BY NAGOOR BABU

Inversion Of Control[IOC]

In general, in enterprise application if we need any service in our application then we have to put request to the service provider, where service provider has to create the required service then service provider has to send that service to the respective application.

In enterprise applications, Inversion of Control is a design pattern or a design principle, it will make service provider to identify the required services of the enterprise applications and it will make the service provider to create and inject the required services to the application with out expecting any request from the application.

IOC is available in the following two forms.

1. Dependency Lookup
2. Dependency Injection

1. Dependency Lookup

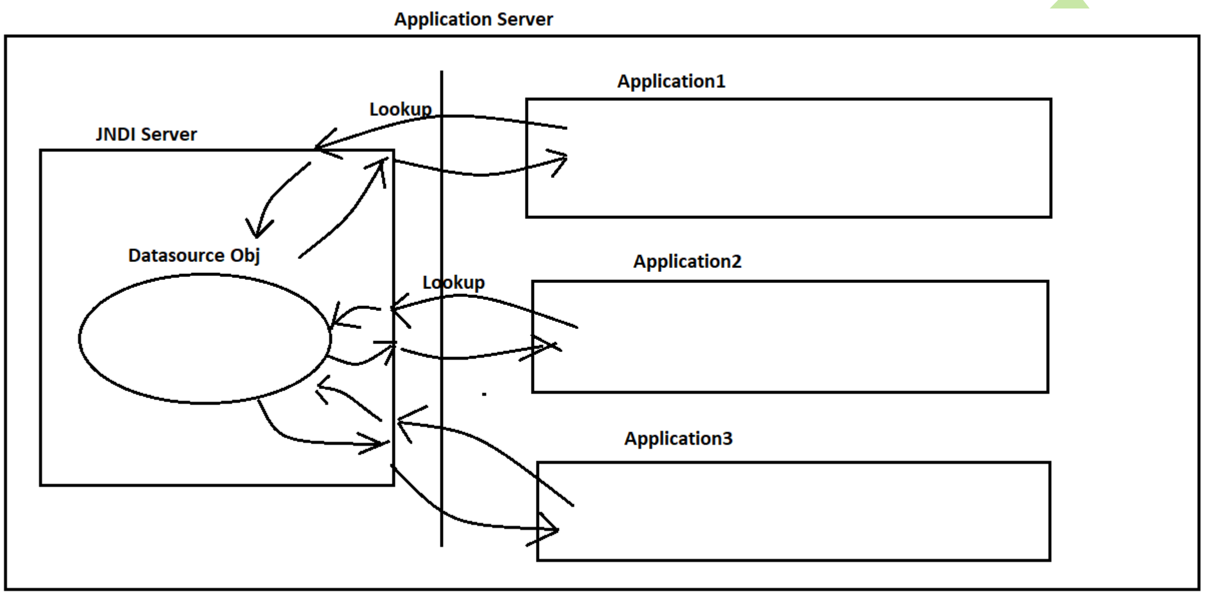
In Dependency Lookup, Service Provider will create the services and mainatined that services either in the regisry Softwares or in the containers , where we have to perform lookup operations inorder to get the required services.

There are two ways to get achieve Dependency lookup.
1. Dependency Pull
2. Contextualized Dependency Lookup

