Autowiring :

Autowiring is a concept where there is no need to inject secondary dependancies. Spring container will inject secondary dependancies on its own.

Autowiring supports only secondary type dependancies(not primitive).

By default Autowiring is disabled. If we enable Autowiring, Spring container will take care of injecting secondary type dependancies.

Autowiring can be achieved –

1. byName
2. byType
3. Constructor
4. autoDetect
5. none (bydefault)
6. by Name :

The byName mode injects the object dependency according to name of the bean. In this case, bean id and reference name must be same.

For eg. Consider Student class with secondary dependency of Address class as ‘address’ then:-

Spring container will find secondary dependancy in Student class, then it will take name (add) & matches it with bean id.

If match found, then it verifies the class of bean & type of secondary injection is same or not.

If same : Setter Injection

If not : property remains unwired(as id is unique, no ambiguity occurs).

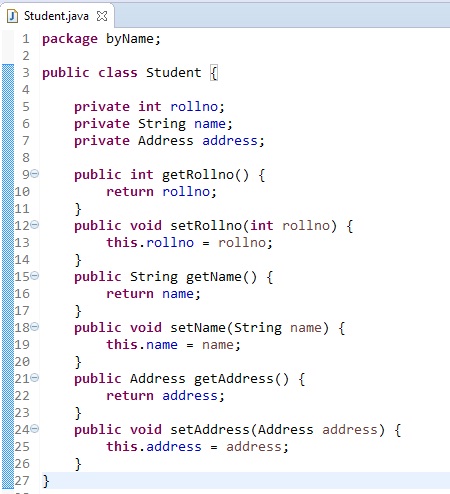
Let us go throgh an example for this..

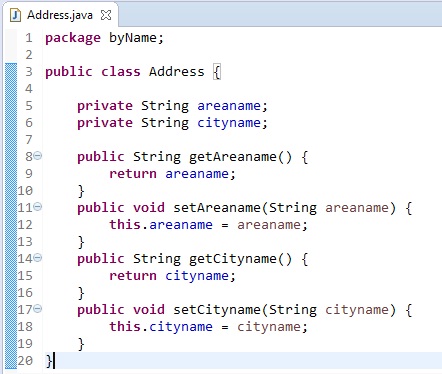
Files required : 1)Student.java

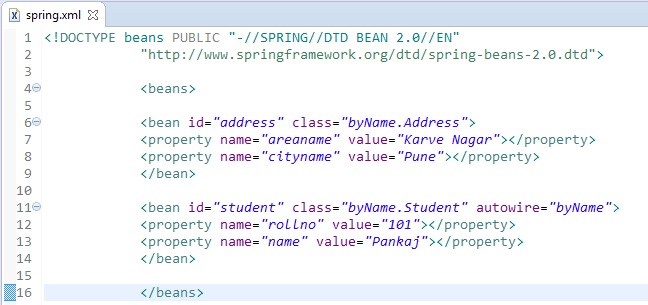
2) Address.java

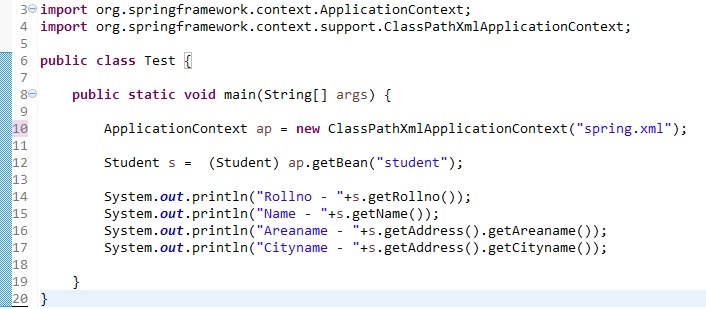
3) Test.java

4)spring.xml +SpringJars

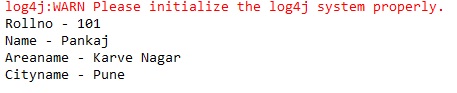








Output :



1. by Type:

The byType mode injects the object dependency according to type. In this case, bean id and reference name may be different. But there must be only one bean of a type.

