Spring Core

1. What is Spring?

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

2. What was the main objective of spring innovation?

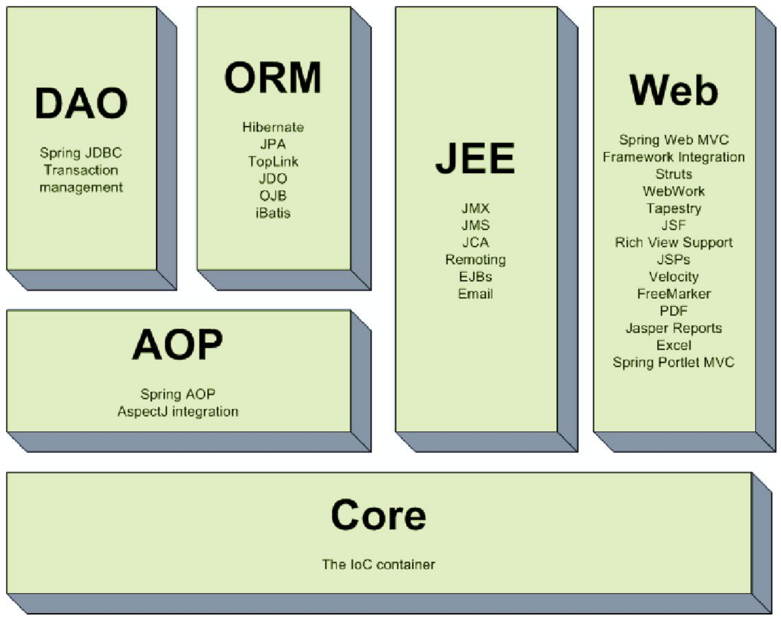
Simplification of j2EE application development, i.e. reducing the complexity involved in enterprise application development by directly using J2EE technologies.

3. What are the advantages of spring?

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

4. What is the Architecture of spring?

<u>Spring Architecture</u>



5. What are the modules of spring framework?

- Spring Core Module
- Spring Context [J2EE]
- Spring DAO Module [Spring JDBC]
- Spring ORM module
- Spring AOP [Aspect Oriented Programming]
- Spring WEB-MVC Module

1) Core module:

- The core module of spring framework is spring container.
- This module implements the principles of IOC (Inversion Of Control)
- We call IOC as heart of spring applications
- IOC is a container used for holding and manages spring beans
- It is the base for all modules, so we are using this module in all layers of the enterprise application

2) DAO module:

- To develop data access layer
- Spring has given support to integrate JDBC into spring
- The aim of this module is to eliminate complexities of direct usage of JDBC technology

3) ORM module:

- To develop data access layer
- Spring has given support to integrate most of the ORM frameworks (Hibernate, Toplink, JPA, Ibatis, JDO… etc) into spring
- The aim of this module is to eliminate complexities of direct usage of ORM frameworks

4) JEE module:

- To develop business layer
- This module provides the SUN enterprise services to the spring bean
- Spring has given support to integrate most of the J2EE technologies(EJB’s, RMI, Java-mail, JMS, JMX, Web Services, JTA, JTS, JAAS …etc) into spring
- The aim of this module is to eliminate complexities of direct usage J2EE technologies

5) Web MVC module:

- To develop controller
    - Spring has given support to integrate most of the MVC frameworks(Struts, JSF, Wicket, Tapestry …etc) into spring
    - Spring also given its own MVC implementation

- To develop presentation layer
    - Spring has given tag support to develop JSP pages
    - Spring has give a support to integrate most of the View technologies(Velocity, Wicket, PDF, Excel, JSP …etc) into spring.

6) AOP module:

- To develop proxies for business layer objects
- To solve the cross cutting concern problems, spring has given Aspect Oriented Programming(AOP)
- To provide declarative enterprise services (logging, security, transaction management...etc)
- Using AOP we can integrate other AOP frameworks(aspectJ ..etc) into spring.

