Bean Scope :

When defining a <bean> in Spring in xml file, you have the option for declaring a scope for that bean.

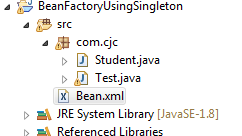
The Spring Framework supports following six scopes-

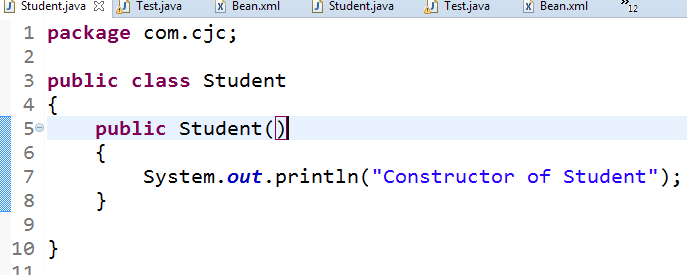
* + *singleton* : This scopes the bean definition to a single instance per Spring IoC container (default).
  + *prototype* : This scopes a single bean definition to have any number of object instances. Opposite to singleton, it produces a new instance each and every time a bean is requested.
  + *request* : This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
  + *session* : This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
  + *websocket*: A single instance will be created and available during complete lifecycle of WebSocket.

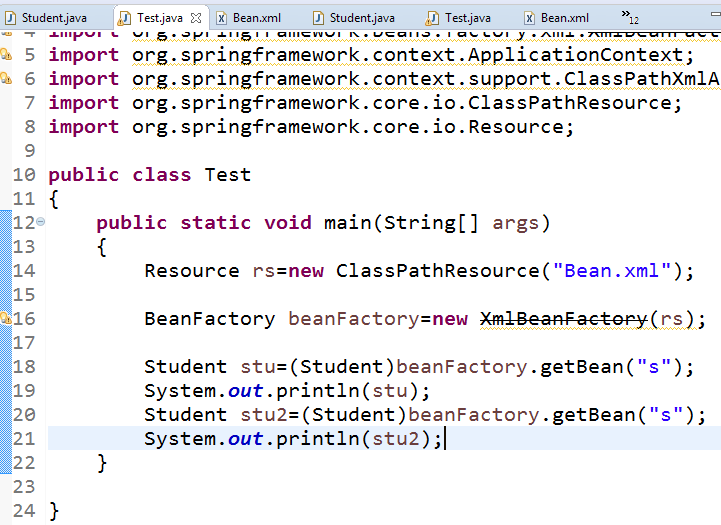
Only valid in web-aware Spring ApplicationContext

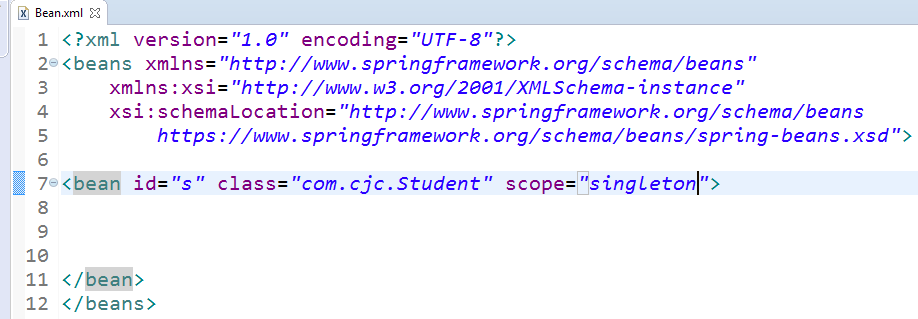
* + *application*: A single instance will be created and available during complete lifecycle of ServletContext. Only valid in web-aware Spring ApplicationContext.

