for web development. | It is used as the default embedded container. |

### 67.What are the differences between CrudRepository and JpaRepository?

| CrudRepository | JpaRepository |
| --- | --- |
| It is a base interface and extends Repository Interface. | It extends PagingAndSortingRepository that extends CrudRepository. |
| It contains methods for CRUD operations. For example save(), saveAll(), findById(), findAll(), etc. | It contains the full API of CrudRepository and PagingAndSortingRepository. For example, it contains flush(), saveAndFlush(), saveAllAndFlush(), deleteInBatch(), etc along with the methods that are available in CrudRepository. |
| It doesn’t provide methods for implementing pagination and sorting | It provides all the methods for which are useful for implementing pagination. |
| It works as a marker interface. | It extends both CrudRepository and PagingAndSortingRepository. |
| To perform CRUD operations, define repository extending CrudRepository. | To perform CRUD as well as batch operations, define repository extends JpaRepository. |
| Syntax:  public interface CrudRepository<T, ID> extends Repository<T, ID> | Syntax:  public interface JpaRepository<T,ID> extends PagingAndSortingRepository<T,ID>, QueryByExampleExecutor<T> |

68.What is Spring boot?

[Spring Boot](https://www.geeksforgeeks.org/introduction-to-spring-boot/) is built on the top of the spring and contains all the features of spring.

It avoids heavy configuration of XML.

It provides easy maintenance.

It includes embedded Tomcat-server.

Deployment is very easy, war and jar files can be easily deployed in the tomcat server

### 69.Explain Spring Boot Flow Architecture?

* The Client makes an **HTTP** request(GET, PUT, POST, etc.)
* The HTTP request is forwarded to the **Controller**. The controller maps the request. It processes the handles and calls the server logic.
* The business logic is performed in the **Service layer**. The spring boot performs all the logic over the data of the database which is mapped to the spring boot model class through [Java Persistence Library](https://www.geeksforgeeks.org/spring-boot-spring-data-jpa/)(**JPA**).
* The [JSP](https://www.geeksforgeeks.org/introduction-to-jsp/) page is returned as Response from the controller.

70.What are the types of Spring Framework Annotations?

Basically, there are 6 types of annotation available in the whole spring framework.

1. Spring Core Annotations
2. Spring Web Annotations
3. Spring Boot Annotations
4. Spring Bean Annotations
5. Spring Data Annotations
6. Spring Scheduling Annotations

71.What are spring core annotations?

We can divide them into two categories.

Dependency Injection related annotations:

@Autowired, @Qualifier, @Primary, @Bean, @Lazy, @Required, @Value

@Scope, @Lookup.

Context Configuration Annotations:

@Profile, @Import, @ImportResource, @PropertySource.

These core annotations present in the ***org.springframework.beans.factory.annotation*** and

***org.springframework.context.annotation*** packages.

72.What is @Autowired and @Qualifier?

@Autowired:

It injects object dependency implicitly by the Spring container. It injects **fields, setter methods, and constructors.**

@Qualifier:

It is used to resolve the autowiring conflict when there are multiple beans of the same type. It is used on any class annotated with @Component or on methods annotated with @Bean:

For example *if you want to inject the Bike bean in VehicleService then you must use @Autowired with @Qualifier annotation. If you didn’t use @Qualifier, it will throw* ***NoUniqueBeanDefinitionException****.*

73.What is @Bean annotation?

Spring [@Bean](https://www.digitalocean.com/community/users/bean) Annotation is applied on a method to specify that it returns a bean to be managed by Spring context.

74.What are Spring Web Annotations?

These annotations present in the ***org.springframework.web.bind.annotation p***ackage

* @RequestMapping
* @RequestBody
* @PathVariable
* @RequestParam
* Response Handling Annotations
  + @ResponseBody
  + @ExceptionHandler
  + @ResponseStatus
* @Controller
* @RestController
* @ModelAttribute
* @CrossOrigin

75.What is difference between @Controller and @RestController?

@Controller is used to mark classes as Spring MVC Controller.

@RestController annotation is a special controller used in RESTful Web services, and it’s the combination of @Controller and @ResponseBody annotation.

In @Controller, we can return a view in Spring Web MVC.

In @RestController, we can not return a view.

In @Controller, we need to use @ResponseBody on every handler method.

In @RestController, we don’t need to use @ResponseBody on every handler method.

76.What is difference between @RequestParam and @PathVariable?

The @RequestParam annotation is used to bind a request parameter to a method parameter that extract query parameters or form data from an HTTP request.

