Spring-Core

1. **What is Spring ?**
   - ▼ **Ans**
     - **Spring is an Open Source, Lightweight and Non-Invasive framework which is used to achieve Loose Coupling. By using Spring, We can simplify the development of JEE Application.**
     - **The Main Advantages or Objective of Spring are IOC and Dependency Injection. Using which, We can achieve Loose Coupling.**
     - **Spring contains different modules using which we can simplify Transection Manage Persistence Logic and Business logic. following are the different Spring Module**
       1. **Spring Core**
       2. **Spring ORM**
       3. **Spring AOP**
       4. **Spring Context**
       5. **Spring JDBC**
       6. **Spring Hibernate**
       7. **Spring MVC**
2. **What are the advantages of Spring ?**
   - ▼ **Ans**
     1. **Spring will provide Pre-Defined Template like JDBC template, Hibernate template and JPA repository using which we can avoid the basic steps of the Execution.**
     2. **Spring is an Open-Source Framework.**
     3. **Spring is a Lightweight Framework, Because it is POJO implementation.**
     4. **If we develop an application using Spring, It will be easy to test and easy to develop. Because of Dependency Injection.**
     5. **Spring will provide a powerful Abstraction.**
3. **What is Coupling ?**
   - ▼ **Ans**
     - **The Dependency between any two Entities is called as Coupling.**
4. **Example for Tight Coupling, Loose Coupling and Dependency Injection ?**
   - ▼ **Ans**
     - ▼ **Example will Result in Tight Coupling b/w Ride and Vehicle**

```
package org.Spring.TightCouplingApp;

public interface Vehicle
{
  void start();
}
class Bike implements Vehicle
{
  @Override
  public void start()
```

```
  {
    System.out.println("Bike has been Started");
  }
}
class Car implements Vehicle
{
  @Override
  public void start()
  {
    System.out.println("Car has been Started");
  }
}
class Ride
{
  Vehicle v =new Car();
  void StartRide()
  {
    v.start();
  }
}
class Driver
{
  public static void main(String[] args)
  {
    Ride r = new Ride();
    r.StartRide();
  }
}
```

▼ **Example will Result in Lose Coupling b/w Ride and Vehicle**

```
package org.Spring.LooseCouplingApp;

public interface Vehicle
{
  void start();
}
class Bike implements Vehicle
{
  @Override
  public void start()
  {
    System.out.println("Bike has been Started");
  }
}
class Car implements Vehicle
{
  @Override
  public void start()
  {
    System.out.println("Car has been Started");
  }
}
class Ride
{
  Vehicle v;

  public Vehicle getV()
  {
    return v;
  }
  public void setV(Vehicle v)
  {
    this.v = v;
  }
  void StartRide()
  {
    v.start();
  }
}
class Driver
{
  public static void main(String[] args)
  {
    Ride r = new Ride();
    r.setV(new Bike()); //Dependency Injection
    r.StartRide();
```

```
    }
}
```

- In the above example Ride class is loosely coupled with Vehicle. Because the Object is Injected during the Runtime by Driver Class.

5. **What is Dependency Injection ?**

   ▼ Ans

   - The process of Injecting the reference of one Object into another during the Runtime is called as Dependency Injection.

6. **What is Inversion Of Control (IOC) ?**

   ▼ Ans

   - It is the process of transferring the control to another class or object, So that the class can focus on the task which is important. We can achieve Inversion of Control with the help of Dependency Injection.

   Note : The Output of IOC is Loose Coupling which can be achieved with the help of Dependency Injection.

7. **What Spring Container ?**

   ▼ Ans

   - Spring Container is an Engine which is used to manage the Life-Cycle of the Spring Beans. We have 2 types of Spring Container
     1. IOC Container
     2. MVC Container

8. **What is Spring Bean ?**

   ▼ Ans

   - It is an Object which is created by Spring Container, the Life-Cycle of Spring Bean is managed by Spring Container.
   - If there is any Dependency present, the Dependency Injection is also done by Spring Container.

9. **What is IOC Container ?**

   ▼ Ans

   - It is a Spring Core Container which is used to managed the Life-Cycle of Spring Bean, If there is any Dependency present, It will Inject the Dependency.