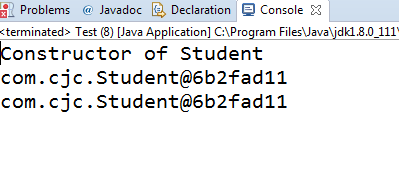
**Singleton example:**



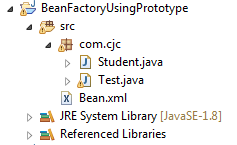


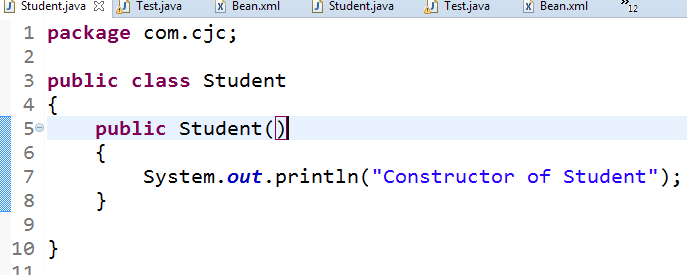


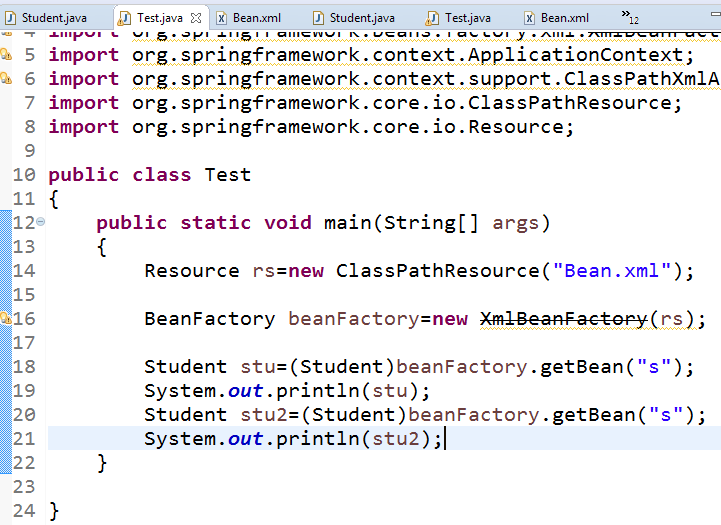


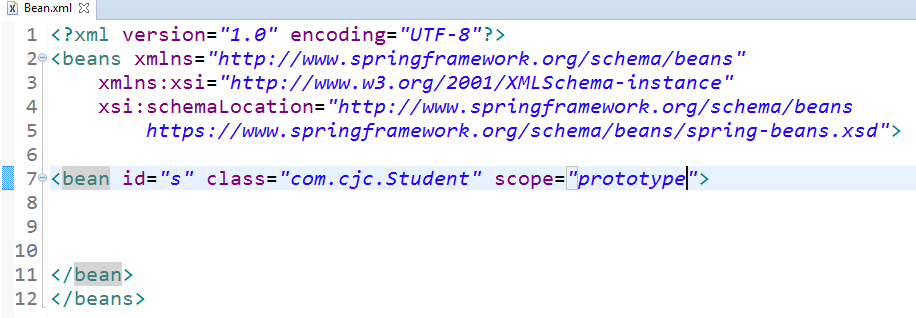
****

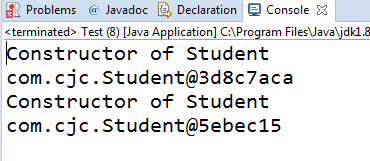
**2)prototype Scope:**











Lazy & Eager Loading :

For Singleton scope : only one object for bean will be created

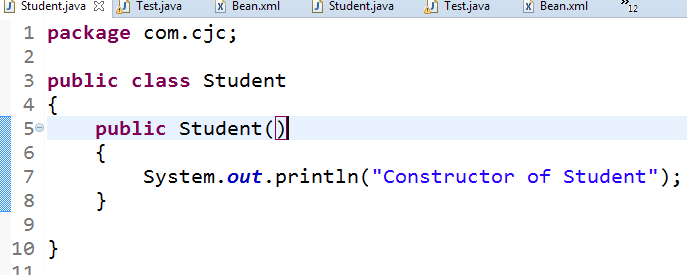
* BeanFactory : lazy-loading (i.e. object is created on request)
* ApplicationContext : eager- loading (i.e. object is created when xml file loads)