The @PathVariable annotation is used to bind a URI variable to a method parameter. It is typically used to extract values from the path portion of a URI.

77.What is @RequestMapping?

The**@RequestMapping** **Annotation**

**It** is used to map HTTP requests to handler methods of MVC and REST controllers.

The @RequestMapping annotation can be applied to class-level and/or method-level in a controller.

The class-level annotation maps a specific request path or pattern onto a controller.

The method-level annotations to make mappings more specific to handler methods.

78.What is @RequestBody?

 The @RequestBody is method parameter level annotation to indicate that method parameter will bind to web request body.

we can use @RequestBody parameter in the methods annotated with @PostMapping and @PutMapping

79.What is @ResponseBody?

@ResponseBody is binds a method return value to the web response body.

It uses HTTP Message converters to convert the return value to HTTP response body, based on the content-type in the request HTTP header.

80.What is CORS?

Cross-Origin Resource Sharing (CORS) is a configuration While working with cross domains, most of the time we tend to worry about what & where it went wrong. There are many factors including security, web components, sockets, etc to be handled at the server side before a request is processed.

81.What does @CrossOrigin in spring boot?

By adding @CrossOrigin annotation to Main class or to specific controllers and methods that API should be accessed only specific domains.

Example:

@CrossOrigin("\*") // to allow from all domains

@CrossOrigin("http://localhost:3001") // to allow from specific domain

@CrossOrigin(origins = "http://localhost:3001")

If spring application is MVC then add CORS mappings by overriding addCorsMappings in WebMvcConfigurer interface.

Example:

@Configuration

@EnableWebMvc

public class WebConfig implements WebMvcConfigurer {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*").allowedOrigins("\*").allowedHeaders("\*");

}

}

When security is enabled in the application then CORS must be implementated in the SecurityConfig. By adding UrlBasedCorsConfigurationSource and by extending CorsFilter we can configure.

82.What are HTTP annotations?

These annotations maps the HTTP requests on the specific handler method.

@GetMapping: It maps the HTTP GET requests on the specific handler method. It is used to create a web service endpoint that fetches It is used instead of using: @RequestMapping(method = RequestMethod.GET)

@PostMapping: It maps the HTTP POST requests on the specific handler method. It is used to create a web service endpoint that creates It is used instead of using: @RequestMapping(method = RequestMethod.POST)

@PutMapping: It maps the HTTP PUT requests on the specific handler method. It is used to create a web service endpoint that creates or updates.

@DeleteMapping: It maps the HTTP DELETE requests on the specific handler method. It is used to create a web service endpoint that deletes a resource.

@PatchMapping: It maps the HTTP PATCH requests on the specific handler method.

83.What are Spring Boot Annotations?

Spring annotations present in the ***org.springframework.boot.autoconfigure*** and ***org.springframework.boot.autoconfigure.condition***packages are commonly known as Spring Boot annotations.

* @SpringBootApplication
* @EnableAutoConfiguration
* Auto-Configuration Conditions
  + @ConditionalOnClass, and @ConditionalOnMissingClass
  + @ConditionalOnBean, and @ConditionalOnMissingBean
  + @ConditionalOnProperty
  + @ConditionalOnResource
  + @ConditionalOnWebApplication and @ConditionalOnNotWebApplication
  + @ConditionalExpression
  + @Conditional

84.What does @SpringBootApplication annotation?

It is class level annotation used for main class of a Spring Boot application.

It encapsulates **@SpringBootConfiguration**, **@EnableAutoConfiguration**, and **@ComponentScan** annotations with their default attributes.

If an application uses @SpringBootApplication, it is already using @SpringBootConfiguration, **@EnableAutoConfiguration** and **@ComponentScan.**

85.What does **@EnableAutoConfiguration Annotation?**

**It** auto-configures the beans that are present in the classpath and configure it to run the application.

For example, when we use **spring-boot-starter-web** dependency in the classpath, Spring boot auto-configures [Tomcat](https://www.geeksforgeeks.org/embedding-tomcat-server-in-maven-project/), and [Spring MVC](https://www.geeksforgeeks.org/difference-between-spring-mvc-and-spring-boot/).

we need to apply the @EnableAutoConfiguration annotation in the root package so that every sub-packages and class can be examined.

86.What are Spring bean annotations?