10. **What are the Types of IOC Containers ?**

    ▼ Ans

    We have 2 types of IOC Containers

    1. Bean Factory
    2. Application Context

    - If we want our Spring Container to manager the Spring Bean Life-Cycle, We need to provide the META data above the Bean. We need to configure the bean. We can configure in 2 ways.
      1. XML Configuration
      2. Annotation Configuration

11. **What is BeanFactory ?**

    ▼ Ans

    - It is an Interface present in org.springframework.beans.factory.BeanFactory

- This is a factory for the Bean. It will create the Bean, manage the Life-Cycle, Inject the Dependency if necessary.
- resource → It is an Interface present in org.springframework.core.io.Resource
- resource have implementation classes, following are the important implementation classes
    1. ClassPathResource
    2. FileSystemResource

12. **What is XmlBeanFactory ?**

    ▼ Ans

    - XmlBeanFactory is the implementation class of BeanFactory Interface.
    - It is present inside org.springframework.beans.factory.xml.XmlBeanFactory package.

13. **What is getBean() ?**

    ▼ Ans

    - It is a method defined in BeanFactory Interface.
    - This method is Overloaded, following are the Overloaded variants of getBean()

Method Signature	Return type
getBean(String)	object
getBean(class<T>)	T
getBean(String , class<T>)	T

14. **Code to Create a Car Object by BeanFactory Interface (IOC Container) ?**

    ▼ Ans

    1. Create Simple maven Project with Group Id → "org.jsp.SpringDemo" and Artifact ID → "SpringDemo"
    2. We need to add one Dependency into pom.xml that is SpringContext. This Dependency we will get in Maven Repository in Google → Open Maven Repository and Search "Spring Context" → Check for the most used version (5.3.18) → Copy the Dependency past it inside <dependencies></dependencies> in pom.xml then Save it and It will automatically download Maven dependencies.

        ▼ pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocatio
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.jsp.SpringDemo</groupId>
  <artifactId>SpringDemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
  <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.18</version>
</dependency>
  </dependencies>

</project>
```

    3. Create one Class inside src/main/java name it as Car

        ▼ Car.java

```
package org.jspSpringDempApp;

public class Car
```

```
{
  public void Start()
  {
    System.out.println("Car has been Started");
  }
}
```

4. **Create one XML file inside src/main/java name it as "mySpringConf.xml" and Add one link which is given by Spring. We will get that link in Satish Sir Git-Hub just copy past the Entire code in our xml file. → "https://github.com/sathishnyadav/supporting-files/blob/master/spring-xsd.xml.txt"**
5. **After Copy pasting from Git-hub. We need to Configure our Bean Class there. We have to configure our Bean inside <beans></beans> Tag in mySpringConf.xml. Just like this <bean id="myCar" class="org.jspSpringDempApp.Car"></bean>**
   a. **id → Unique**
   b. **class → FQCN**

   ▼ **mySpringConf.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<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"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-3.0.xsd">
  <bean id="myCar" class="org.jspSpringDempApp.Car"></bean>
</beans>
```

6. **Create another one Class inside src/main/java name it as TestCar and Run It**

   ▼ **TestCar.java**

```
package org.jspSpringDempApp;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class TestCar
{
  public static void main(String[] args)
  {
    Resource r = new ClassPathResource("mySpringConf.xml");
    BeanFactory factory = new XmlBeanFactory(r);
    Car c = (Car) factory.getBean("myCar");
    c.Start();
  }
}
```

15. **Variable Initialization ?**

    ▼ **Ans**

    - **It is the process of assigning the value of a Variable.**
    - **The variable initialization can be done with the help of Spring Container in 2 ways Using XML**
      1. **Using setter**
      2. **Using Constructor**

16. **Variable Initialization Using Setter ?**

    ▼ **Ans**

- **To Achieve the variable initialization using setter we need to make use of property tag(<property></property>) which is the sub tag of Bean Tag (<bean></bean>)**

▼ **Person.java**

```
package org.jspSpringDempApp;

public class Person
{
private int age;
private String name;
public int getAge() {
  return age;
}
public void setAge(int age) {
  this.age = age;
}
public String getName() {
  return name;
}
public void setName(String name) {
  this.name = name;
}
}
```

▼ **mySpringConf.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<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"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">
  <bean id="myCar" class="org.jspSpringDempApp.Car"></bean>
  <bean id="myPerson" class="org.jspSpringDempApp.Person">
    <property name="age" value="24"></property>
    <property name="name" value="manu"></property>
  </bean>
</beans>
```

▼ **TestPerson.java**