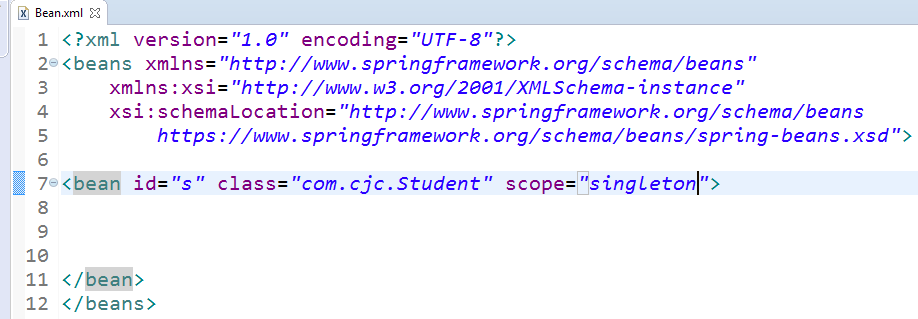
Let us see above concept practically...

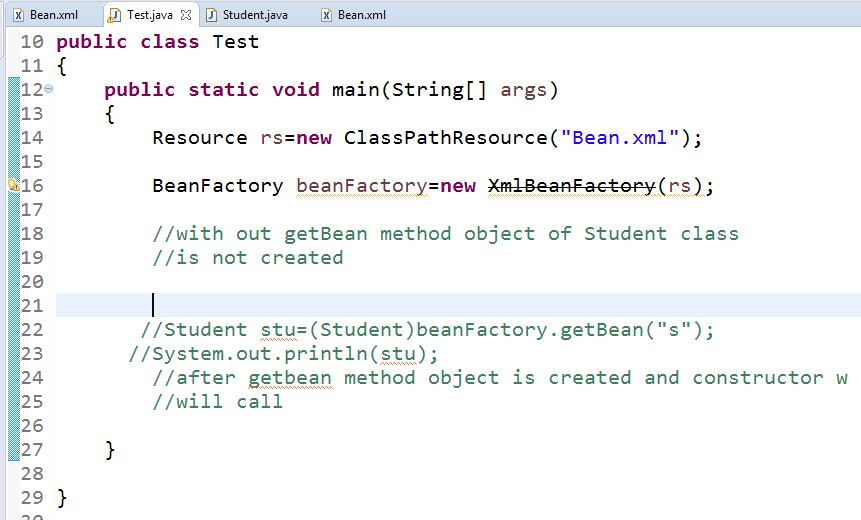
For prototype Scope:

* BeanFactory : lazy-loading (i.e. object is created on request)
* ApplicationContext : Lazy- loading (i.e. object is created on request)

Let us see above concept practically...







