**Important note: Before going further you are requested to go through theory of Spring Dependency Injection with look up method mentioned in JavaLive Notes on Spring Core thoroughly so that you can understand the concept very well. (\*\*This activity i.e. going through theory part of respective subject is applicable for each of our project.)**

**What this project does??**

This project explains how DI carried out using Look Up method in spring.

**Need of Lookup Method:**

Consider a scenario when a Singleton bean is using Prototype bean, so when we refer a Singleton bean, the referred Prototype bean get returned, but note that same instance of the Prototype bean is returned on every call. To overcome this drawback, lookup method is used

**By using lookup method we get new instance of prototype bean that is referred by given singleton bean each time when we refer the given singleton bean.**

**\*\*Also note that, we can achieve same kind of functionality by implementing ApplicationContextAware interface by Singleton bean. This code is also explained at the end of this note.**

**Note that this is xml based spring project having configuration file viz. ' spring-config.xml' in resources folder.**

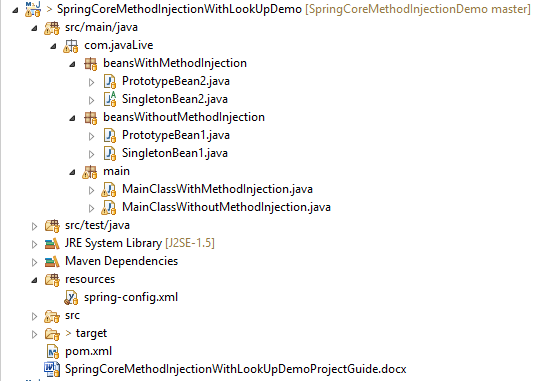
**Steps to create project:-**

1. Create quick start maven project. Refer file 'CreateQuickStartMavenProjectInEclipse.docx' in this project for more details.

2. Add require dependencies for spring as shown in pom.xml file.

3. Create required packages and add the files.

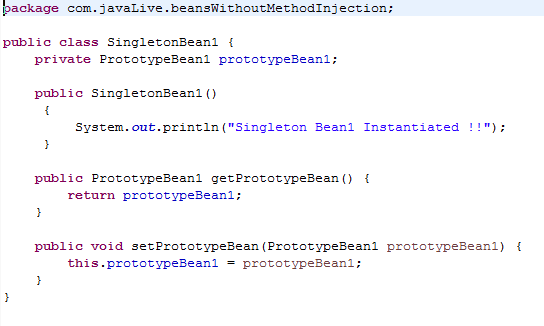
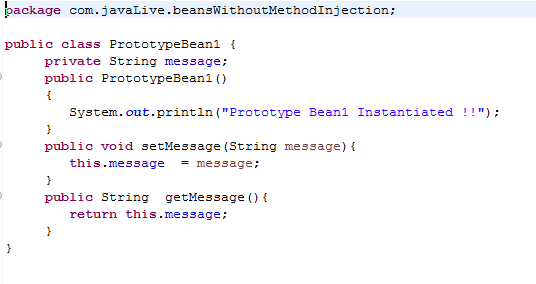
**Project Structure is as follows :-**

****

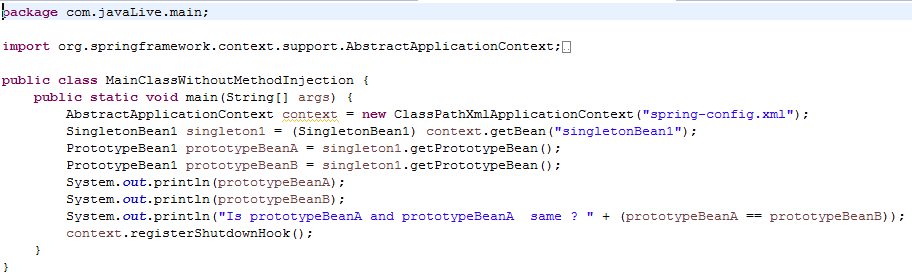
**Functioning of the project**

This project provides comparative analysis of behavior of spring container with and without 'look up method'.

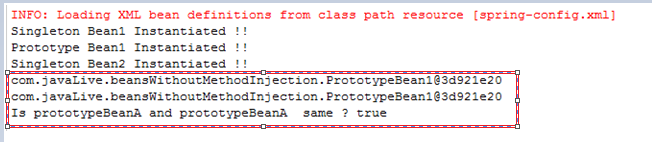
Package com.javaLive.beansWithoutMethodInjection contains normal code without any look up method. Which behaves normally and both beans ' singletonBean1' and ' prototypeBean1' created only once at the time of loading spring application context.



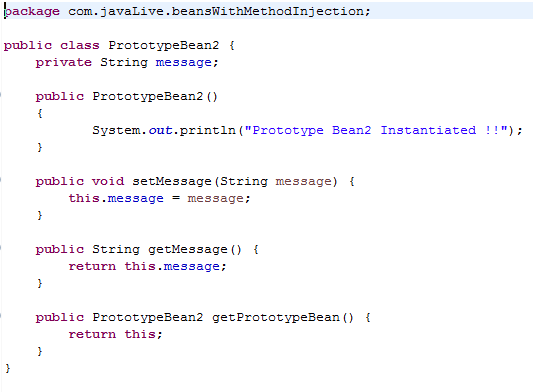


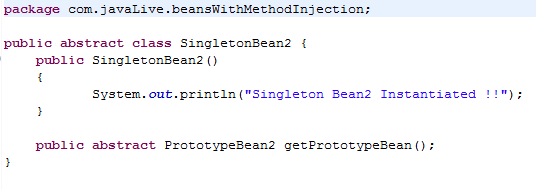


**Look below output, we have got same instance of Prototype bean on new call which is against the definition of Prototype bean hence we need Lookup Method to get new instance of Prototype bean on every new call.**