```
package org.jspSpringDempApp;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class TestPerson
{
  public static void main(String[] args)
  {
    Resource r = new ClassPathResource("mySpringConf.xml");
    BeanFactory factory = new XmlBeanFactory(r);
    Person p = factory.getBean("myPerson", Person.class);
    System.out.println("Name : "+p.getName());
    System.out.println("Age : "+p.getAge());
  }
}
---------OUTPUT-----------
Name : manu
Age : 24
```

17. **Variable Initialization Using contractor ?**

▼ **Ans**

- **To Initialize the variable using constructor, We need to make use of <constructor-arg></constructor-arg> tag which is the sub tag of <bean></bean> tag.**
- **If we use constructor arg then we must have a parameterized constructor inside the class otherwise we will get exception.**

▼ **Employee.java**

```
package org.jspSpringDempApp;

public class Employee {
  private double salary;
  private String desg;

  public Employee(double salary, String desg) {
    this.salary = salary;
    this.desg = desg;
  }
  public Employee() {
  }
  public double getSalary() {
    return salary;
  }
  public void setSalary(double salary) {
    this.salary = salary;
  }
  public String getDesg() {
    return desg;
  }
  public void setDesg(String desg) {
    this.desg = desg;
  }
}
```

▼ **mySpringConf.xml**

```
<bean id="myEmployee" class="org.jspSpringDempApp.Employee">
    <constructor-arg name="salary" value="1500.0" />
    <constructor-arg name="desg" value="SE" />
  </bean>
```

▼ **TestEmployee.java**

```
package org.jspSpringDempApp;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class TestEmployee {
  public static void main(String[] args) {
    Resource r = new ClassPathResource("mySpringConf.xml");
    BeanFactory factory = new XmlBeanFactory(r);
    Employee e = factory.getBean("myEmployee", Employee.class);
    System.out.println("Salary : " + e.getSalary());
    System.out.println("Designation : " + e.getDesg());
  }
}
```

18. **What is Application Context ?**

▼ **Ans**

- **It is an Interface present in org.springframework.context package. It is the sub Interface of BeanFactory.**
- **Application content have the following implementation class**
    1. **ClassPathXMLApplicationContext**

2. **FileSystemApplicationContext**
3. **AnnotationConfigApplicationContext**

▼ **Creation of Employee, Person and Car Using Application Context**

```
package org.jspSpringDempApp;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestApplicationContext {
  public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("mySpringConf.xml");
    Car c = context.getBean("myCar", Car.class);
    c.Start();
    System.out.println("------------");
    Employee e = context.getBean("myEmployee", Employee.class);
    System.out.println("Salary : " + e.getSalary());
    System.out.println("Designation : " + e.getDesg());
    System.out.println("------------");
    Person p = context.getBean("myPerson", Person.class);
    System.out.println("Name : " + p.getName());
    System.out.println("Age : " + p.getAge());
  }
}
```

19. **Dependency Injection Using Spring container ?**

▼ **Ans**

- **It is the process of injecting the reference of one object into another object at the runtime by Spring Container achieve the loose coupling.**
- **We achieve dependency Injection in 2 ways with a help of XML**
  1. **Setter Injection**
  2. **Constructor Injection**

20. **Dependency Injection Using Setter ?**

▼ Ans

- **To the dependency injection using setter, We need to make use property tag which is the sub tag bean tag.**

▼ **Example for dependency injection using setter**

▼ **Car.java**

```
package org.jsp.SpringApp;

public class Car
{
  private Engine e;
  public void StartCar()
  {
    e.start();
  }
  public Engine getE() {
    return e;
  }
  public void setE(Engine e) {
    this.e = e;
  }
}
```

▼ **Employee.java**

```
package org.jsp.SpringApp;

public class Engine
{
  public void start()
  {
    System.out.println("Engine has been Started");
  }
}
```

▼ **mySpringConf.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<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"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">
  <bean id="myCar" class="org.jsp.SpringApp.Car">
  <property name="e" ref="myEngine"></property>
  </bean>
  <bean id="myEngine" class="org.jsp.SpringApp.Engine"></bean>
</beans>
```

▼ **TestCar.java**

```
package org.jsp.SpringApp;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestCar
{
  public static void main(String[] args)
  {
    ApplicationContext context = new ClassPathXmlApplicationContext("mySpringConf.xml");
    Car c = context.getBean("myCar",Car.class);
    c.StartCar();
  }
}
```