If BeanFactory scope is prototype then also lazy loading will Occure

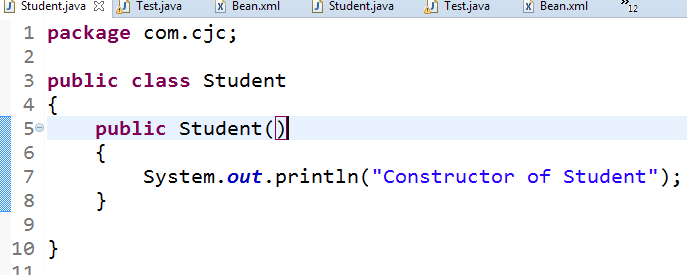
All program is same as previous program

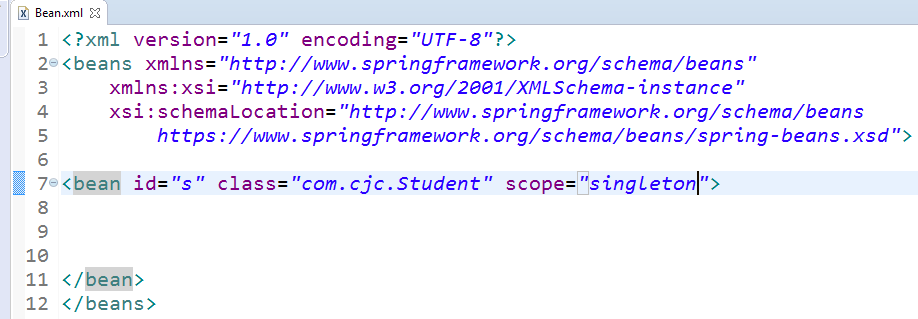
Only change scope=prototype

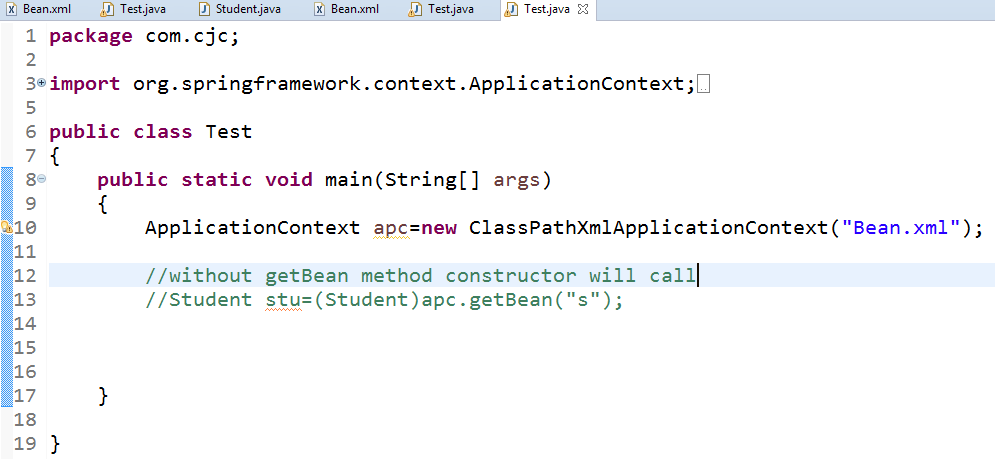
* ApplicationContext : eager- loading (i.e. object is created when xml file loads)

Let us see above concept practically...

For this scenario eager loading will occure







LazyLoading And EagerLoading both container:

|  |  |  |
| --- | --- | --- |
| Containers | Singleton | Prototype |
| BeanFactory | LazyLoading | LazyLoading |
| ApplicationContext | EagerLoading | LazyLoading |

Dependancy Injection:

In our java class datamembers are nothing but the dependancies and assigning values to them is Dependancy Injection.

Types of DI:

1)Primitive Type

2)secondary type

3)Collection Type

This Dependancy Injection can be achieved in 2 ways –

1. Setter Injection
2. Constructor Injection

In Spring these dependancies will be injected by Spring IOC container to our Spring class (bean).

Spring container will come to know about injection type from spring configuration file.

Setter Injection :

In Setter Injection, container uses setter methods in POJO class to inject dependancies.

In this, our class must contain setter methods otherwise we cannot inject dependancies.

Constructor Injection :

In Constructor Injection, container uses constructor in class to inject dependancies.

Note:

If both Constructor & Setter Injections are applied on same property then Constructor Injection will be overridden by Setter Injection. Since, Constructor Injection takes place when object is created.

Setter Injection

Constructor Injection

Partial injection of dependancies is possible.

Partial injection of dependancies is not possible.

Setter injection will override constructor injected values.

Constructor injection will not override Setter injected values.

Constructor injection makes bean class object as immutable (unchangeble).