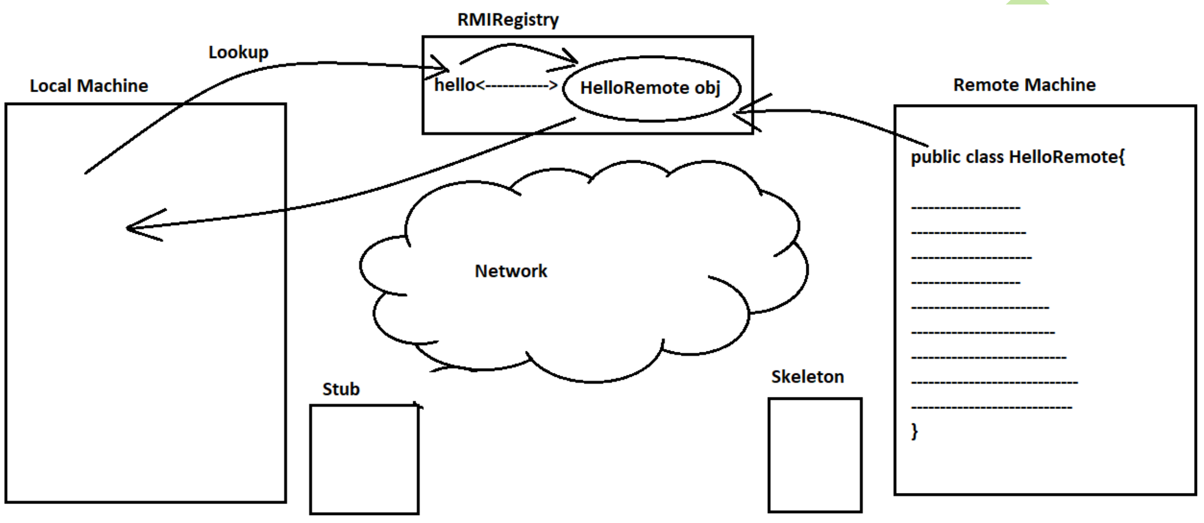
1. Dependecy Pull:

In Dependency Pull, Service Provider will create services and it will keep that services in any registry softwares, where we have to perform lookup operation inorde to get the required services, that is, pull the required services from registry softwares.

EX1: When we start application Servers like Weblogic, JBOSS , Websphere and Glassfish , automatically, Application Servers will create Datasource object and Application Servers will keep that Datasource object in JNDI Server as per the server settings which we made in application Server. In this context, we have to perform lookup(--) operation over JNDI server on the basis of the logical name inorder to get the required Datasource object.



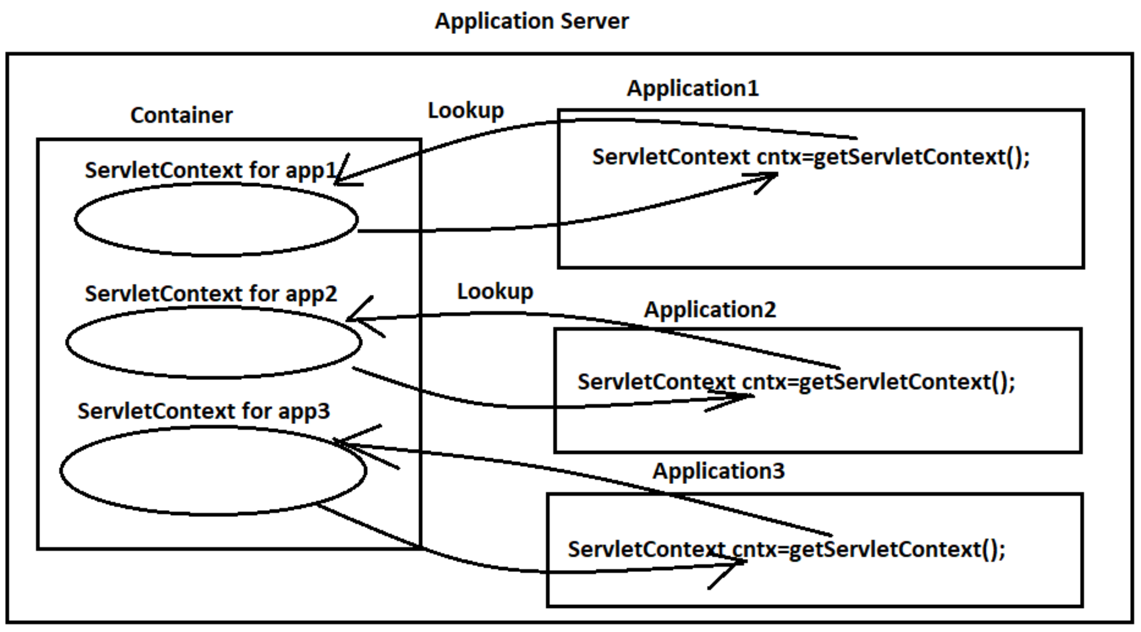
EX2:IN RMI Applications, User defined Registry Application will create Remote object and it will keep that Remote object in RMIRegistry with a particular logical name , here to get the required Remote object in Client Application we have to perform lookup operation over RMIRegistry on the basis of the logical name.