21. **Dependency Injection constructor with the help of XML ?**

▼ **Ans**

▼ **PanCard.java**

```
package org.jsp.SpringApp.XML;

public class PanCard {
  private String number;
  private String place;

  public void displayAttributes() {
    System.out.println("Number : " + number);
    System.out.println("Place : " + place);
  }

  public String getNumber() {
    return number;
  }

  public void setNumber(String number) {
    this.number = number;
  }

  public String getPlace() {
    return place;
  }
```

```
  public void setPlace(String place) {
    this.place = place;
  }
}
```

▼ **Person.java**

```
package org.jsp.SpringApp.XML;

public class Person {
  private PanCard card;

  public Person() {

  }

  public Person(PanCard card) {
    this.card = card;
  }

  public PanCard getCard() {
    return card;
  }

  public void setCard(PanCard card) {
    this.card = card;
  }
}
```

▼ **personCard.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<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"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">
  <bean id="myCard" class="org.jsp.SpringApp.XML.PanCard">
    <property name="number" value="abc123"></property>
    <property name="place" value="India"></property>
  </bean>
  <bean id="myPerson" class="org.jsp.SpringApp.XML.Person">
    <constructor-arg name="card" ref="myCard"></constructor-arg>
  </bean>
</beans>
```

▼ **Test.java**

```
package org.jsp.SpringApp.XML;

import org.jsp.SpringApp.Car;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test
{
  public static void main(String[] args)
  {
    ApplicationContext context = new ClassPathXmlApplicationContext("personCard.xml");
    Person p = context.getBean("myPerson",Person.class);
    p.getCard().displayAttributes();
  }
}
```

22. **Collection Injection Using Spring Container ?**

▼ **Ans**

- **It can be done using either setter or constructor.**

▼ **Employee.java**

```
package org.jsp.SpringApp.SpringContainer;

import java.util.List;
import java.util.Map;
import java.util.Set;

public class Employee {
  List<String> names;
  Set<Integer> ids;
  Map<Integer, String> details;

  public List<String> getNames() {
    return names;
  }

  public void setNames(List<String> names) {
    this.names = names;
  }

  public Set<Integer> getIds() {
    return ids;
  }

  public void setIds(Set<Integer> ids) {
    this.ids = ids;
  }

  public Map<Integer, String> getDetails() {
    return details;
  }

  public void setDetails(Map<Integer, String> details) {
    this.details = details;
  }

}
```

▼ **employee.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<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"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">

  <bean id="emps" class="org.jsp.SpringApp.SpringContainer.Employee">
    <!-- INJECTION THE LIST -->
    <property name="names">
      <list>
        <value>Pushpa</value>
        <value>Rocky</value>
        <value>Balayya</value>
      </list>
    </property>
    <!-- INJECTION THE SET -->
    <property name="ids">
      <set>
        <value>101</value>
        <value>102</value>
        <value>103</value>
      </set>
    </property>
    <!-- INJECTION THE MAP -->
    <property name="details">
      <map>
        <entry key="101" value="Pushpa"></entry>
        <entry key="102" value="Rocky"></entry>
        <entry key="103" value="Balayya"></entry>
      </map>
```

```
    </property>
  </bean>
</beans>
```

▼ **TestEmployee.java**

```
package org.jsp.SpringApp.SpringContainer;

import org.jsp.SpringApp.XML.Person;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestEmployee
{
  public static void main(String[] args)
  {
    ApplicationContext context = new ClassPathXmlApplicationContext("employee.xml");
    Employee emps = context.getBean("emps",Employee.class);
    System.out.println(emps.getNames());
    System.out.println(emps.getIds());
    System.out.println(emps.getDetails());
  }
}
```

23. **InitializingBean ?**

▼ **Ans**

- **It is an Interface present in org.springframework.factory.**
- **This Interface need to be Implemented by the Beans which should react once the properties are set by the BeanFactory.**
- **This Interface abstract method after properties set() which will be called once the properties are set by BeanFactory.**

24. **DisposableBean ?**

▼ **Ans**

- **This Interface present in org.springframework.beans.factory package.**
- **beans need to implement this Interface if all the resource to be released on distraction of object**
- **This Interface have an abstract method destroy(), Which will be called by BeanFactory on destraction of Object.**

