# Spring XML Based DI and Builder Pattern

Recently, I had faced a situation where a project using Spring framework for its Business layer and maintains the Spring dependencies in the XML file. They can not upgrade this XML based configuration to java based configuration as this project is old it will take time and they can’t afford it.

That is fine but the problem is they want to add a third party module (another module developed by other set of developers of that project family) to re-use some utilities of that library.

But when they try to incorporate aforesaid module they discovered that many classes which are exposed as services were created using Builder pattern. So we will see here How we can incorporate Builder pattern in Spring XML-based DI.

**Builder Pattern :**

In a brief Builder pattern is used to create complex Object, which has some required and some optional parameters,

We create a static inner class called Builder and pass require parameters in the constructor of that builder. Then set Optional parameter calling the Setter method of that Builder.

Lastly, it has a **build()** method, which will produce the actual Object.

Say, We create an Employee Object using builder pattern where id and the name of the employee are mandatory but a hobby, gender, address are optional.

**Code:**

package com.example.advanceSpring.entity;

public class Employee {

private Long id;

private String name;

private int gender;

private String address;

private String hobby;

private Employee()

{

}

public static class EmployeeBuilder{

private Long id;

private String name;

private int gender;

private String address;

private String hobby;

public EmployeeBuilder(Long id, String name)

{

this.id=id;

this.name=name;

}

public void setGender(int gender)

{

this.gender = gender;

}

public void setAddress(String address)

{

this.address = address;

}

public void setHobby(String hobby)

{

this.hobby = hobby;

}

public Employee build()

{

Employee emp = new Employee();

emp.id=id;

emp.name=name;

emp.gender=gender;

emp.address=address;

emp.hobby=hobby;

return emp;

}

}

@Override

public String toString() {

return "Employee [id=" + id + ", name=" + name + ", gender="+( gender == 1?"Male":"Female") + ", address=" + address + ", hobby="

+ hobby + "]";

}

}

Say, This class comes from a third party jar and we need to create Employee Object using Spring Xml based Dependency injection.

**How we can achieve this?**

**Step 1.** Register the builder class using the context of Employee class with $. So class attribute will be look like this

***Class =”com.example.advanceSpring.entity.Employee$EmployeeBuilder”*** *by doing this Spring will understand Employee builder is an inner class of Employee.*

**Step 2:** pass the required parameters using constructor-args.

**Step3**: Pass the optional parameter using property tag.

**Step 4:** Register the Employee bean in Spring XML, While registering one thing you can notice that Employee Object constructor is private so Spring bean can’t call it by reflection. So we will use the factory-bean and the factory-method attribute of spring xml.

**factory-bean** tells Spring to use the bean as a Factory class for creating the Object, so here Spring treats EmployeeBuider as a Factory of Employee Object.

**factory-method** tells Spring to, upon calling the method instance of the Object will return so here build() method act as factory method to create Employee Object.

**The Spring Xml file**

<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-2.5.xsd">

<bean id="helloBean" class="com.example.advanceSpring.HelloWorld">

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

</bean>

<bean id="builder" class="com.example.advanceSpring.entity.Employee$EmployeeBuilder">

<constructor-arg value="1"/>

<constructor-arg value="Shamik Mitra"/>

<property name="gender" value="1"/>

<property name="hobby" value="Blogging"/>

</bean>

<bean id="employee" class="com.example.advanceSpring.entity.Employee" factory-bean="builder" factory-method="build"/>

</beans>

**Step5:** Now we will test the settings.

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.example.advanceSpring.entity.Employee;

public class SpringTest {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext(

"SpringTest.xml");

Employee employee = (Employee) context.getBean("employee");

System.out.println(employee);

}

}

**Step 6:** Output:

Employee [id=1, name=Shamik Mitra, gender=Male, address=null, hobby=Blogging]

Please note that If the Setter method using Fluent Style, that is calling setter it return the instance of that object(**this**) then Spring can’t call this Setter as this is not the method signature Spring calls internally.