6. What is Framework?

- It is a reusable semi-finished application that can be customized according to our business requirement

7. What is IOC? What is role of it in spring?

- **Inversion Of Control** is a design principle, which explains that an object creation, dependency injection, object lifecycle management, object destruction… etc. has to managed by an external entity.
- In Spring, Spring Container is the external entity, which implements principles of IOC.

8. How many ways we can give a value for dependencies?

In two ways

- Using Constructor
- Using Setter method

9. What is DI? What are different types of DI?

- The process of injecting dependences into the dependent objects is called Dependency Injection.
- Dependency Injection taking care by the spring container(IOC container) in spring.
    - In spring DI is generally achieved in 2 ways,
        - setter injection .
        - constructor injection.

10. What is setter injection? How to implement setter injection in a spring container?

If dependency is injected to dependent object via setter method, it is known as setter injection.

Step 1: In spring bean class declare the dependency variable.

Step 2: Define setter method for that variable.

Step 3: Use **<property>** tag in bean configuration file & supply the value.

11. How to implement constructor injection in a spring container?

If dependency is injected to dependent object via constructor, it is known as constructor injection.

Step 1: In spring bean class declare dependency variable.

Step 2: Define constructor which will take that variable as parameter.

Step 3: Use **<constructor-arg>** tag in bean configuration file & supply the value.

12. How inject primitives using spring Container?

- The **value** attribute or **<value>** sub element of **<property>** tag is used to inject primitives through setter injection.
- The **value** attribute or **<value>** sub element of <constructor-arg> tag is used to inject primitives through constructor injection.

13. How inject objects using spring Container?

- The **ref** attribute or **<ref>** sub element of **<property>** tag is used to inject objects through setter injection.
- The **ref** attribute or **<ref>** sub element of **<constructor-arg>** tag is used to inject objects through constructor injection.

14. What are the Benefits of Dependency Injection?

- Dynamically we can change dependency objects. Even dependency changing dependent object code remains same, thereby it is offering loose coupling among dependent and dependencies.

15. What is Spring Bean?

- Java class that is controlled by a spring container is known as Spring Bean. i.e. Java class that is configured in spring configuration file.

16. Which injection is better?

- Setter injection is always preferable because we can change the dependency value any number of times.
- But with the constructor we can't change the dependency object value.

17. What is the difference between constructor injection and setter injection?

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

18. What are the types of IOC containers?

There are 2 types of IOC containers in spring framework.

- BeanFactory
- Application Context

19. What is difference between BeanFactory And Application Context?

- BeanFactory is the **basic container** whereas ApplicationContext is the **advanced container**.
- BeanFactory is the **Lazy container** whereas ApplicationContext is the **Early container**.
- BeanFactory is the **Light Weight container** whereas ApplicationContext is the **Heavy Weight container** than BeanFactory based container.
- ApplicationContext extends the BeanFactory interface.
- ApplicationContext provides more facilities than BeanFactory such as integration with spring AOP, message resource handling for i18n etc. These facilities not supported by BeanFactory.
- In enterprise level application always **ApplicationContext** based container is used.

20. What is lazy-init attribute of <bean> tag?

- When we are using ApplicationContext based container, As and when we creates the spring container, it creates all the beans (Having scope value as '**singleton'**- we know that default scope is '**singleton**') which are configured in the spring configuration file.

- Sometimes we don't want to create the beans until and unless the request comes**(call getBean("bean-ref"))** for particular bean. This can be achieved with **lazy-init** attribute of **<bean>** tag.

- if we configure, **lazy-init="true"** for any bean, that bean object will not be created when we create the ApplicationContext based container. That will be created only when we call **getBean("bean-ref")**.

21. What are bean alias names?

Bean alias name can be used to define multiple names for a configured spring bean. We can define alias names with "**name**" attribute of **<bean>** tag or **<alias>** name tag.