For eg. Consider Student class with secondary dependency of Address class as ‘address’ then:-

Spring container will find secondary dependancy in Student class, then it will take type of that dependancy (Address) & matches it with class in bean.

If match found, it will inject property using Setter Injection.

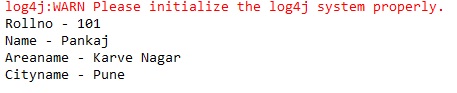
If match not found, property remains unwired.

here, Student.java, Address.java and Test.java will be same as in above example.

So we have shown only spring.xml–



Output :



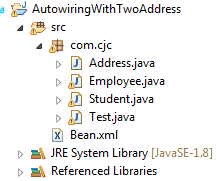
Note:

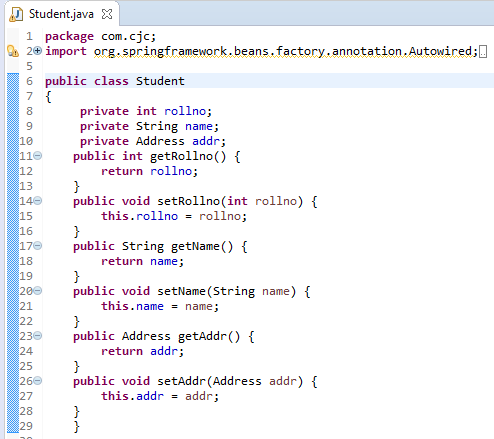
If multiple bean with same class are found, ambiguity will occur. To resolve this add tag “autowire-candidate=”false””, now that bean will not get involved in autowiring process.

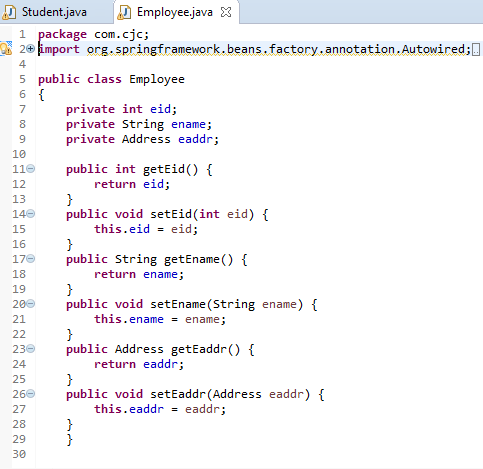
For example if we have student and Employee class and both have address field with ref of Address class so we have to create Address class only one time because address class fields are same for both student and employee.

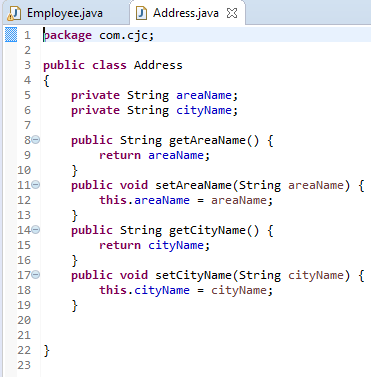
So we create two beans of address class with two different address.but class name is same for both beans so if we go for ByType autowire then it throw exception(org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'com.cjc.Address' available: expected single matching bean but found 2: addr,eaddr)

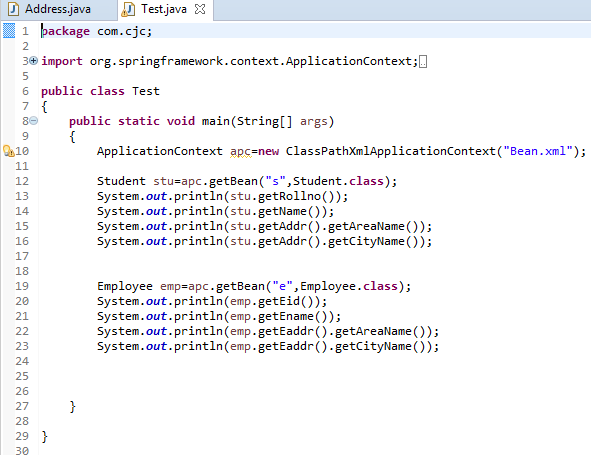
s it found two beans with same class name. To resolve this add tag “autowire-candidate=”false””, now that bean will not get involved in autowiring process.