There’re several ways to configure beans in a Spring container. You can declare them using XML configuration or you can declare beans using the @Bean annotation in a configuration class or you can mark the class with one of the annotations from the ***org.springframework.stereotype*** package and leave the rest to component scanning. Some of the annotations that are available in this category are:

* @ComponentScan
* @Configuration
* [Stereotype Annotations](https://www.geeksforgeeks.org/spring-stereotype-annotations/)
  + [@Component](https://www.geeksforgeeks.org/spring-component-annotation-with-example/)
  + @Service
  + [@Repository](https://www.geeksforgeeks.org/spring-repository-annotation-with-example/)
  + [@Controller](https://www.geeksforgeeks.org/spring-controller-annotation-with-example/)

87.What does **@ComponentScan Annotation?**

**@ComponentScan Annotation**

It enables Spring to scan for things like configurations, controllers, services, and other components that are defined.

Generally, @ComponentScan annotation is used with @Configuration annotation to specify the package for Spring to scan for components.

88.What does @Configuration?

**@Configuration annotation which indicates that the class has @Bean definition methods**.

So Spring container can process the class and generate Spring Beans to be used in the application.

89.What does @SpringBootConfiguration annotation?

@SpringBootConfiguration Annotation

It is a class-level annotation provides Spring Boot application configuration.

It can be used as an alternative to Spring’s standard **@Configuration** annotation.

so that configuration can be found automatically.

90.What are Stereotype Annotations?

These are used to create Spring beans automatically in the application context.

@Component annotation is the main Stereotype Annotation.

@Service, @Repository, @Controller are derived from @Component.

@Component is a class-level annotation across the application to mark the beans as Spring’s managed components.

A class with @Service to indicate that they’re holding the business logic.

A class with @Repository to indicate that they’re dealing with **CRUD operations** that deal with database tables.

A class with @Controller to indicate that they’re handling user requests and return the appropriate response.

91.What are Spring data annotations?

* Common Spring Data Annotations
  + @Transactional
  + @NoRepositoryBean
  + @Param
  + @Id
  + @Transient
  + @CreatedBy, @LastModifiedBy, @CreatedDate, @LastModifiedDate
* Spring Data JPA Annotations
  + @Query
  + @Procedure
  + @Lock
  + @Modifying
  + @EnableJpaRepositories
* Spring Data Mongo Annotations
  + @Document
  + @Field
  + @Query
  + @EnableMongoRepositories

92.What is ACID properties in DBMS?

**Atomicity** Consistency Isolation Durability

Transactions access data using read and write operations.   
In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

93.What is Atomicity in DBMS?

Entire transaction happens at once or doesn’t happen at all.

It involves Abort & Commit operations.

If a transaction is aborts then changes made to the database are not visible.

If a transaction is commits then changes made are visible.

94.Could you tell me an example for Atomicity in DBMS?

Consider a transaction that transfer of 100 from account X to account Y.

Suppose if transaction is fails after deduction of 100 from account X then the amount has been deducted from **X** but not added to **Y**.

Therefore, the transaction must be executed at once in order to ensure the correctness of the database state.

95.What is Consistency in DBMS?

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction.

For Example Consider a transaction that transfer of 100 from account X to account Y. The total amount before and after the transaction must be maintained. Inconsistency occurs if transaction fails.

96.What is isolation in DBMS?

This property ensures that the execution of multiple transactions concurrently and changes should be visible only after they have been made to the main memory.

97.What is durability in DBMS?

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs.

98.What does @Transactional annotation?

We can annotate a bean with @Transactional at the class or method level. It supports propagation type of transaction, Isolation level of transaction and also used for rollback.

rollbackFor, noRollbackFor, propagation, isolation attributes are used.

99.What does @Transient?

@Transient annotation is used to mark a field to be transient for the mapping.

This annotation is used to specify that a given entity attribute should not be persisted to the table.

100.What is the use of @Query?

The @Query annotation defines queries directly on repository methods.

JPQL is used to make queries against entities stored in a relational database.

We can write native query by setting attribute nativeQuery is True.

We can bind parameters to query according to name or index.

Named parameters are easier to read and less error-prone.

Example:

// single named parameter

@Query("SELECT n FROM Note n WHERE n.title = :title")

List<Note> findByTitleNamedBind(@Param("title") String title);

// multiple named parameters

@Query("SELECT n FROM Note n WHERE n.title = :title AND n.featured = :featured")

List<Note> findByTitleAndFeaturedNamedBind(@Param("featured") boolean featured, @Param("title") String title);

// multiple bind parameters based on index

@Query("SELECT n FROM Note n WHERE n.title = ?1 AND n.featured = ?2")

List<Note> findByTitleAndFeaturedPositionalBind(String title, boolean featured);

101.What is @Param?