Key notes on **id, name** attributes :

- **"id"** attribute is used to define the reference, to configured spring bean.
- **"id"** attribute value should be **unique,** whereas "**name"** no need to be **uniqure.**
- **"id"** value won't allow special characters (like /, _,*…etc). Whereas "**name"** attribute allows any kind of special characters.
- The ***"name"*** attribute is useful when we develop spring MVC based web applications, because as we know every URL should start with **"/".** So at that time we need to configure a spring bean whose reference should start with "/" character.
- Generally most of the cases we use ***id*** attribute only but not ***name*** attribute.

22. What are the different bean scopes in spring?

There are 5 bean scopes in spring framework.

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

Note: default scope of spring bean is singleton.

23. What is spring bean life cycle?

<u>Spring bean life-cycle</u>

- Spring container controls the life cycle of spring bean i.e. from instantiation to destruction.

Spring bean has the following life cycle states:

- **Instantiation**
  - Creation of the object
  - In spring objects are created by spring container
- **Injecting dependencies**
  - This is Dependency Injection
  - If any dependencies are configured to spring bean (either constructor or setter approach) those will be injected by the spring container.
- **Initialization**
  - When a bean is instantiated, it may be required to perform some initialization to get it into a usable state. If any initialization methods are configured (either with interfaces or with the configuration or with the annotations) those will be called by the spring container
- **Method Ready state**
  - In this stated Object ready to use, means we can business methods. Because in the previous states object was created, happens dependencies are injected, initialization, so now the bean ready serve.
- **Destruction**
  - When the bean is no longer from the required and is removed container, some cleanup may be required. If any destruction methods are configured (either with interfaces or with the configuration or with the annotations) those will be called by the spring container, when the container is shutdown.

- Initialization method, destruction method can be given to spring been in the following ways
    - By implementing spring framework given interfaces they are
        - InitializingBean
        - DisposableBean
    - By configuring custom initialization and destruction method in the spring configuring file, by using the following attributes <bean> tag
        - init-method
        - destroy-method
    - By using annotations given by JDK (which are implemented spring container)
        - @PreDestroy
        - @PostConstruct

24. It is used to verify all dependencies of been that are configured via injection are injected or not?

- To implement dependency checking to a spring bean we make use of **'dependency-check'** attribute of <bean> tag.

- This attribute takes any one of the four following values.
    - **none**(it won't check whether dependencies injected or not)
    - **simple**(it checks all primitive dependencies injected or not)
    - **objects**(it checks all the object type dependencies injected or not)
    - **all**(it checks both primitives and object type dependencies injected or not)

- If incase if we forgot to set any value either primitive or objective type it raise an exception called '**UnsatisfiedDependencyException'**.

25. What is factory method? And it's types?

- Generally we create the objects by calling the constructor. Even we can create the objects by calling some factory methods on the objects.
- Factory methods are of two types.
    - Static factory method
        - If the factory method contains static as access specifier, then that method is called static factory method.
    - non-static factory method
        - If the factory method not contains static as access specifier, then that method is called non-static factory method.

26. How to configure static factory method in spring?

Using **'factory -method'** attribute of **<bean >**tag. We can configure static factory method in spring.
Example:

27. How to configure on static factory method in spring?
Using **factory –bean ' and 'factory -method'** attribute of **<bean >**tag. we can configure non static factory method in spring.
Example:

28. What is autowiring in spring? What are the autowiring modes?

Auto wiring enables the programmer to inject the bean automatically. We don't need to write explicit injection logic.
Let's see the code to inject bean using dependency injection.
<bean id="emp" class="com.javatpoint.Employee" autowire="byName" />

The autowiring modes are given below:

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

Note : The autodetect mode is deprecated since spring 3

29. What are inner beans in Spring?
A <bean/> element inside the <property/> or <constructor-arg/> elements defines a so-called inner bean. An inner bean definition does not require a defined id or name; the container ignores these values. It also ignores the scope flag. Inner beans are always anonymous and they are always scoped as prototypes.