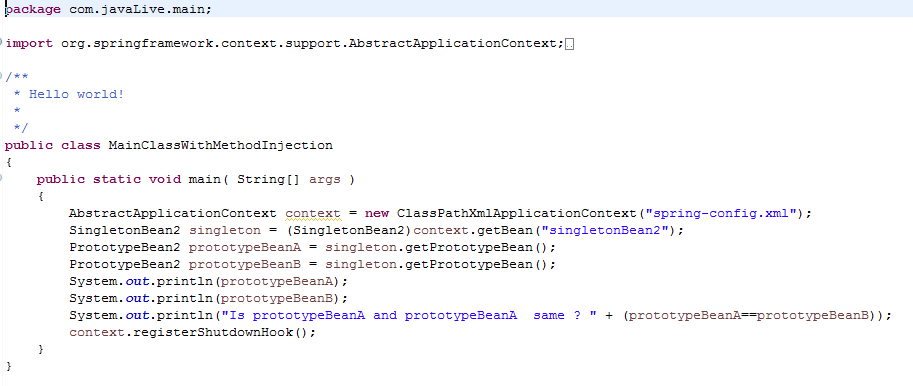
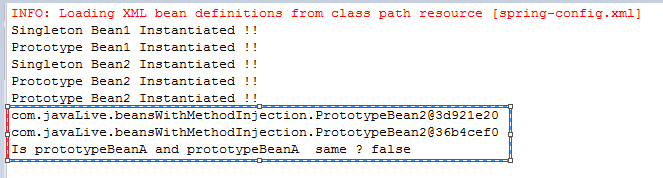


Package com.javaLive.beansWithMethodInjection contains class 'SingletonBean2.java' **with partial lookup method code i.e. abstract method 'getPrototypeBean() ' (Rest of the code in included in spring-config.xml file under definition of bean class 'SingletonBean2' with <lookup-method> tag.)** Thus when we call singleton bean with id 'singletonBean2' in spring-config file, each time fresh instance of bean 'prototypeBean2' will get created.







**Please carefully observe output below. We have got new instance of Prototype bean on every new call. This is possible due to LookUpMethod.** 

**Above mechanism is also possible with following approach:**

**Using Application Context Aware approach**

We discussed Application Context Aware interface in earlier chapters with which we can get the instance of application context in which the bean is configured.

Every time a call getBean() for a prototype object will returns a different bean so

1. Implements Application Context Aware interface in a singleton bean
2. In the getter method of prototype bean, we can explicitly make a call to getBean and returns the object.

|  |  |
| --- | --- |
|  | PrototypeBean.java  public class PrototypeBean {         private String message;         public PrototypeBean()         {                System.out.println("Prototype Bean Instantiated !!");         }         public void setMessage(String message){                this.message  = message;         }         public String  getMessage(){                return this.message;         }  }  SingletonBean.java  import org.springframework.beans.BeansException;  import org.springframework.context.ApplicationContext;  import org.springframework.context.ApplicationContextAware;  public  class SingletonBean implements ApplicationContextAware {         private PrototypeBean prototypeBean;         private ApplicationContext applicationContext;         public SingletonBean()         {                System.out.println("Singleton Bean Instantiated !!");         }         public PrototypeBean getPrototypeBean()  {  prototypeBean= (PrototypeBean)applicationContext.getBean("prototypeBean");         return prototypeBean;         }         @Override         public void setApplicationContext(ApplicationContext applicationContext)                       throws BeansException {                this.applicationContext=applicationContext;         }  } |

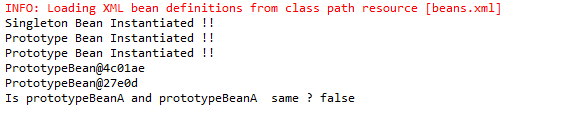
Define entries of beans in beans.xml – In the singleton bean , we did not add the dependency of prototype bean, instead the dependency is resolved programatically

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?>  <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-3.0.xsd">         <bean id="prototypeBean" class="PrototypeBean" scope="prototype">                <property name="message" value="Test Message" />         </bean>         <bean id="singletonBean" class="SingletonBean" >         </bean>  </beans> |

Test Program

|  |  |
| --- | --- |
|  | import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;  public class TestProgram {         public static void main(String[] args) {                ApplicationContext context =                new ClassPathXmlApplicationContext("beans.xml");                SingletonBean singleton = (SingletonBean)context.getBean("singletonBean");                PrototypeBean prototypeBeanA = singleton.getPrototypeBean();                PrototypeBean prototypeBeanB = singleton.getPrototypeBean();                System.out.println(prototypeBeanA);                System.out.println(prototypeBeanB);                System.out.println("Is prototypeBeanA and prototypeBeanA  same ? " + (prototypeBeanA==prototypeBeanB));         }  } |

Run the program and we can see that getPrototypeBean() returned different  object both time



**Conclusion**

Method Injection is the approach in which method name is configured which will be used to resolve the dependency. This approach  is primarily used when a singleton bean has a dependency on a prototype bean and a new instance of prototype bean is required from a singleton bean every time.