2. Contextualized Dependency Lookup

IN this mechanism, Service Provider will create the services and manage services , in this context, we have to perform lookup operation inorder to get the required services from Service Provider.

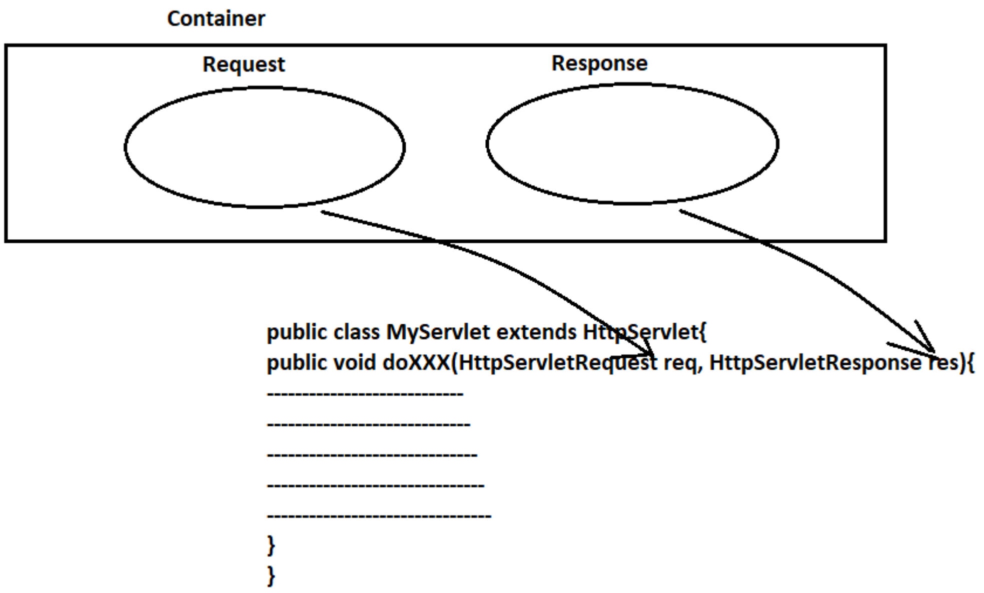
EX: In general, in web applications, when we start application Server then container will recognize all the web applications and container will deploy all the web applications into the server, when web applications are deployed in server , container will create ServletContext object automatically and Container will manage ServletContext object , in this context , to get ServletContext object we have to use getServletContext() method from GenericServlet directly with out using any reference variable, from ServletConfig object reference and from ServletRequest object reference, that is, performing lookup operation in Container to get ServletContext object.



2. Dependency Injection

IN this mechanism, Service Provider will create the services and inject the required services to the application directly with out performing lookup operations and with out getting request from client.
EX:In web applications, when we submit request to a particular Servlet , Container will execute the requested Servlet by performing the Servlet life cycle actions like Servlet Loading, Servlet Instantiation, Servlet Initialization , Request Processing and Servlet Deinstantiation. In Servlet initialization phase, container will create ServletConfig object and Container will inject ServletConfig object to the Servlet program through init(--) method, in Request Processing phase, Container has to create request and response objects and Container has to inject request and response objects to the Servlet program through service() method.

```
public class MyServlet extends HttpServlet{
ServletConfig config;
public void init(ServletConfig config)throws ServletException{
this.config=config;
}
public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException{
----
}
}
```



```
public class MyServlet extends HttpServlet{
public void doXXX(HttpServletRequest req, HttpServletResponse res){
----------------------------
----------------------------
----------------------------
----------------------------
----------------------------
}
}
```

There are two ways to achieve dependency injection.
1.Constructor Dependency Injection
2.Setter Method Dependency Injection

1. Constructor Dependency Injection

If we inject dependent values to the Bean object through Constructor then this type of Dependency Injection is called as "Constructor Dependency Injection".