25. **Difference BeanFactory and ApplicationContext ?**

▼ **Ans**

BeanFactory	ApplicationContext
1. It is root container or Interface for Spring Containers	1. It is the sub-Interface of Listable BeanFactory which is a child of BeanFactory.
2. The Objects will be created only on demand. (Lazy Loading)	2. This Objects will be created on-startup. (Eager Loading)
3. This light weight container, Because the objects will be created or loaded on demand.	3. It is heavy container because Objects will be created on Start-Up
4. It supports only 2 types Bean scopes (Singleton and Prototype)	4. It supports all types Bean scopes.

26. **Bean Scope**

▼ **Ans**

- **It is used to decide the creation of objects (scope of the Bean object) following are the different bean scopes.**

1. **Singleton**
2. **Prototype**

Note: By default scope of the bean Singleton

27. **Singleton ?**

▼ **Ans**

- **If we make Bean Scope has Singleton only one Object will be created per container on every request the reference of some object will be written.**

28. **Prototype ?**

▼ **Ans**

- **A new object will be returned for every request.**

29. **Example Spring Bean Life Cycle ?**

▼ **Ans**

1. **Class will be loaded into JVM memory.**
2. **Object will be created for the Spring Bean.**
3. **Inti() method will be called once after the initialization is completed.**
4. **destroy() method will be called once the container is closed.**

▼ **Demo.java**

```
package org.jsp.SpringDemoApp.SpringLifeCycle;

import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;

public class Demo implements InitializingBean, DisposableBean {
  static {
    System.out.println("Demo class is Loaded into Memory");
  }

  public Demo() {
    System.out.println("Demo Object is getting Created");
  }

  public void destroy() throws Exception {
    System.out.println("Demo is Destroyed");
  }

  public void afterPropertiesSet() throws Exception {
    System.out.println("Demo is Iniatialized");
  }
}
```

▼ **demo.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<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"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">
           <bean id="demo" class="org.jsp.SpringDemoApp.SpringLifeCycle.Demo"></bean>
</beans>
```

▼ **Test.java (Spring Bean Life Cycle Using BeanFactory)**

```
package org.jsp.SpringDemoApp.SpringLifeCycle;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class Test {
  public static void main(String[] args) {
    Resource r = new ClassPathResource("demo.xml");
    BeanFactory factory = new XmlBeanFactory(r);
    System.out.println(factory);
  }
}
--------------------OUTPUT--------------------
org.springframework.beans.factory.xml.XmlBeanFactory@2471cca7: defining beans [demo]; root of factory hierarchy
```

▼ **TestApplicationContext.java (Spring Bean Life Cycle Using ApplicationContext)**

```
package org.jsp.SpringDemoApp.SpringLifeCycle;

import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestApplicationContext {
  public static void main(String[] args) {
    ConfigurableApplicationContext context = new ClassPathXmlApplicationContext("demo.xml");
    context.close();
  }
}
-----------------OUTPUT-----------------
Demo class is Loaded into Memory
Demo Object is getting Created
Demo is Iniatialized
Demo is Destroyed
```

30. **Spring Life Cycle Configuration Using XML ?**

▼ **Ans**

- **We can configure the init() method and destroy() method for bean sing XML is 2 ways**

1. **Using the Attributes init method = " " & destroy method =" " which are attribute of <bean></bean> Tag**

   Note - This is specific to only one particular bean

2. **Using the Attributes default init method = " " default destroy method = " " which are the attribute of <bean> </bean> Tag**

   Note - It is applicable for all the configure beans

▼ **Testing.java**

```
package org.jsp.SpringApp.LifeCycleUsingXML;

public class Testing {
  public void hi() {
    System.out.println("Hi, This is init() method of Testing");
  }

  public void hello() {
    System.out.println("Hello, This is hello() method of Testing");
  }

  public void bye() {
    System.out.println("Bye, This is destroy() method of Testing");
  }
}
```

▼ **LifeCycleDemo.java**

```
package org.jsp.SpringApp.LifeCycleUsingXML;

public class LifeCycleDemo {
  public void hi() {
    System.out.println("Hi, This is init() method of LifeCycleDemo");
  }

  public void bye() {
    System.out.println("Bye, This is destroy() method of LifeCycleDemo");
  }
}
```

▼ **demo.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans default-init-method="hi" default-destroy-method="bye" xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">
  <bean id="testing" class="org.jsp.SpringApp.LifeCycleUsingXML.Testing"></bean>
  <bean id="lcd" class="org.jsp.SpringApp.LifeCycleUsingXML.LifeCycleDemo"></bean>
</beans>
```