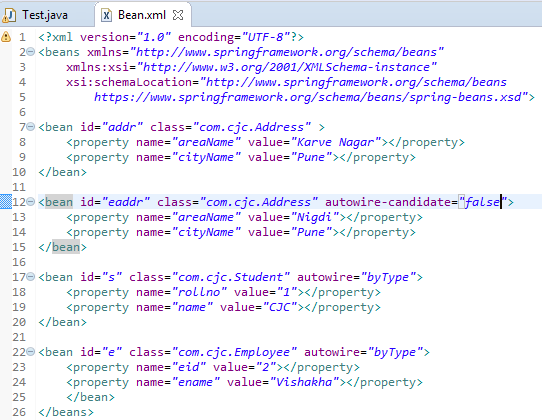




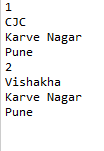








output:



1. Constructor:

The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters.

It internally uses ‘byType’, but while injecting property it uses constructor based injection.

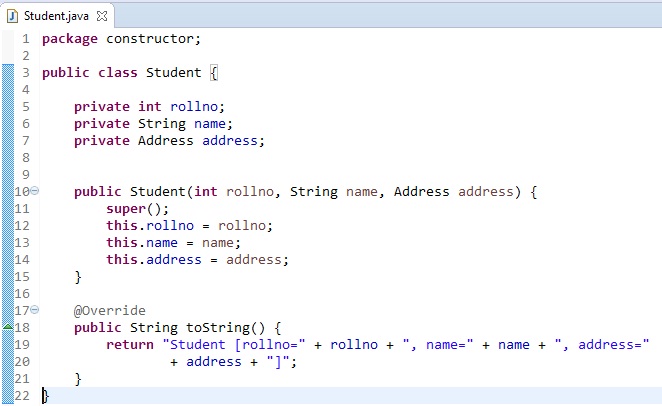
Let us go through an example for this..

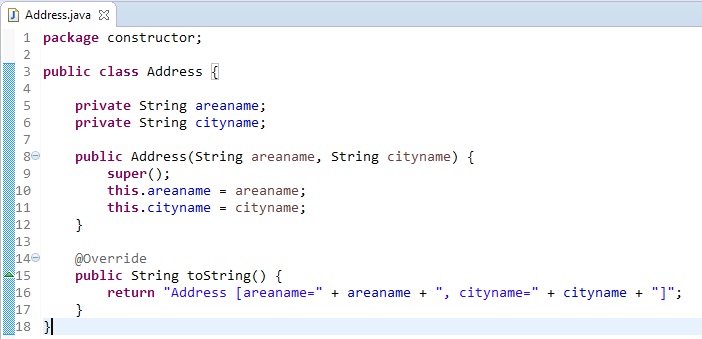
Files required : 1)Student.java

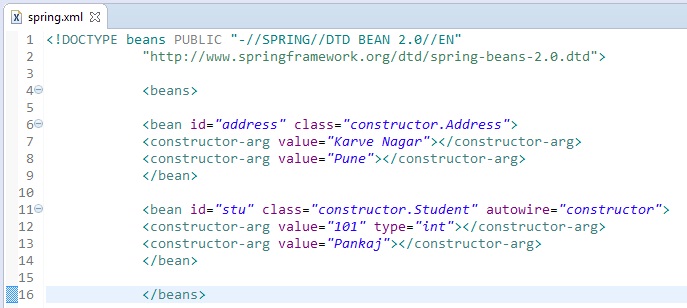
2) Address.java

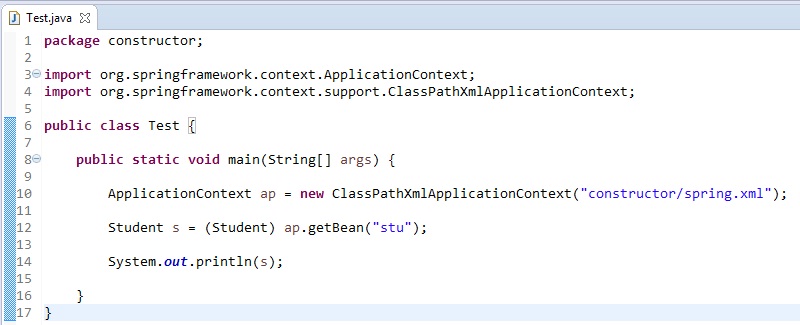
3) Test.java

4)spring.xml +SpringJars









Output :



Autowiring with Annotation:

We can use Spring @Autowired annotation for spring bean autowiring.