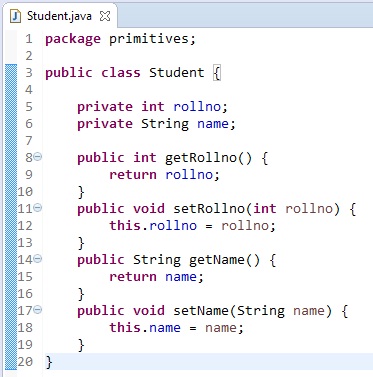
Setter injection makes bean class object as mutable (changeble).

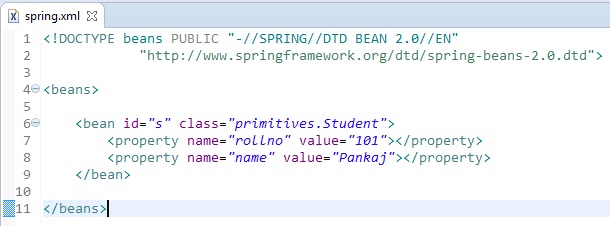
Setter Injection–primitives:

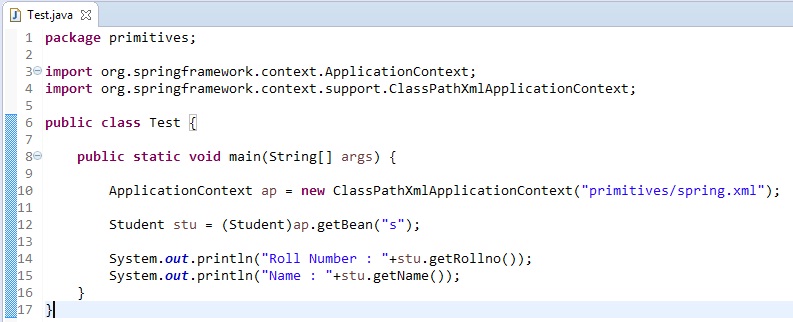
Files required : 1)Student.java

2) Test.java

3)spring.xml +SpringJars







Output :



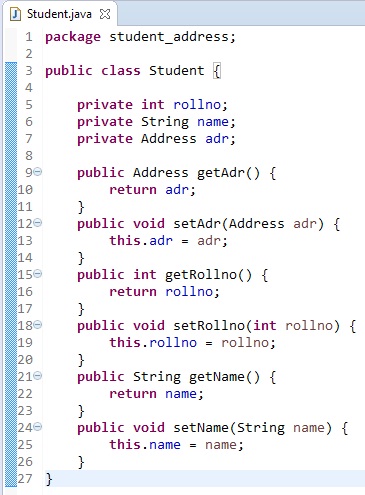
Setter Injection –secondary:

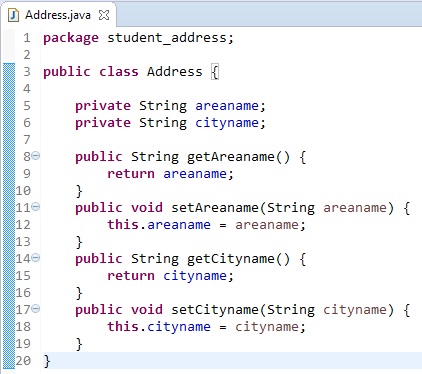
Files required : 1) Student.java

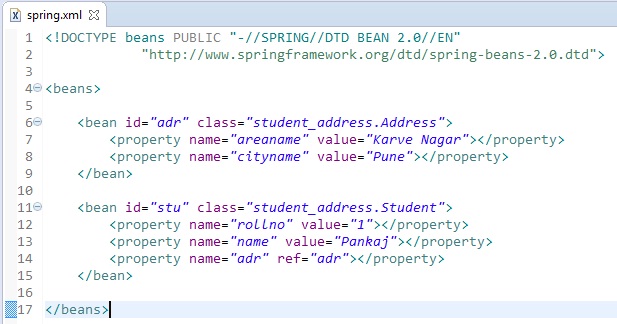
2) Address.java

3) Test.java

4) spring.xml +Spring Jars









Output :