▼ **TestApplicationContext.java**

```
package org.jsp.SpringApp.LifeCycleUsingXML;

import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestApplicationContext {
  public static void main(String[] args) {
    ConfigurableApplicationContext context = new ClassPathXmlApplicationContext("demo.xml");
    context.close();
  }
}
```

31. **Bean Configuration using Annotation ?**

▼ **Ans**

1. **@Component**
   - **It is a class level annotation present in org.springframework.stereotype package.**
   - **It used to mark the class as a component.**
   - **If a class annotated with @Component then such classes are eligible for the auto detection by the Spring Container**

   ▼ **Code to create the Car object Using Spring Container with the help of Annotation Configuration**

   ▼ **Car.java**

```
package org.jsp;

import org.springframework.stereotype.Component;

@Component(value = "myCar")
public class Car {
  public void start() {
    System.out.println("Car is Started");
  }
}
```

▼ **Test.java**

```
package org.jsp;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Test {
  public static void main(String[] args) {
    ApplicationContext context = new AnnotationConfigApplicationContext("org.jsp");
    Car c = context.getBean("myCar", Car.class);
    c.start();
  }
}
```

2. **@Value**

- **This is an annotation which can be used for a field for a method or for constructor parameter.**
- **It is used to assign the default for the annotated elements.**
- **We can assign the value for variable using at rate value in 3 ways**
  1. **By using Setter**
  2. **By using Constructor**
  3. **By using Variable**

▼ **Car.java**

```
package org.jsp;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component(value = "myCar")
public class Car {
  String brand;
  @Value(value = "5655233.1")//By using Variable
  private double price;
  private String color;

  public String getColor() {
    return color;
  }

  @Value(value = "White") //By using Setter
  public void setColor(String color) {
    this.color = color;
  }

  public Car(@Value(value = "BMW") String brand)  //By using Constructor
  {
    this.brand = brand;
  }

  public void display() {
    System.out.println("Brand : " + brand);
    System.out.println("Price : " + price);
    System.out.println("Color : " + color);
  }
}
```

▼ **Test.java**

```
package org.jsp;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Test {
  public static void main(String[] args) {
```

```
    ApplicationContext context = new AnnotationConfigApplicationContext("org.jsp");
    Car c = context.getBean("myCar", Car.class);
    c.display();
  }
}
----------------------OUTPUT-------------------------
Brand : BMW
Price : 5655233.1
Color : White
```

32. **Dependency Injection Using Annotation ?**

▼ **Ans**

- **We can achieve Dependency Injection using Annotation in 3 Ways,**

1. **Variable Injection**
2. **Setter Injection**
3. **Constructor Injection**

33. **@Autowired ?**

▼ **Ans**

1. **It is an annotation present in org.springframework.beans.factory.annotation.Autowired package**
2. **It can be used for a field, constructor, method as well as parameter.**
3. **It is used to mark the field or setter or constructor as a autowired candidate to achieve Dependency Injection.**

34. **Example for Dependency Injection Using Annotation ?**

▼ **Ans**

▼ **PanCard.java**

```
package org.jsp;

import org.springframework.stereotype.Component;

@Component
public class PanCard
{
  void display()
  {
    System.out.println("This is your PanCard");
  }
}
```

▼ **Person.java**

```
package org.jsp;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component(value = "myperson")
public class Person
{
  @Autowired
  private PanCard card;

  public PanCard getCard() {
    return card;
  }

  public void setCard(PanCard card) {
    this.card = card;
  }
  public void displayPanCard()
  {
    card.display();
```

```
  }
  public Person()
  {

  }
  public Person(PanCard card)
  {
    this.card=card;
  }
}
```

▼ **Test.java**

```
package org.jsp;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Test
{
  public static void main(String[] args)
  {
    ApplicationContext context = new AnnotationConfigApplicationContext("org.jsp");
    Person p = context.getBean("myperson",Person.class);
    p.displayPanCard();
  }
}
--------------------OUTPUT-------------------------
This is your PanCard
```