The @Param annotation specifies the name of the bind parameter in the method definition.

Each method parameter annotated with @Param must have a corresponding bind parameter in the JPQL or SQL query.

102.What does @ID annotation?

This annotation is placed on a specific field that holds the persistent identifying properties. This field is treated as a primary key in database.

What are Spring Scheduling Annotations?

Spring annotations present in the ***org.springframework.scheduling.annotation*** packages are commonly known as Spring Scheduling annotations.

* @EnableAsync
* @EnableScheduling
* @Async
* @Scheduled
* @Schedules

103.What is POJO?

POJO stands for “Plain Old Java Object”. It has fields with getters and setters and many override methods like equals etc.

POJOs basically define an entity.

POJOs are used for increasing the readability and re-usability of a program.

POJO should not extend a class and should not implement an interface.

POJO should not contain pre specified annotations.

All JavaBeans are POJOs but not all POJOs are JavaBeans.

104.What is difference between POJO and Bean?

| POJO | Java Bean |
| --- | --- |
| It doesn’t provide much control on members. | It provides complete control on members. |
| It can implement Serializable interface. | It should implement serializable interface. |
| Fields can be accessed by their names. | Fields are accessed only by getters and setters. |
| Fields can have any visibility. | Fields have only private visibility. |
| There may/may-not be a no-arg constructor. | It must have a no-arg constructor. |

105.What is JPA?

JPA(Java/Jakartha Persistence API) is a collection of classes and methods that have been provided by Oracle to store the massive amount of data in a database.

It forms a bridge between object models and relational models.

we can use JPA Upon Object-Relation Mapping (ORM).

It has the runtime EntityManager API and it is responsible for processing queries and transactions on the Java objects against the database.

It uses JPQL (Java Persistent Query Language).

106.What are JPA advantages?

* No need to write DDL/DML queries, instead we can map by using XML/annotations.
* JPQL is used and since it is platform-independent, we no need to depend on any native SQL table. Complex expressions and filtering expressions are all handled via JPQL only.
* Dynamic generation of queries is possible.
* Integration with Spring framework is easier with a custom namespace.

107.What are JPA related Annotations?

@Entity, @Table, @Column, @OnetoOne, @OnetoMany, @ManytoOne, @ManytoMany

108.What is @Data?

@Data generates all the boilerplate code that is normally associated with simple POJO and beans i.e getters & setters, toString, equal, hashcode etc.

109.What is @Entity?

The @Entity annotation specifies that the class is an entity and is mapped to a database table.

110.What are the differences between JPA & Hibernate?

|  |  |
| --- | --- |
| JPA | HIBERNATE |
| JPA : It is a Java specification for mapping relational data in Java application. It is not a framework | Hibernate is an ORM framework and in that way data persistence is possible. |
| In JPA, no implementation classes are provided. | In Hibernate, implementation classes are provided. |
| Main advantage is It uses JPQL (Java Persistence Query Language)  and it is platform-independent query language. | Here it is using HQL (Hibernate Query Language). |
| It is available under javax.persistence package. | It is available under org.hibernate package. |
| Persistence of data is handled by EntityManager. | Persistence of data is handled by Session. |

111.What is INNER JOIN?

It selects records that have matching values in both tables.

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns.

If there are records in the "table1" table that do not have matches in "table2", these table1 records will not be shown.

Syntax:

SELECT column\_names FROM table1 INNER JOIN table2 ON table1.column\_name = table2.column\_name;

112.What is LEFT JOIN?

It returns all records from the left table (table1), and the matching records from the right table (table2).

The result is 0 records from the right side, if there is no match.

The LEFT JOIN keyword returns all records from the left table, even if there are no matches in the right table.

In some databases LEFT JOIN is called LEFT OUTER JOIN.

Syntax:

SELECT column\_names FROM table1 LEFT JOIN table2 ON table1.column\_name = table2.column\_name;

113.What is RIGHT JOIN?

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1).

The result is 0 records from the left side, if there is no match.

The RIGHT JOIN keyword returns all records from the right table, even if there are no matches in the left table.

In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

Syntax:

SELECT column\_names FROM table1 RIGHT JOIN table2 ON table1.column\_name = table2.column\_name;

114.What is FULL JOIN or FULL OUTER JOIN?

The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN can potentially return very large result-sets!