30. What are different ways to configure a class as Spring Bean?

There are three different ways to configure Spring Bean.

- XML Based Configuration
- Annotation-based configuration
- Java-based configuration

- **XML Configuration**: This is the most popular configuration and we can use bean element in context file to configure a Spring Bean. For example:

```
<bean name="myBean" class="com.balaji.spring.beans.MyBean"></bean>
```

- **Java Based Configuration**: If you are using only annotations, you can configure a Spring bean using @Bean annotation. This annotation is used with @Configuration classes to configure a spring bean. Sample configuration is:

```
@Configuration
@ComponentScan(value="com.journaldev.spring.main")
public class MyConfiguration {

   @Bean
   public MyService getService(){
      return new MyService();
   }
}
```

To get this bean from spring context, we need to use following code snippet:

```
AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(
      MyConfiguration.class);
MyService service = ctx.getBean(MyService.class);
```

- **Annotation Based Configuration**: We can also use @Component, @Service, @Repository and @Controller annotations with classes to configure them to be as spring bean. For these, we would need to provide base package location to scan for these classes. For example:

```
<context:component-scan base-package="com.balaji.spring" />
```

31. How can you inject a Java Collection in Spring? Give example?

Spring offers four types of collection configuration elements which are as follows:

<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.

Let’s see example of each type.

```
<beans>
  <!-- Definition for javaCollection -->
  <bean id="javaCollection" class="com.howtodoinjava.JavaCollection">
    <!-- java.util.List -->
    <property name="customList">
      <list>
        <value>INDIA</value>
        <value>Pakistan</value>
        <value>USA</value>
        <value>UK</value>
      </list>
    </property>
   <!-- java.util.Set -->
   <property name="customSet">
      <set>
        <value>INDIA</value>
        <value>Pakistan</value>
        <value>USA</value>
        <value>UK</value>
      </set>
    </property>
    <!-- java.util.Map -->
    <property name="customMap">
      <map>
        <entry key="1" value="INDIA"/>
        <entry key="2" value="Pakistan"/>
        <entry key="3" value="USA"/>
        <entry key="4" value="UK"/>
      </map>
     </property>
     <!-- java.util.Properties -->
   <property name="customProperies">
      <props>
         <prop key="admin">admin@nospam.com</prop>
         <prop key="support">support@nospam.com</prop>
      </props>
   </property>
  </bean>
</beans>
```

32. How do you turn on 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/>.

33.. What does @Required annotation mean?

- Spring's dependency checking in bean configuration file is used to make sure all properties of a certain types (primitive, collection or object) have been set. In most scenarios, you just need to make sure a particular property has been set, but not all properties… For this case, you need @Required annotation.

- Simply apply the @**Required** annotation will not enforce the property checking, you also need to register an **RequiredAnnotationBeanPostProcessor** to aware of the **@Required** annotation in bean configuration file.

- The RequiredAnnotationBeanPostProcessor can be enabled in two ways.

1. Include <context:annotation-config />

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

- **<beans ... >**
- **...**
- **<context:annotation-config />**
- **...**
- **</beans>**

2. Include RequiredAnnotationBeanPostProcessor

- Include 'RequiredAnnotationBeanPostProcessor' directly in the bean configuration file.

- **<beans ...>**
- **..................**
- **<bean**
- **class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"/>**
- **..................**
- **</beans>**

34. What does @Autowired annotation mean?

In Spring, you can use the @Autowired annotation to auto wire bean on the setter method, constructor or a field. Moreover, it can autowired property in a particular bean.

Note: The @Autowired annotation is auto wire the bean by matching data type.

To enable **@Autowired**, you have to register '**AutowiredAnnotationBeanPostProcessor**', and you can do it in two ways :

1. Include <context:annotation-config />
2. Include AutowiredAnnotationBeanPostProcessor

35. What does @Qualifier annotation do?

- In Spring, @Qualifier means, which bean is qualify to autowired on a field.
- 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. Below is an example to show the use of @Qualifier annotation.