35. **@Configuration ?**

▼ **Ans**

1. **It is a class level annotation present org.springframework.context.annotation package.**
2. **It is used to mark the class has a configuration class.**
3. **It indicates that the annotated class contains one or more @Bean methods present which will return the reference of an object. which are supposed to be managed by Spring Container.**

36. **@ComponentScan**

▼ **Ans**

- **It is a class level annotation present in org.springframework.context.annotation packagee.**
- **This annotation is used to represents the base packages in which the Spring Container has to scan for the beans.**

▼ **PanCard.java**

```
package org.jsp;

import org.springframework.stereotype.Component;

@Component
public class PanCard
{
  void display()
  {
    System.out.println("This is your PanCard");
  }
}
```

▼ **MyConfig.java**

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
@ComponentScan("org.jsp")
public class MyConfig
{

}
```

▼ **Test.java**

```
package org.jsp;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Test
{
  public static void main(String[] args)
  {
    ApplicationContext context = new AnnotationConfigApplicationContext(MyConfig.class);
    Person p = context.getBean("myperson",Person.class);
    p.displayPanCard();
  }
}
--------------------OUTPUT-------------------------
This is your PanCard
```

37. **@Bean ?**

▼ **Ans**

- **It is method level annotation present org.**
- **This annotation is used for method which produces a Bean which has to be managed by Spring Container.**
- **At the rate bean methods must be defined inside configuration classes**

▼ **MyConfig.java**

```
package org.Annotation;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan("org.Annotation")
public class MyConfig {
  @Bean
  public List<Integer> getList() {
    return new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 5));
  }
}
```

▼ **Employees.java**

```
package org.Annotation;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Employees {
  @Autowired
  public List<Integer> ids;

  public List<Integer> getIds() {
```

```
    return ids;
  }

  public void setIds(List<Integer> ids) {
    this.ids = ids;
  }

}
```

▼ **TestEmployees.java**

```
package org.Annotation;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class TestEmployees {
  public static void main(String[] args) {
    ApplicationContext context = new AnnotationConfigApplicationContext(MyConfig.class);
    Employees e = context.getBean(Employees.class);
    System.out.println("Ids : " + e.getIds());
  }
}
```

38. **@Primary ?**

▼ **Ans**

- **It is annotation in “org.springframework.context.annotation” package.**
- **This annotation can be used for a method or for a class.**
- **This can be used with @Component or @Bean.**
- **This annotation will be used to prefer a single bean whnevr we have more hen 1 qualifying bean for a specific dependency as Autowired candidate.**

39. **@Qualifier ?**

▼ **Ans**

- **It is an annotation present in “org.springframework.beans.factory.annotation” package.**
- **It can be used for a field, method, paramter also class.**
- **It can be used with @Autowired**
- **It is used to specify the qualifying bean whenever we have more then 1 qualifying bean**

Note → @Qualifier has given more preference then @Primary.

40. **Example Code for @Primary and @Qualifier ?**

▼ **Ans**

▼ **pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="h
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.jsp</groupId>
  <artifactId>SpringDemo4</artifactId>
  <version>0.0.1-SNAPSHOT</version>
      <dependencies>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.18</version>
    </dependency>
  </dependencies>
</project>
```

▼ Vehicle.java

```
package org.jsp;

public interface Vehicle
{
  void start();
}
```

▼ Bike.java

```
package org.jsp;

import org.springframework.stereotype.Component;

@Component(value = "myBike")
public class Bike implements Vehicle
{
  public void start()
  {
    System.out.println("Bike has been Started");
  }
}
```

▼ Car.java

```
package org.jsp;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;

@Component(value = "myCar")
@Primary
public class Car implements Vehicle
{
  public void start()
  {
    System.out.println("Car has been Started");
  }
}
```

▼ Ride.java

```
package org.jsp;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Ride
{
  @Autowired
  //@Qualifier("myBike") -> It executes bike class
  private Vehicle v;
  public void startRide()
  {
    v.start();
  }
}
```

▼ MyConfig.java

```
package org.jsp;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
```

```
@ComponentScan(basePackages = "org.jsp")
public class MyConfig
{

}
```

▼ **Test.java**

```
package org.jsp;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Test
{
  public static void main(String[] args)
  {
    ApplicationContext context = new AnnotationConfigApplicationContext(MyConfig.class);
    Ride r = context.getBean(Ride.class);
    r.startRide();
  }
}
```

