Spring

1. **Spring in Java: -**
   - Spring is a lightweight framework it can be described as framework of the frameworks. Because it provides support to various frameworks such as hibernate, struts, hibernate, EJB etc.,
   - The spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEBMVC etc.,
   - Spring is most popular application development framework for enterprise java.

   Benefits Of Spring: -
   - Spring helps developers to develop enterprise- class applications using POJOs (The main use of POJO is we no need to have heavy containers like EJB etc.., but we can make use to run on light weight containers).
   - Spring does not reinvest new tools , instead it will use the existing technologies like ORM framework, logging frameworks, JEE, Quartz and JDK timers.
   - Spring uses light weight containers like IOC for developing applications, the benefit is its good to develop and deploy applications on computers with limited memory and CPU resources.
2. **IOC container(Inversion of control)**
   - IOC container is responsible to instantiate, configure and assemble the objects.
   - The IOC container gets information from the XML file and works accordingly.
   - The Important functionalities of IOC container is
     1. To instantiate the application class.
     2. To configure the Objects
     3. To assemble dependencies between the objects

     Explanation of IOC: -

     Internally IOC container will go to the specified xml file and Searches for the specified class id and Instantiates the class, and loads the class, implements the methods or sets the values to the attributes and destroys the object.

3. **Dependency injection: -**

- Dependency Injection uses to connect two classes together and at the same time keeping them independent.
- Lets have a look on there two words separately here dependency part express the meaning of association between two classes.
    - For Examle class A is dependent of class B
      Lets look at second part, injection. All this means is, class B will get injected into class A by the IOC.
- Dependency injection can achieved in two ways described below
    - By using constructor.
    - By using Setters(Post Construction)
- Where constructor is used for injection for mandatory initialization in other hand setter used for optional injection.
- DI makes our application loosely coupled
- DI is a design pattern which removes the dependency of the program.
- In DI We provide the information from the external source such as XML file.

4.Different ways of Dependency injection using XML file Configurations: -

1. Constructor based dependency injection: -

***** We can inject the dependency by constructor by using <constructor-arg> sub element of <bean> is used for constructor injection.

* Here we can inject

1. Primitive and String based values

2. Dependent Object

3. Collection values etc.,

* for constructor injection we have to declare a suitable constructor to inject the targeted values.

* After creation of class we have to create tag of constructor-arg in xml file inside specific bean and we can insert values by using(value) attribute and object by using(ref) attribute.

Ex:

```
Class A {}

Class B {

A a;

Int I;

B(A a , int i){
this.a = a;

this.i = I;

}}
```

```
<bean id = “a” class = “FQN of A”> </bean>

<bean id = “b” class=”FQN of B”>

<constructor-arg index = “0” value = “a” />

<constructor-arg index = “1” value = “b”/>

</bean>
```

- In above scenario we achieved has-a relation between class A and B and then we injected dependent object through parameter of a constructor.

2.By using Setters:

* We can inject the dependency by using setter methods also.

* By using setter we can inject following values

1. primitive and string-based values.

2. Dependent object

3. Collection values etc.

* for setter injection we need to declare setter method for each property of specific class.

* then inside the respective bean tag we name to declare<property> sub element where we can pass the value to the properties by providing its variable name to name attribute.

* here name points the attribute where value specifies the data which is going to assign to it.

Class A{}

Class B{

Int I;

A a;

Create setter methods

}

```
<bean id = “a” class = “FQN of A”> </bean>
<bean id = “b” class=”FQN of B”>
<property name = “i” value = “11” />
<property name=”a”  ref = “a”/>
</bean>
```

Bean Factory: -

- BeanFactory is an interface present in “org.springframework.beans.factory.BeanFactory”.
- It is having an implementation class XmlBeanFactory
- It is a simple container which provides basic support for dependency injection.
- To create XMLBeanFactory Object we require ClassPathResource we have to provide the xml file name along with extension.
- Then we have to pass the object reference to the XmlBeanFactory(“classpathresource ref”) to instantiate BeanFactory.

ApplicationContext: -

- Application context is a advanced container which is also used to mange the life cycle of bean.
- ApplicationContext is an Interface which implements BeanFactory and present in org.springframework.context.ApplicationContext.
- Generally ApplicationContext is in more use compare to BeanFactory.
- There are so many implementation classes for the ApplicationContext some of those are
  1. ClassPathXmlApplicationContext
  2. AnnotationConfigApplicationContext
- Based on requirement we choose implementation context.

Difference between BeanFactory and ApplicationContext

BeanFactory	ApplicationContext
1. BeanFactory is an Interface. 2. Support lazy loading 3. Doesn't support annotation based dependency injection. 4. It doesn't support internalization	1. Sub interface of BeanFactory 2. Support Aggressive loading. 3. Support annotation based dependency injection. 4. It supports internalization.

Different Bean Scopes in Spring: -

Scope	**Description**
1. Singleton	The bean instance will be only once and same instance will be returned by the IOC container. It is the default scope.
2. Prototype	The bean instance will create each time new when we request.
3. Request	The bean instance will be created per HTTP request
4. Session	The bean instance will be created per HTTP session

init-method attriute of bean tag in spring.

- The init-method attribute used for to specify a method that is to be called on bean immediately upon instantiation.
- We can declare init-attribute inside bean tag of the xml file of spring.
- We have to pass method name as value to the init-method.

For example:-

Class A{ public void m1(){}}

In xml file

<bean id = “a” class = “FQN of A” init-method= “m1”></bean>

Use of destroy-method attribute of bean tag in spring: -

- The destroy-method is called before the bean is removed from the countainer.
- We use declare destroy-method attribute inside a bean tag of xml file and can initialize with a method name.
- The method which initialized well get executed before bean instance is removed from container.

  Example:-

  Class A{public void k1()}

  In xml file

<bean id = “a” class = “FQN of A” destroy-method= “m1”></bean>

Dependency injection of a variable using annotation

i. **Injecting a value via variable**
   a. We can inject the values to the variable by using @value annotation for that we need to pass value attribute and value in double quotes.

   Example:

```
@Component
Class A{
@value(value= “10”)
Int I ;
}
```

ii. Injecting value via constructor

We have to use same annotation for constructor injection also but place will be inside the constructor paranthasis

Example:

```
@Component
Class A{
Int i;
A(@value(value= “10”) int i){
this.i = I;
}
}
```

Here in above Variable i is assigned with 10.

iii. Injecting value for the variable via setter:

We can inject the value to the variable using setter also.

Example:

```
@Component
Class A{
Int i;
@value(value= “10”)
Public void setI(int i)
{
        this.i = i;
}
}
```

Injection of an object using annotation.

i. By using variable

- For both variable injection and object injection annotation places are same but only difference is annotation for object injection we use @Autowired annotation.

Example:-

```
Class A {}
Class B{
    @Autowired
    A a;
}
```

- For object instantiation we need not to pass any values. So there should be default or zero parameterized constructor is mandatory in dependent class.

ii. Using Constructor

```
Class A{}
@component
Class B{
A a;
B(@Autowired A a){
    this.a = a;
}
}
```

iii. Using setter

```
Class A{}
@component
Class B{
A a;
@Autowired
Public void setA(A a){
this.a = a;
}
}
```

- We have to use @component on the dependent class before using @Autowired or it's object declaration else it throw exception.
- Any object we want to get it's instance from container either we can declare bean tag in xml file or we should use annotation.