Syntax:

SELECT column\_name FROM table1 FULL OUTER JOIN table2 ON table1.column\_name = table2.column\_name WHERE condition;

115.What is SELF JOIN?

A self join is a regular join, but the table is joined with itself by different table aliases.

Syntax:

SELECT column\_names FROM table1 T1, table1 T2 WHERE condition;

116.What is difference between JOIN and UNION?

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

The UNION operator is used to combine the result-set of two or more SELECT statements.

Every SELECT statement within UNION must have the same number of columns.

For UNION, The columns must also have similar data types.

For UNION, The columns in every SELECT statement must also be in the same order.

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL.

117.What is Hibernate?

Hibernate is an open-source, ORM(Object Relational Mapping) framework that provides CRUD operations in the form of objects.

It is a top layer of JDBC, JTA, and JNDI Technologies. It follows the ORM concept

118.What are the Associations in Hibernate?

* One-To-One
* One-To-Many
* Many-To-One
* Many-To-Many

119.What is One to One Relation?

In this type of association one instance of source entity can be mapped atmost one instance of target entity.

Relationship between a Student and Library in such a way that one student can be issued only one type of book.

Syntax: @OneToOne

private Student stud;

120.What is One to Many Relation?

In this type of association the instance of one entity can be mapped with any number of instances of another entity.

Relationship between a Student and Library in such a way that one student can be issued more than one type of book.

Syntax: @OneToMany(targetEntity=Library.class)

121.What is Many to One Relation?

The Many-To-One mapping represents a single-valued association where a collection of entities can be associated with the similar entity.

Many employees work in the same company so all the working employees have the same company address. Many employees have one address.

Syntax: @ManyToOne(cascade = CascadeType.ALL)

@JoinColumn(name = "Foreign key column")

122.What is Many to Many Relation?

In relational database any number of rows of one entity can be referred to any number of rows of another entity.

The relationship between a Student and Library in such a way that any number of students can be issued any type of books.

Syntax: @ManyToMany(targetEntity=Library.class)

     private List lib;

123.What are types of mapping in Hibernate?

The hibernate works to link the JAVA language to the database table along with this link we can establish relations/mappings.

1. Primitive Types : Maps java data type to RDBMS data type.
2. Date and Time Types : Maps date and time datatype mappings.
3. Binary and Large Object Types : Maps large objects like images and videos.
4. JDK-related Types : class, locale, currency, timezone mappings.

124.What is JPA Cascading Operations?

In JPA, if any operation is applied on one entity then it will perform on that particular entity only. These operations will not be applicable to the other entities that are related to it.

To establish a dependency between related entities, JPA provides **javax.persistence.CascadeType** enumerated types that define the cascade operations.

These cascading operations can be defined with any type of mapping i.e. One-to-One, One-to-Many, Many-to-One, Many-to-Many.

|  |  |
| --- | --- |
| Cascade Operations | Description |
| PERSIST | In this cascade operation, if the parent entity is persisted then all its related entity will also be persisted. |
| MERGE | In this cascade operation, if the parent entity is merged then all its related entity will also be merged. |
| DETACH | In this cascade operation, if the parent entity is detached then all its related entity will also be detached. |
| REFRESH | In this cascade operation, if the parent entity is refreshed then all its related entity will also be refreshed. |
| REMOVE | In this cascade operation, if the parent entity is removed then all its related entity will also be removed. |
| ALL | In this case, all the above cascade operations can be applied to the entities related to parent entity. |

125.What is JPA Collection Mapping?

we embed an object in an entity class and define it as a collection type List. The annotation @ElementCollection represents the embedded object. create a class of embedded object Address.java under com.javatpoint.jpa package. The annotation @Embeddable represents the embeddable object.

126.What is Hibernate Validator Framework?

* 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
* JSR303 and JSR349 provide specifications for bean validation by using annotations.

127.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.

128.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.
* The application context is where Spring bean resides. For a web application, there is a subclass called WebAppliationContext.
* 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.

129.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.

### 130.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. 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.

### 131.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

132.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. |

### 133.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

### 134.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.

### 135.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

### 136.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.

@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.

137.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

### 138.Is there any need to keepspring-mvc.jar on the classpath or is it already present as part of spring-core?

The spring-mv.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.

### 139.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.

### 140.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.

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

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

@Autowired

private ServletContext servletContext;

@Autowired

private ServletConfig servletConfig;

142.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.
* LocaleChangeInterceptor: This bean is useful to resolve the parameter from the incoming request.
* 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.

143.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 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.