- **dto (Data Transfer Object) used to create Entity Classes.**
- **dao (Data Access Object) used for Persistence Logic.**
- **service used for Business Logic.**
- **controller used for calling purpose.**

41. **User-App Project ?**

▼ **Ans**

1. **Create a Simple maven Project**
2. **Add these 3 dependencies in pom.xml**
   a. **Spring Context**
   b. **Hibernate Core Relocation**
   c. **MySQL Connector**

▼ **pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="h
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.jsp</groupId>
  <artifactId>user-app</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
    <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.6.5.Final</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.28</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.18</version>
    </dependency>
  </dependencies>
</project>
```

1. **Copy past code from Satish sir git-hub to persistence.xml**

▼ **persistence.xml**

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
  http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">

  <persistence-unit name="dev">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>      <!-- for caching -->
    <properties>
      <property name="javax.persistence.jdbc.driver"
        value="com.mysql.cj.jdbc.Driver" />
      <property name="javax.persistence.jdbc.url"
        value="jdbc:mysql://localhost:3306/UserApp1?createDatabaseIfNotExist=true"/>
      <property name="javax.persistence.jdbc.user" value="root" />
      <property name="javax.persistence.jdbc.password"
        value="admin" />
      <property name="hibernate.show_sql" value="true" />

      <property name="hibernate.hbm2ddl.auto" value="update" />
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.MySQL8Dialect" />
    </properties>
  </persistence-unit>
</persistence>
```

4. **Create 5 packages**

▼ **org.jsp.userapp**

▼ **UserConfig.java**

```
package org.jsp.userapp;

import javax.persistence.EntityManager;
import javax.persistence.Persistence;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "org.jsp.userapp")
public class UserConfig
{
  @Bean
  public EntityManager getEntityManager()
  {
    return Persistence.createEntityManagerFactory("dev").createEntityManager();
  }
}
```

▼ **org.jsp.userapp.controller**

▼ **SaveUser.java**

```
package org.jsp.userapp.controller;

import org.jsp.userapp.UserConfig;
import org.jsp.userapp.dto.User;
import org.jsp.userapp.service.UserService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class SaveUser
{
  public static void main(String[] args)
  {
     User user = new User();
```

```
    user.setName("ABC");
    user.setPassword("a123456");
    user.setPhone(123456);
    ApplicationContext context = new AnnotationConfigApplicationContext(UserConfig.class);
    UserService service = context.getBean(UserService.class);
    service.saveUser(user);
  }
}
```

▼ **org.jsp.userapp.dao**

▼ **UserDao.java**

```
package org.jsp.userapp.dao;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;

import org.jsp.userapp.dto.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class UserDao
{
  @Autowired
  EntityManager manager;

  public User saveUser(User user)
  {
    EntityTransaction transaction = manager.getTransaction();
    manager.persist(user);
    transaction.begin();
    transaction.commit();
    return user;
  }
  public User updateUser(User user)
  {
    return null;
  }
  public User getUserById(int id)
  {
    return null;
  }
  public User deleteUser(int id)
  {
    return null;
  }
  public List<User> FetchAllUser()
  {
    return null;
  }
  public User verifyUser(long phone, String password)
  {
    return null;
  }
}
```

▼ **org.jsp.userapp.dto**

```
package org.jsp.userapp.dto;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class User
{   @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int id;
  private String name;
```

```
  private long phone;
  private String password;
  public int getId() {
    return id;
  }
  public void setId(int id) {
    this.id = id;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public long getPhone() {
    return phone;
  }
  public void setPhone(long phone) {
    this.phone = phone;
  }
  public String getPassword() {
    return password;
  }
  public void setPassword(String password) {
    this.password = password;
  }

}
```

▼ **org.jsp.userapp.service**

▼ **UserService.java**

```
package org.jsp.userapp.service;

import java.util.List;

import org.jsp.userapp.dao.UserDao;
import org.jsp.userapp.dto.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserService
{
  @Autowired
  private UserDao dao;
  public User saveUser(User user)
  {
    return dao.saveUser(user);
  }
  public User updateUser(User user)
  {
    return dao.updateUser(user);
  }
  public User getUserById(int id)
  {
    return dao.deleteUser(id);
  }
  public List<User> findAllUser()
  {
    return dao.FetchAllUser();
  }
}
```