**What is IoC Container?**

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets information’s from the XML file and works accordingly. The main tasks performed by IoC container are:

* to instantiate the application class
* to configure the object
* to assemble the dependencies between the objects

There are two types of IoC containers. They are:

1. **BeanFactory**
2. **ApplicationContext**

Difference between BeanFactory and the ApplicationContext

The org.springframework.beans.factory.**BeanFactory** and the org.springframework.context.**ApplicationContext** interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory

Using BeanFactory

The XmlBeanFactory is the implementation class for the BeanFactory interface. To use the BeanFactory, we need to create the instance of XmlBeanFactory class as given below:

Resource resource=**new** ClassPathResource("applicationContext.xml");

BeanFactory factory=**new** XmlBeanFactory(resource);

Using ApplicationContext

The ClassPathXmlApplicationContext class is the implementation class of ApplicationContext interface. We need to instantiate the ClassPathXmlApplicationContext class to use the ApplicationContext as given below:

1. ApplicationContext context =     **new** ClassPathXmlApplicationContext("applicationContext.xml");

The constructor of ClassPathXmlApplicationContext class receives string, so we can pass the name of the xml file to create the instance of ApplicationContext.

**Dependency Injection**

The Dependency Injection is a design pattern that removes the dependency of the programs. In such case we provide the information from the external source such as XML file. It makes our code loosely coupled and easier for testing.

There are two way to inject dependency in spring framework.

1. Setter method

2. Constructor.

**1 Inject dependency using Setter method.**

<bean id=*"emp"* class=*"com.spring.Employee"*>

<!-- set the value using setter method -->

<property name="empid" value="101"></property>

<property name="fname" value="Radhey"></property>

<property name="lname" value="Krishan"></property>

<property name="mobile" value="89999999"></property>

<property name="email" value="radhakrishan@gmail.com"></property>

</bean>

**2.Inject dependency using Constructor.**

<bean id=*"emp"* class=*"com.spring.Employee"*>

<!-- Set the value using constructor -->

<constructor-arg value=*"101"*/>

<constructor-arg value=*"Radhey"*/>

<constructor-arg value=*"Krishan"*/>

<constructor-arg value=*"89999999"*/>

<constructor-arg value=*"radhakrishan@gmail.com"*/>

</bean>

**E**

**EmpEmployee Object**

// bean class enitity class pojo(plain java object) class

**public** **class** Employee {

// there are two we can set in those variable

// 1 setter getter method

// 2 constractor

**private** **int** empid;

**private** String fname;

**private** String lname;

**private** String mobile;

**private** String email;

// if you have create parameterized constructor then we should create defalut constructor bcoz JVM will not provide default Constructor

**public** Employee() {

// **TODO** Auto-generated constructor stub

}

**public** Employee(**int** empid, String fname, String lname, String mobile, String email) {

**super**();

**this**.empid = empid;

**this**.fname = fname;

**this**.lname = lname;

**this**.mobile = mobile;

**this**.email = email;

}

// create setter getter method to corresponding variable

}