**What is Dependency ?**

What if the Air on this earth become Vanished?

Yes, you are right. We will unable take a breath and will die in few minutes.

So here we can say we are totally depend upon Air. In this case, Air is dependency for us.

**What is Injection ?**

Suppose a person is suffering from taking breath in and out. So there is a mechanical device called **Ventilator** which helps the person to in and out the air to his lungs. And now person able to take breath easily.

Here, Ventilator is working as a Injection because it is injecting air into our lungs.

**Conclusion :**

A breathing sick person is depend upon air so Air is dependency for the person and ventilator is injecting this dependency inside the person. Hence, ventilator is working as a Dependency Injection here.

**Spring has its own Ventilator called as IOC Container.**

**What is Dependency Injection In The World Of Spring ?**

Suppose we have class called **Person** with the following body :

public class Person {

private double money;

private Air air;

private ArrayList<String> friends;

}

The Person class is depending upon money, air and friends so they all are dependency for the class Person and it can have many more dependency as well.

So, Who is going to provide these dependency to the class Person ?

The answer is **Spring.**

Although we can directly provide all these dependency by assigning some value in it but it will become Hard Coding which is not good programming practice.

**Types Of Dependency ?**

1. Dependencies in form of **literals**

The dependencies whose data type comes in the following list are comes in this category.

* All data types
* String

Ex : private double money; of Person class.

1. Dependencies in form of **objects**

If any dependency required object of our self made class will come in this category.

Ex: private Air air; of person class.

1. Dependencies in form of **collection**

If any dependency required object of any collection framework class will come in this category.

Ex : private ArrayList<String> friends; of person class.

**How Spring Dependency Injection Performs ?**

Spring provide two injection mechanisms :

* **Setter Injection**
* **Constructor Injection**

**Setter Injection Vs Constructor Injection**

As per the principle of Encapsulation, we should always make our data members / dependencies private and give their access using either constructor or some special methods better know as Setter/Getter methods.

Look at the below program :

**Bean Class :**

public class Person {

private double money;

public Person() {

}

public Person(double money) {

super();

this.money = money;

}

public double getMoney() {

return money;

}

public void setMoney(double money) {

this.money = money;

}

}

**Injecting dependency using setter injection :**

public class Earth {

public static void main(String []args) {

Person person = new Person();

person.setMoney(5000);

System.out.println(person.getMoney());

}

}

**Injecting dependency using constructor injection :**

public class Earth {

public static void main(String []args) {

Person person = new Person(5000);

System.out.println(person.getMoney());

}

}

}

In this way, we can inject dependency into a class. But here we are creating Person class object on our own.

Spring says, do not create class object on your own and even do not write any code to inject dependency I can do that for you.

Just let me know through config.xml which class object you want to be create and what dependency you want to inject in it.

Let’s start Literal dependency injection,

**Literal Injecting Using Setter Injection ?**

**Bean Class :**

public class Person {

private double money;

public double getMoney() {

return money;

}

public void setMoney(double money) {

this.money = money;

}

}

**config.xml :**

<bean id="person" class="com.spring.literal.constructorinjection.Person" >

<property name="money" value="5000"/>

</bean>

**What is <property> Tag ?**

The property tag use for setter injection. It takes two attribute :

**name :** it is the name of dependency.

private double money;

**value :** it is the value to be injected in dependency.

**Main Class :**

public class Earth {

public static void main(String []args) {

String p = "com/spring/literal/setterinjection/config.xml";

ApplicationContext context = new ClassPathXmlApplicationContext(p);

Person p = context.getBean("person", Person.class);

System.out.println(p.getMoney());

}

}

**Literal Injecting Using Constructor Injection ?**

**Bean Class :**

public class Person {

private double money;

public Person(double money) {

this.money = money;

}

public void show() {

System.out.println(this.money);

}

}

**config.xml :**

<bean id="person" class="com.spring.literal.constructorinjection.Person" >

<constructor-arg value="5000" type="double" />

</bean>

**What is** <constructor-arg> **Tag ?**

The property tag use for setter injection. It takes two attribute :

**name :** it is the name of dependency.

private double money;

**value :** it is the value to be injected in dependency.

**type :** it is an optional attribute here, but it is help to resolve ambiguity. The **value** attribute only takes string argument and it may be possible there is parameter in constructor with different type. Although spring does this conversion implicitly in most cases but it is good practice to specify in what type you want to convert the value of **value** attribute.

**Main Class :**

public class Earth {

public static void main(String []args) {

String p = "com/spring/literal/constructorinjection/config.xml";

ApplicationContext context = new ClassPathXmlApplicationContext(p);

Person p = context.getBean("person", Person.class);

p.show();

}

}

**Object Injecting Using Setter Injection ?**

As we discussed a scenario before, a Human is depend upon Air. In this case, we have two classes called Human and Air.

We always try to make our code loosely coupled as much as possible. Hence, we always ignore to make use the keyword new.

Let’s take it practically, how spring will help us to inject Object dependencies with loose coupling.

**Air.java**

public class Air {

public String toString() {

return "Air blowing";

}

}

**Person.java**

public class Person {

private Air air;

public void setAir(Air air) {

this.air = air;

}

public void breath() {

System.out.println("Person is Breathing - "+air);

}

}

**config.xml**

<bean id="person" class="com.spring.object.setterinjection.Person" >

<property name="air">

<bean class="com.spring.object.setterinjection.Air" />

</property>

</bean>

**Main Class**

public class Earth {

public static void main(String []args) {

String path = "com/spring/object/setterinjection/config.xml";

ApplicationContext context = new ClassPathXmlApplicationContext();

Person p = context.getBean("person", Person.class);

p.breath();

}

}

In this exercise, you learnt how we can injecting Air dependency into Person. But practically, in this world there is lot many Person and everyone need of Air.

Now suppose we have two person class **– Person, AnotherPerson**

Both of them having Air object dependency.

In this case, our **config.xml** will look like :

<bean id="person" class="com.spring.object.setterinjection.Person" >

<property name="air">

<bean class="com.spring.object.setterinjection.Air" />

</property>

</bean>

<bean id="person2" class="com.spring.object.setterinjection.AnotherPerson" >

<property name="air">

<bean class="com.spring.object.setterinjection.Air" />

</property>

</bean>

So the point is we are asking for different Air object for every Person. When spring will look the highlighted part it will create object. So two object of Air class will be created.

But we should always try to make our application **Light Weighted.** We should avoid making of unwanted objects.

How we can overcome this problem ?

Just mention a <bean> tag as a sibling of both the <bean> tag instead of mentioning child to it.

<bean id="air" class="com.spring.object.setterinjection.Air" />

<bean id="person" class="com.spring.object.setterinjection.Person" >

<property name="air" ref="air" />

</bean>

<bean id="person2" class="com.spring.object.setterinjection.AnotherPerson" >

<property name="air" ref="air" />

</bean>

Now spring will create only on Air object.

The ref attribute is used to refer an object. Whatever value we assign to it. It search the object having id same as the ref value and assign it into the given dependency name.

**Object Injecting Using Construtor Injection ?**

**Collection Injecting Using Setter Injection ?**

**Collection Injecting Using Constructor Injection ?**