For @Autowired annotation to work, we also need to enable annotation based configuration in spring bean configuration file. This can be done by <**context:annotation-config />** element or by defining a bean of type org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.

In the following example, we have defined bean for enabling the @Autowired annotation

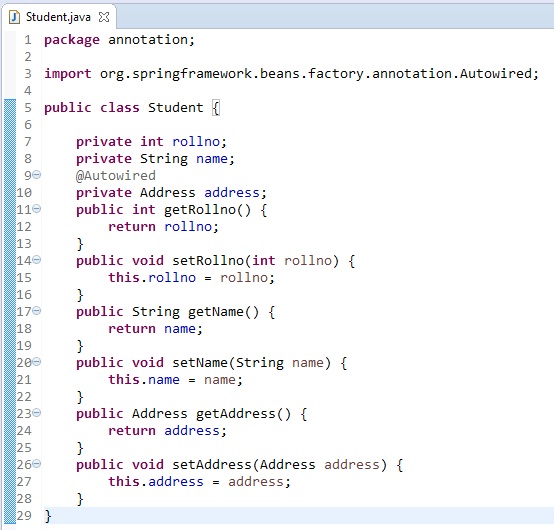
Files required : 1)Student.java

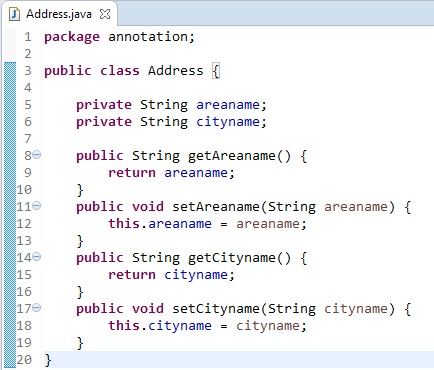
2) Address.java

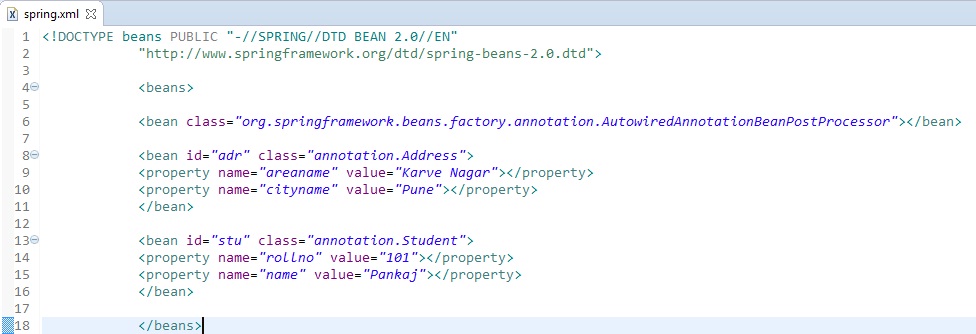
3) Test.java

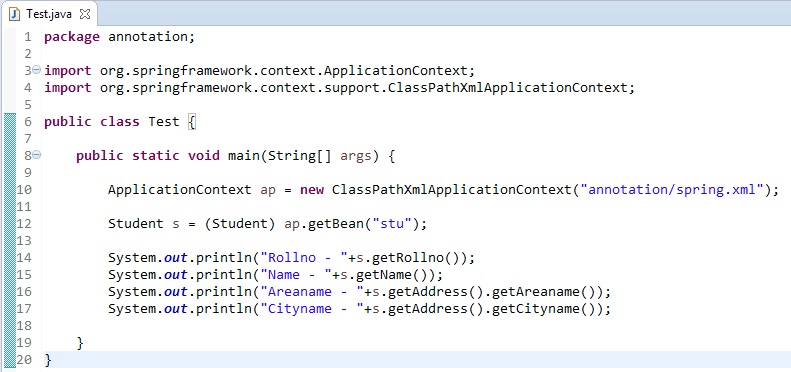
4)spring.xml +SpringJars

0

