Dependency Injection with Factory Method in Spring

Spring framework provides facility to inject bean using factory method. To do so, we can use two attributes of bean element.

1. **factory-method:** represents the factory method that will be invoked to inject the bean.
2. **factory-bean:** represents the reference of the bean by which factory method will be invoked. It is used if factory method is non-static.

A method that returns instance of a class is called **factory method**.

```
public class A {
public static A getA(){//factory method
   return new A();
}
}
```

Factory Method Types

There can be three types of factory method:

1) A **static factory method** that returns instance of **its own** class. It is used in singleton design pattern.

```
<bean id="a" class="com.javatpoint.A" factory-method="getA"></bean>
```

2) A **static factory method** that returns instance of **another** class. It is used instance is not known and decided at runtime.

```
<bean id="b" class="com.javatpoint.A" factory-method="getB"></bean>
```

3) A **non-static factory** method that returns instance of **another** class. It is used instance is not known and decided at runtime.

```
<bean id="a" class="com.javatpoint.A"></bean>
<bean id="b" class="com.javatpoint.A" factory-method="getB" factory-bean="a"></bean>
```

Type 1

Let's see the simple code to inject the dependency by static factory method.

```
<bean id="a" class="com.javatpoint.A" factory-method="getA"></bean>
```

Let's see the full example to inject dependency using factory method in spring. To create this example, we have created 3 files.

1. **A.java**
2. **applicationContext.xml**
3. **Test.java**

A.java

This class is a singleton class.

```
package com.javatpoint;
public class A {
private static final A obj=new A();
private A(){System.out.println("private constructor");}
public static A getA(){
   System.out.println("factory method ");
   return obj;
}
public void msg(){
   System.out.println("hello user");
}
}
```

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"
   xmlns:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="a" class="com.javatpoint.A" factory-method="getA"></bean>

</beans>
```

Test.java

This class gets the bean from the applicationContext.xml file and calls the msg method.

```
package org.sssit;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Test {
public static void main(String[] args) {
   ApplicationContext context=new ClassPathXmlApplicationContext("applicationContext.xml");
   A a=(A)context.getBean("a");
   a.msg();
}
}
```

Output:

```
private constructor
factory method
hello user
```

Type 2

Let's see the simple code to inject the dependency by static factory method that returns the instance of another class.

To create this example, we have created 6 files.

1. **Printable.java**
2. **A.java**
3. **B.java**
4. **PrintableFactory.java**
5. **applicationContext.xml**
6. **Test.java**

Printable.java

```
package com.javatpoint;
public interface Printable {
void print();
}
```

A.java

```
package com.javatpoint;
public class A implements Printable{
    @Override
    public void print() {
        System.out.println("hello a");
    }

}
```

B.java

```
package com.javatpoint;
public class B implements Printable{
    @Override
    public void print() {
        System.out.println("hello b");
    }
}
```

PrintableFactory.java

```
package com.javatpoint;
public class PrintableFactory {
public static Printable getPrintable(){
    //return new B();
          return new A();//return any one instance, either A or B
```

```
}
}
```

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"
   xmlns:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="p" class="com.javatpoint.PrintableFactory" factory-method="getPrintable">
</bean>

</beans>
```

Test.java

This class gets the bean from the applicationContext.xml file and calls the print() method.

```
package org.sssit;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Test {
public static void main(String[] args) {
   ApplicationContext context=new ClassPathXmlApplicationContext("applicationContext.xml");
   Printable p=(Printable)context.getBean("p");
   p.print();
}
}
```

Output:

```
hello a
```

Type 3

Let's see the example to inject the dependency by non-static factory method that returns the instance of another class.

To create this example, we have created 6 files.

1. **Printable.java**
2. **A.java**
3. **B.java**
4. **PrintableFactory.java**
5. **applicationContext.xml**
6. **Test.java**

All files are same as previous, you need to change only 2 files: PrintableFactory and applicationContext.xml.

PrintableFactory.java

```
package com.javatpoint;
public class PrintableFactory {
//non-static factory method
public Printable getPrintable(){
    return new A();//return any one instance, either A or B
}
}
```

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"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="pfactory" class="com.javatpoint.PrintableFactory"></bean>
<bean id="p" class="com.javatpoint.PrintableFactory" factory-method="getPrintable"
factory-bean="pfactory"></bean>

</beans>
```

Output:

```
hello a
```