**=> Inversion Of Control (IoC) :-**

-> Inversion of Control is a design pattern/principal that focus on inverting the conrol flow of an application

-> It shifts the responsibility of managing the flow of execution and the lifecycle of objects from application itself to external entity i.e. framework or container

-> It identifies the client required dependencies or services and then it will create and inject the required dependencies or service to the application without client request

-> Spring Container works on the basis of IoC principal and thus it is also known as IoC Container

-> Advantages of IoC principal :-

1. Classes are loosly coupled

2. Modularity can be achieved

3. Easier to test and maintain the application

etc

-> In spring IoC principal can be achieved by following :-

1. Dependency Injection (DI)

2. Service Locator

3. Contextualized Lookup

4. Template Method Design Pattern

5. Event Based IoC

etc

-> NOTE : From above, only DI is most commonly IoC principal used in spring

=**> Dependency Injection (DI) :-**

-> Dependency Injection is a design pattern that is used to implement IoC principal

-> Dependency Injection main functionality is to "inject" one object into another object

-> How to achieve DI in Spring Configuration File (xml) :-

= We can achieve DI by 2 ways :-

1. Setter Method DI

2. Constructor DI

================================================================================================

**=> What is difference between Setter Method DI and Constructor DI :-**

1. How dependency is injected :-

= Setter Method DI uses setter methods i.e. setXXX() method to inject the dependency

= Constructor DI uses constructor to inject the dependency

2. Readability :-

= Setter Method DI has more readability because we have to provide the property name and its value

= Constructor DI has less readability because we dont provide the property name with value

3. Partial Dependency :-

= Partial Dependency is possible in case of Setter Method DI

= Partial Dependency is not possible in case of Constructor DI

4. Circular DI :-

= We can achieve Circular DI using Setter Method DI

= We cannot achieve Circular DI using Constructor DI

Program:- (WE ARE ACHIVING THIS TO XML FILE WITH SETTER METHOD DI)

Beans folder

**Student class:-**

package in.sp.beans;

public class Student

{

    private Teacher teacher;

    public Teacher getTeacher() {

        return teacher;

    }

    public void setTeacher(Teacher teacher) {

        this.teacher = teacher;

    }

    public void studentDisplay()

    {

        System.out.println("im in student class : "+teacher);

    }

}

**Address class:-**

package in.sp.beans;

public class Address

{

    private int houseno;

    private String city;

    private int pincode;

    public int getHouseno() {

        return houseno;

    }

    public void setHouseno(int houseno) {

        this.houseno = houseno;

    }

    public String getCity() {

        return city;

    }

    public void setCity(String city) {

        this.city = city;

    }

    public int getPincode() {

        return pincode;

    }

    public void setPincode(int pincode) {

        this.pincode = pincode;

    }

    @Override

    public String toString()

    {

        return "#"+houseno+", "+city+" - "+pincode;

    }

}

**Resource file:**

**ApplicationContext.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"

    xsi:schemaLocation="

        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean class="in.sp.beans.Address" id="addrId">

        <property name="houseno" value="123" />

        <property name="city" value="Delhi" />

        <property name="pincode" value="12345" />

    </bean>

    <bean class="in.sp.beans.Student" id="stdId">

        <property name="name" value="Deepak" />

        <property name="rollno" value="111" />

        <property name="address" ref="addrId" /> //We need to use ref here to provide ref of a obj

    </bean>

</beans>

**Main class:-**

package in.sp.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import in.sp.beans.Student;

public class Main

{

    public static void main(String[] args)

    {

        ApplicationContext context = new ClassPathXmlApplicationContext("/in/sp/resources/applicationContext.xml");

        Student std = (Student) context.getBean("stdId");

        std.display();

    }

}

**ACHIVING DI WITH CONSTRUCTOR:-**

**ApplicationContext.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"

    xsi:schemaLocation="

        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean class="in.sp.beans.Address" id="addrId">

        <constructor-arg value="123" />

        <constructor-arg value="Mumbai" />

        <constructor-arg value="54321" />

    </bean>

    <bean class="in.sp.beans.Student" id="stdId">

        <constructor-arg value="Rahul" />

        <constructor-arg value="222" />

        <constructor-arg ref="addrId" />

    </bean>

</beans>

**Main.class**

package in.sp.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import in.sp.beans.Student;

public class Main

{

    public static void main(String[] args)

    {

        ApplicationContext context = new ClassPathXmlApplicationContext("/in/sp/resources/applicationContext.xml");

        Student std = (Student) context.getBean("stdId");

        std.display();

    }

}

**Circular Dependency:-**

Suppose we have 2 bean classes Student and teacher And both classes need each other that’s why we to pass ref of each obj to another

**ApplicationContext.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"

    xsi:schemaLocation="

        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean class="in.sp.beans.Student" id="stdId">

        <property name="teacher" ref="teachId" />

    </bean>

    <bean class="in.sp.beans.Teacher" id="teachId">

        <property name="student" ref="stdId" />

    </bean>

</beans>

From the above exam Circular dependency is possible is Setter based dependency Injection with xml based configuration because in setter method obj is already created and initialize properties with default value, then setter method is called. But in constructor based DI it is not possible because obj is not made at first.

**Constructor based Xml Config:-**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="

        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean class="in.sp.beans.Student" id="stdId">

        <constructor-arg ref="teachId" />

    </bean>

    <bean class="in.sp.beans.Teacher" id="teachId">

        <constructor-arg ref="stdId" />

    </bean>

</beans>

**NOT POSSIBLE**