36. What is **PropertyPlaceholderConfigurer ?**

PropertyPlaceholderConfigurer

- Oftentimes, most Spring developers just put the entire deployment details (database details, log file path) in XML bean configuration file. But it is advisable to Externilize deployment details into some properties**.**

- To fix it, you can use the PropertyPlaceholderConfigurer class to externalize the deployment details into a properties file, and access from the bean configuration file via a special format – **${variable}**.

- You also can use **PropertyPlaceholderConfigurer** to share some constant variables to all other beans. For example, define your log file location in a properties file, and access the properties value from different beans configuration files via ${log.filepath}.

- **To load properties file we need to configure as follows**

```
<bean  class=
        "org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
                    <property name="location"
                                value="com/vidvaan/spring/ioc/properties/dbproperties.properties" >
      </property>
        </bean>

            OR

     <context:property-placeholder location=
            "classpath:com/vidvaan/spring/properties/dbproperties.properties"/>
```

- In above application we have only one properties file. If we have multiple properties files we need to configure as follows.

```
<bean class=
    "org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
            <property name="locations">
                <list>
                 <value>
    com/vidvaan/spring/ioc/properties/dbproperties01.properties
                 </value>
                 <value>
    com/vidvaan/spring/ioc/properties/dbproperties02.properties
                 </value>
                </list>
            </property>

      </bean>

                          OR

<context:property-placeholder location=
      "classpath:com/vidvaan/spring/ioc/properties/dbproperties01.properties,
              com/vidvaan/spring/ioc/properties/dbproperties02.properties"/>
```

Q.) What is p-namespace?

- Use the special “p-namespace” to limit the amount of XML you have to write, to configure your beans.
- Instead of using nested <**property/>** elements, using the p-namespace you can use attributes as part of the <**bean/>** element that describe your property values.
- P-namespace is used for setter injection
- It is a short cut for setter injection.

Q.) What is c-namespace?

- The ***c-namespace***, newly introduced in Spring 3.1, allows usage of inlined attributes for configuring the constructor arguments rather then nested <**constructor-arg/>** elements.
- C-namespace is used for constructor injection
- It is a short cut for constructor injection.

What is Spring Expression Language ? How define in spring Application?

The Spring Expression Language (SpEL for short) is a powerful expression language that supports querying and manipulating an object graph at runtime. SpEL expressions can be used with XML or annotation based configuration metadata for defining BeanDefinitions. In both cases the syntax to define the expression is of the form #{ <expression string> }.

- In annotation mode, you use @Value to define Spring EL. In this case, you inject a String and Integer values.
- To inject String, integer and bean into property, both in XML and annotation we can use **Spring Expression Language(SpEL)**.
- In Spring EL, you can reference a bean, and nested properties using a '**dot (.)**' symbol. For example, "**bean.property_name**".
- Spring expression language (SpEL) allow developer uses expression to execute method and inject the method returned value into property, or so called "**SpEL method invocation**".
- Spring EL supports most of the standard mathematical, logical or relational operators. For example,
  - **Relational operators** – equal (==, eq), not equal (!=, ne), less than (<, lt), less than or equal (<= , le), greater than (>, gt), and greater than or equal (>=, ge).
  - **Logical operators** – and, or, and not (!).
  - **Mathematical operators** – addition (+), Subtraction (-), Multiplication (*), division (/), modulus (%) and exponential power (^).
- Spring EL supports **ternary operator** , perform "**if then else**" conditional checking. For example,

condition ? **true** : **false**

- We can inject **Map** and **List elements using** SpEL.
- Spring EL supports regular expressions using a simple keyword "**matches**", which is really awesome! For examples,

```
@Value("#{'100' matches '\\d+' }")
private boolean isDigit;
```

It test whether '**100**' is a valid digit via regular expression '**\\d+**'.