If we want to use constructor dependency injection in SPring applications , first we have to declare the respective parameterized constructor in the corresponmding bean class then we have to declare that constructor arguments in spring configuration file by using the following tags.

```
<beans>

 <bean .....  >
   <constructor-arg value="--"/>
   <constructor-arg> value </constructor-arg>

 </bean>
 ----
</beans>
```

EX:

Course.java

```
package com.durgasoft.beans;

public class Course {
	private String cid;
	private String cname;
	private int ccost;

	public Course(String cid, String cname, int ccost){
		this.cid=cid;
		this.cname=cname;
		this.ccost=ccost;
	}

	public void getCourseDetails(){
		System.out.println("Course Details");
		System.out.println("-------------------");
		System.out.println("Course Id     :"+cid);
		System.out.println("Course Name  :"+cname);
		System.out.println("Course Cost  :"+ccost);
```

```
        }

}
```

applicationContext.xml

```
<beans>
        <bean id="crs" class="com.durgasoft.beans.Course">
                <constructor-arg value="C-111"/>
                <constructor-arg value="JAVA"/>
                <constructor-arg value="5000"/>
        </bean>
</beans>
```

Test.java

```
package com.durgasoft.test;

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

import com.durgasoft.beans.Course;

public class Test {
public static void main(String[] args)throws Exception {
ApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
Course course=(Course)context.getBean("crs");
course.getCourseDetails();
}
}
```

2. Setter Method Dependency Injection

If we inject dependent values to the Bean through setXXX() methods then it is called as "Setter Method Dependency Injection".

To inject primive values and String value to the bean object then we have to use "value" attributes in <property> or <constructor-arg> tags in beans configuration file, but, If we want to inject User defined data types , that is, Object reference values then we have to use "ref" attribute in <property> tag or in <constructor-arg> tag.

EX:

Account.java

```
package com.durgasoft.beans;

public class Account {
        private String accNo;
        private String accName;
        private String accType;
        private long balance;

      setXXX()
      getXXX()
}
```

Employee.java

```
package com.durgasoft.beans;

public class Employee {
        private String eid;
        private String ename;
```

```
        private float esal;
        private String eaddr;
        private Account acc;

      setXXX()
      getXXX()

        public void getEmployeeDetails(){

                System.out.println("Employee Details");
                System.out.println("-------------------");
                System.out.println("Employee Id      :"+eid);
                System.out.println("Employee Name    :"+ename);
                System.out.println("Employee Salary :"+esal);
                System.out.println("Employee Address:"+eaddr);
                System.out.println("Account Details");
                System.out.println("-------------------");
                System.out.println("Account Number :"+acc.getAccNo());
                System.out.println("Account Name    :"+acc.getAccName());
                System.out.println("Account Type    :"+acc.getAccType());
                System.out.println("Account Balance:"+acc.getBalance());
        }
}
```

applicationContext.xml

```
<beans>
        <bean id="acc" class="com.durgasoft.beans.Account">
                <property name="accNo" value="abc123"/>
                <property name="accName" value="Durga"/>
                <property name="accType" value="Savings"/>
                <property name="balance" value="25000"/>
        </bean>
        <bean id="emp" class="com.durgasoft.beans.Employee">
```

```
			<property name="eid" value="E-111" />
			<property name="ename" value="Durga"/>
			<property name="esal" value="50000"/>
			<property name="eaddr" value="Hyd"/>
			<property name="acc" ref="acc"/>
		</bean>
</beans>
```

Test.java

```
package com.durgasoft.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.durgasoft.beans.Employee;
public class Test {
public static void main(String[] args)throws Exception {
ApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
Employee emp=(Employee)context.getBean("emp");
emp.getEmployeeDetails();
}
}
```

Different Types of Elements Injection:

In Spring applications, if we want to inject User defined data types then we have to use either "ref" attribute in <property> and <constructor-arg> tags or we have to use <ref> nested tag under <property> and <constructor-arg> tags
EX:

```
<beans>
  <bean id="--" class="--">
    <property name="--" ref="--"/>
      <ref bean="--"/>
    </property>
```

```
 </bean>

</beans>
```

In spring applications, if we want to inject List of elements in beans then we have to declare the corresponding property as java.util.List and we have to provide values in configuration file by using <list> tag in <property> tag or in <constructor-arg> tag.
EX:

```
<beans>
 <bean id="---" class="--">
   <property name="--">
     <list>
        <value>value1</value>
        <value>value2</value>
         ----
         ----
     </list>
   </property>
 </bean>
</beans>
```

In Spring applications, if we want to inject Set of elements in Bean object then we have to declare the corresponding property as java.util.Set and we have to provide values in configuration file by using <set> tag under <property> tag or <constructor-arg> tag.
EX:

```
<beans>
 <bean id="---" class="--">
   <property name="--">
     <set>
        <value>value1</value>
        <value>value2</value>
```

```
      ----
      ----
    </set>
  </property>
 </bean>
</beans>
```

In Spring applications, if we want to inject Map of elements in Bean object then we have to declare the corresponding property as java.util.Map and we have to provide Key-Value pairs in configuration file by using <map> and <entry> tags under <property> tag or <constructor-arg> tag.

EX:

```
<beans>
  <bean id="--" class="--">
    <property name="--">
      <map>
        <entry key="key1" value="value1"/>
        <entry key="key2" value="value2"/>
        ----
      </map>
    </property>
  </bean>
</beans>
```

In Spring applications, if we want to inject Properties of elements in Bean object then we have to declare the corresponding property as java.util.Properties and we have to provide Key-Value pairs in configuration file by using <props> and <prop> tags under <property> tag or <constructor-arg> tag.

EX:

```
<beans>
  <bean id="--" class="--">
    <property name="--">
    <props>
      <prop key="key1"> value1</prop>
      <prop key="key2"> value2</prop>
      ----
    </props>
  </property>
 </bean>
</beans>
```

Example:

Student.java

```
package com.durgasoft.beans;

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

public class Student {
        private String sid;
        private String sname;
        private Address saddr;
        private List<String> squal;
        private Set<String> scourses;
        private Map<String, String> scourses_And_Faculty;
        private Properties scourse_And_Cost;
          setXXX()
        getXXX()

public void getStudentDeails(){
System.out.println("Student Details");
System.out.println("------------------");
System.out.println("Student Id      :"+sid);
System.out.println("Student Name     :"+sname);
System.out.println("Student Address  :"+saddr);
System.out.println("Student Qualifications :"+squal);
System.out.println("Student Courses  :"+scourses);
System.out.println("Student Courses And Faculty :"+scourses_And_Faculty);
System.out.println("Student Courses And Cost    :"+scourse_And_Cost);
}
}
```

Address.java

```
package com.durgasoft.beans;
public class Address {
        private String pno;
        private String street;
        private String city;
        private String country;

      setXXX()
      getXXX()

        public String toString(){
                return pno+","+street+","+city+","+country;
        }
}
```

applicationContext.xml

```
<beans>
        <bean id="addr" class="com.durgasoft.beans.Address">
                <property name="pno" value="202"/>
                <property name="street" value="M G Road"/>
                <property name="city" value="Banglore"/>
                <property name="country" value="India"/>
        </bean>
        <bean id="std" class="com.durgasoft.beans.Student">
                <property name="sid" value="S-111"/>
                <property name="sname" value="Durga"/>
                <property name="saddr">
                        <ref bean="addr"/>
                </property>
                <property name="squal">
                        <list>
                                <value>BTech</value>
                                <value>MTech</value>
                                <value>PHD</value>
                        </list>
                </property>
                <property name="scourses">
                        <set>
                                <value>Core Java</value>
                                <value>Adv Java</value>
                                <value>Spring</value>
                                <value>Hibernate</value>
                                <value>WebServices</value>
                        </set>
                </property>
                <property name="scourses_And_Faculty">
                        <map>
                                <entry key="Core Java" value="Ratan"/>
                                <entry key="Adv Java" value="Durga"/>
                                <entry key="Spring" value="Sriman"/>
                                <entry key="Hibernate" value="Naveen"/>
                                <entry key="Webservices" value="Naidu"/>
                        </map>
                </property>
                <property name="scourse_And_Cost">
```

```
                        <props>
                                <prop key="Core Java">1500</prop>
                                <prop key="Adv Java">2000</prop>
                                <prop key="Spring">3000</prop>
                                <prop key="Hibernate">3000</prop>
                                <prop key="Webservices">3000</prop>
                        </props>
                </property>
        </bean>
</beans>
```

Test.java

```
package com.durgasoft.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.durgasoft.beans.Student;
public class Test {
public static void main(String[] args)throws Exception {
ApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
Student std=(Student)context.getBean("std");
std.getStudentDeails();
}
}
```

Circular Dependency Injection:

In Spring applications, if more than one bean objects are depending on each other through constructor dependency injection then it is called as Circular Dependency Injection, which is not supported by Spring famework, it able to rise an exception like " org.springframework.beans.factory.BeanCurrentlyInCreationException"

Ex:

Student.java

```
package com.durgasoft.beans;

public class Student {
	Branch branch;
	public Student(Branch branch) {
		this.branch=branch;
	}
	public String getStudentName(){
		return "Durga";
	}
}
```

Branch.java

```
package com.durgasoft.beans;

public class Branch {
	Student student;
	public Branch(Student student) {
		this.student=student;
	}
	public String getBranchName(){
		return "S R Nagar";
	}
}
```

applicationContext.xml

```
<beans>
	<bean id="student" class="com.durgasoft.beans.Student">
```

```
                <constructor-arg ref="branch"/>
        </bean>
        <bean id="branch" class="com.durgasoft.beans.Branch">
                <constructor-arg ref="student"/>
        </bean>
</beans>
```

Test.java

```
package com.durgasoft.test;

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

import com.durgasoft.beans.Branch;
import com.durgasoft.beans.Student;

public class Test {
public static void main(String[] args)throws Exception {
ApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
Student std=(Student)context.getBean("student");
System.out.println(std.getStudentName());
Branch branch=(Branch)context.getBean("branch");
System.out.println(branch.getBranchName());
}
}
```

In Spring applications, if we want to resolve Circular Dependency Injection then we have to use Setter Method dependency Injection instead of Constructor Dependency Injection.

Student.java

```
package com.durgasoft.beans;

public class Student {
        Branch branch;
        setXXX()
      getXXX()
}
```

Branch.java

```
package com.durgasoft.beans;

public class Branch {
        Student student;
        setXXX()
      getXXX()
}
```

applicationContext.xml

```
<beans>
        <bean id="student" class="com.durgasoft.beans.Student">
                <property name="branch" ref="branch"/>
        </bean>
        <bean id="branch" class="com.durgasoft.beans.Branch">
                <property name="student" ref="student"/>
        </bean>
</beans>
```

Q) In Spring applications, if we provide both Setter method dependency injection and Constructor dependency injection to a single bean then what will happen in Spring Application?

Ans:

If we provide both Constructor dependency injection and Setter method dependency injection to a single bean then IOC Container will perform constructor dependency injection first at the time of creating Bean object , after that, IOC Container will perform Setter method dependency injection, that is, Constructor Dependency Injection provided values are overridden with setter method dependency injection provided values, finally, Bean object is able to manage Setter method dependency Injection provided values.

Example:

Student.java

```
package com.durgasoft.beans;

public class Student {
        private String sid;
        private String sname;
        private String saddr;

        public Student(String sid, String sname, String saddr){
                this.sid=sid;
                this.sname=sname;
                this.saddr=saddr;
                System.out.println("Student(---)-Constructor");
        }

        setXXX()
      getXXX()
```

```
	public void getStudentDetails(){
		System.out.println("Student Details");
		System.out.println("-------------------");
		System.out.println("Student Id      :"+sid);
		System.out.println("Student Name    :"+sname);
		System.out.println("Student Address :"+saddr);
	}

}
```

spring_beans_config.xml

```
<beans>
	<bean id="std" class="com.durgasoft.beans.Student">
		<constructor-arg index="0" value="S-111"/>
		<constructor-arg index="1" value="AAA"/>
		<constructor-arg index="2" value="Hyd"/>

		<property name="sid" value="S-222"/>
		<property name="sname" value="BBB"/>
		<property name="saddr" value="Sec"/>
	</bean>
</beans>
```

Test.java

```
package com.durgasoft.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.durgasoft.beans.Student;
public class Test {
public static void main(String[] args) {
ApplicationContext context=new
ClassPathXmlApplicationContext("/com/durgasoft/cfgs/spring_beans_config.xml");
```

```
Student std=(Student)context.getBean("std");
std.getStudentDetails();
}
}
```

Q) What are the differences between Constructor Dependency Injection and Setter Method Dependency Injection?

Ans:

1. In Constructor dependency injection, dependent values injected through a particular constructor.

In Setter method dependency injection, dependent values are injected through properties respective setXXX() methods.

2.In Constructor Dependency Injection readability is not good , because, in Constructor dependency injection we are unable to identify to which property we are injecting dependent values.

In setter method Dependency injection Readability is very good, because, in Setter method Dependency injection we are able to identify that to property we are able to inject the dependent values.

3.In Constructor Dependency injection , dependency injection is possible when all dependent objects are getting ready, if dependent objects are not ready then Constructor dependency injection is not possible.

In Setter method dependency injection, even though dependent values are not ready, Setter method dependency injection will be performed.

4.In case of constructor dependency injection ,partial dependency injection is not possible, because, we have to access the constructor by passing the required no of parameter values.

In case of setter method dependency injection, partial dependency injection is possible , because, we are able to access setXXX() method individually.

5.IN case of constructor dependency injection, it is not simple to change the values in bean object.

In case of Setter method dependency injection , it is very simple to change the values in bean object.

6. In Constructor dependency injection, for every change on values a new bean object is created, because, for every change we have to call constructor explicitly.

In Setter method dependency injection, for every change on values new object is not created, because, for every change we can access setXXX() method explicitly.

7. Constructor dependency injection will make the bean object as "Immutable Object".

Setter method dependency injection will make the bean object as "mutable Object".

8.If we provide both Constructor and setter method dependency injection to a single bean object then setter method dependency injection overrides constructor dependency injection, but, constructor dependency injection is not overriding setter cmethod dependency injection.

9.Constructor dependency injection may provide circular dependency injection.

Setter method dependency injection will not provide circular dependency injection.

10.Constuctor dependency injection will give guarantee for dependency injection.

Setter method dependency injection will not give guarantee for dependency injection.

11.In Spring applications, if we have more no of elements to inject then it is suggestible to use Constructor dependency injection instead of setter method dependency injection.

P-Namespace and C-Namespace :

P-Namespace:

IN general, in setter method dependency injection, to specify dependent values in spring configuration file we have to use <property> tags as per the no of properties in the bean class. In this context, to remove <property> tags and to provide dependent values as atributes in <bean> tag in spring configuration file we have to use "P-Namespace".

Note: To use p-namespace in spring configration file we have to define "p" namespace in XSD like below.

```
xmlns:p="http://www.springframework.org/schema/p"
```

To provide value as attribute by using "p" namespace in <bean> tag we have to use the following syntax.

```
<bean id="--" class="--" p:prop_Name="value" p:prop_Name="value".../>
```

If we want to specify object referernce variable as dependent value the we have to use "-ref" along with property.

```
<bean id="--" class="--" p:prop_Name-ref="ref"/>
```

C-Namespace:

In general, in constgructor dependency injection, to specify dependent values in spring configuration file we have to use <copnstructor-arg> tags as per the no of parameters which we defined in the bean class constructor . In this context, to remove <constructor-arg> tags and to provide dependent values as attributes in <bean> tag in spring configuration file we have to use "C-Namespace".