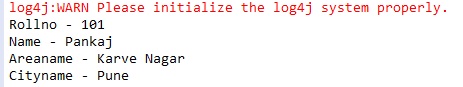






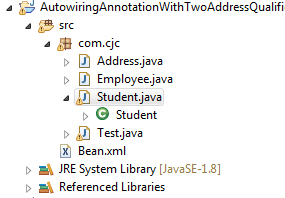


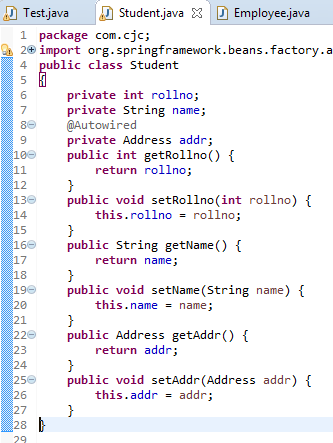
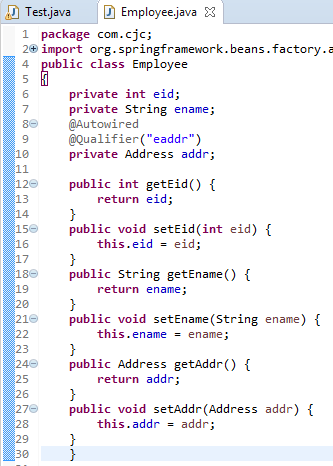
Output :

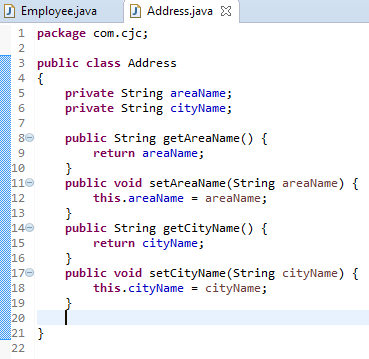


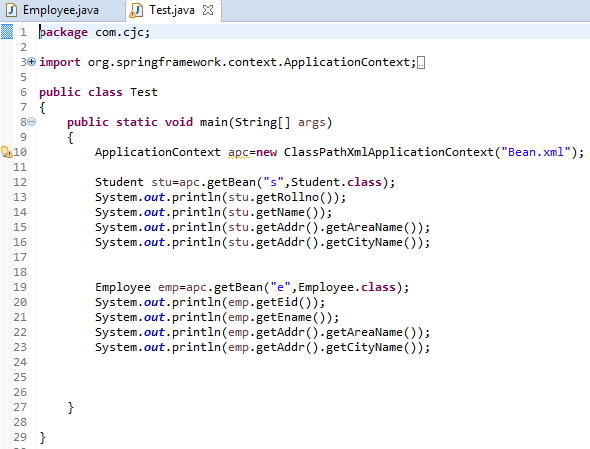
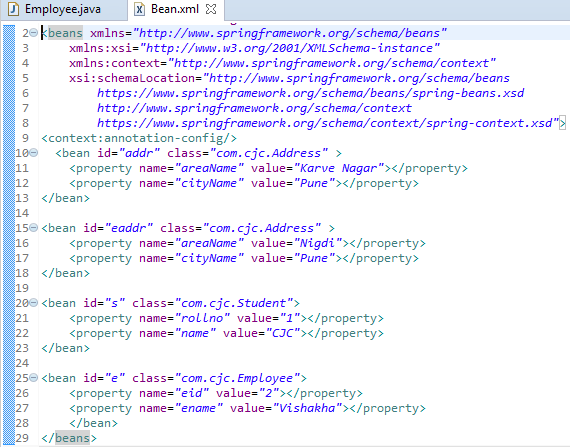
Use of @Qualifier annotation:

The @Qualifier annotation is used to resolve the autowiring conflict, when there are multiple beans of same type. The @Qualifier annotation can be used on any class annotated with @Component or on method annotated with @Bean . This annotation can also be applied on constructor arguments or method parameters.





O/P:

