Spring – Understanding Inversion of Control with Example

Last Updated : 18 Feb, 2022

Spring IoC (Inversion of Control) Container is the core of Spring Framework. It creates the objects, configures and assembles their dependencies, manages their entire life cycle. The Container uses Dependency Injection(DI) to manage the components that make up the application. It gets the information about the objects from a configuration file(XML) or Java Code or Java Annotations and Java POJO class. These objects are called Beans. Since the Controlling of Java objects and their lifecycle is not done by the developers, hence the name Inversion Of Control.

There are 2 types of IoC containers:

- BeanFactory
- ApplicationContext

That means if you want to use an IoC container in spring whether we need to use a BeanFactory or ApplicationContext. The BeanFactory is the most basic version of IoC containers, and the ApplicationContext extends the features of BeanFactory. The followings are some of the

- Managing our objects,
- Helping our application to be configurable,
- Managing dependencies

Implementation: So now let's understand what is IoC in Spring with an example. Suppose we have one interface named Sim and it has some abstract methods calling() and data().

Java

```
// Java Program to Illustrate Sim Interface
public interface Sim
{
    void calling();
    void data();
}
```

Now we have created another two classes Airtel and Jio which implement the Sim interface and override the interface methods.

Java

```
// Java Program to Illustrate Airtel Class

// Class
// Implementing Sim interface
public class Airtel implements Sim {

    @Override public void calling()
    {
        System.out.println("Airtel Calling");
    }

    @Override public void data()
```

Java

```
// Java Program to Illustrate Jio Class

// Class
// Implementing Sim interface
public class Jio implements Sim{
    @Override
    public void calling() {
        System.out.println("Jio Calling");
    }

    @Override
    public void data() {
        System.out.println("Jio Data");
    }
}
```

So let's now call these methods inside the main method. So by implementing the Run time polymorphism concept we can do something like this

Java

```
// Java Program to Illustrate Mobile Class

// Class
public class Mobile {

    // Main driver method
    public static void main(String[] args)
    {
```