Note: To use c-namespace in spring configration file we have to define "c" namespace in XSD like below.

xmlns:c="http://www.springframework.org/schema/c"

To provide value as attribute by using "c" namespace in <bean> tag we have to use the following syntax.

<bean id="--" class="--" c:arg_Name="value" c:arg_Name="value".../>

If we want to specify object referernce variable as dependent value then we have to use "-ref" along with argument_Name.

<bean id="--" class="--" c:arg_Name-ref="ref"/>

If we want to specify dependent values in beans configuration file on the basis of index values then we have to use xml code like below.

<bean id="--" class="--" c:_0="val1" c:_1="val2"...c:_4-ref="ref"/>

Example:

Employee.java

```
package com.durgasoft.beans;

public class Employee {
	private String eid;
	private String ename;
	private float esal;
	private Address eaddr;

	setXXX()
```

```
        getXXX()

        public void getEmpDetails(){
                System.out.println("Employee Details");
                System.out.println("--------------------");
                System.out.println("Employee Id     :"+eid);
                System.out.println("Employee Name   :"+ename);
                System.out.println("Employee Salary :"+esal);
                System.out.println();
                System.out.println("Employee Address Details");
                System.out.println("--------------------------");
                System.out.println("House Number   :"+eaddr.getHno());
                System.out.println("Street         :"+eaddr.getStreet());
                System.out.println("City           :"+eaddr.getCity());
                System.out.println("State          :"+eaddr.getState());
        }
}
```

Address.java

```
package com.durgasoft.beans;

public class Address {
        String hno;
        String street;
        String city;
        String state;

        setXXX()
        getXXX()

}
```

Student.java

```
package com.durgasoft.beans;

public class Student {
	String sid;
	String sname;
	String saddr;
	Course crs;
	public Student(String sid, String sname, String saddr, Course		crs) {
		this.sid=sid;
		this.sname=sname;
		this.saddr=saddr;
		this.crs=crs;
	}

	public void getStudentDetails(){
		System.out.println("Student Details");
		System.out.println("------------------");
		System.out.println("Student Id	:"+sid);
		System.out.println("Student Name	:"+sname);
		System.out.println("Student Address :"+saddr);
		System.out.println();
		crs.getCourseDetails();
	}
}
```

Course.java

```
package com.durgasoft.beans;

public class Course {
	String cid;
	String cname;
	int ccost;
```

```
        public Course(String cid, String cname, int ccost) {
                this.cid=cid;
                this.cname=cname;
                this.ccost=ccost;
        }
        public void getCourseDetails(){
                System.out.println("Course Details");
                System.out.println("--------------------");
                System.out.println("Course Id   :"+cid);
                System.out.println("Course Name :"+cname);
                System.out.println("Course Cost :"+ccost);
        }
}
```

spring_beans_config.xml

```
<beans -------
   xmlns:p="http://www.springframework.org/schema/p"
   xmlns:c="http://www.springframework.org/schema/c"
    -----
   >
        <bean id="emp" class="com.durgasoft.beans.Employee"
        p:eid="E-111" p:ename="AAA" p:esal="15000" p:eaddr-ref="addr"/>
        <bean name="addr" class="com.durgasoft.beans.Address"
        p:hno="23/3rt" p:street="M G Road" p:city="Hyd" p:state="Tel"/>

        <bean id="std" class="com.durgasoft.beans.Student" c:sid="S-111"
        c:sname="AAA" c:saddr="Hyd" c:crs-ref="crs"/>
        <bean id="crs" class="com.durgasoft.beans.Course" c:_0="C-111"
        c:_1="JAVA" c:_2="10000"/>
</beans>
```

Test.java

```
package com.durgasoft.test;

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

import com.durgasoft.beans.Employee;
import com.durgasoft.beans.Student;

public class Test {
public static void main(String[] args) {
ApplicationContext context=new
ClassPathXmlApplicationContext("/com/durgasoft/cfgs/spring_beans_config.xml");
Employee emp=(Employee)context.getBean("emp");
emp.getEmpDetails();
System.out.println();
Student std=(Student)context.getBean("std");
std.getStudentDetails();
}
}
```