Spring Cloud & Microservices

* Spring Boot
* Spring Microservices
* Deploying the microservices on cloud using Docker & Kubernates

Pre-requisites

* Java
* Spring Framework

OOPS concepts

Encapsulation: Where data is private and the only way you can access it is through public methods (setters & getters)

Spring Framework:

It is an application framework used to develop any kind of applications like

* Desktop/Console based
* Web applications
* Enterprise applications
* Mobile applications
* Cloud based applications

Framework: It allows you to develop complex applications in a simple way, framework takes care of all the repeating tasks in the application, it provides all the common features which every application needs, so that development time will be less and production will be more

Benefits we get from Spring Framework

* Dependency Injection Feature (Object creation)
* MVC based web applications (layered architecture)
* REST API’s
* Spring Boot
* Microservices
* Cloud deployment

Spring Framework examples has to be done with Maven project, because maven avoids dependency problems

i.e., Developers may forget to add some jars into their classpath when working on their project, this can be solved with Maven, as maven keeps all the dependencies of jars in its repository

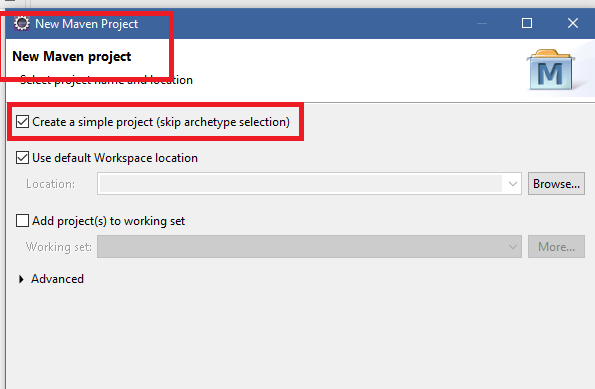
Things to understand in maven project

1. group-id: organization id
2. artifact-id: project-id or project-name
3. version: version number of the release
4. packaging: jar or war

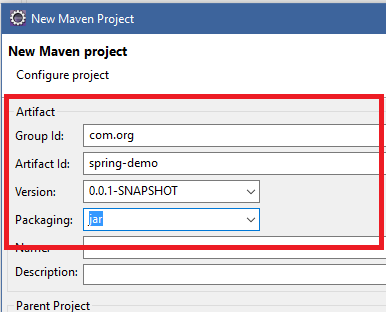
First Spring Framework program

1. Create maven project
2. Select create a simple project (skip archetype)
3. Enter project informations like group-id, artifact-id, packaging etc
4. Once project is created add dependencies i.e., jars for the project like spring-context, spring-mvc, mysql-connector and etc in pom.xml
5. You can also add java version in pom.xml if your maven project doesn’t have the right version of java matching to your system

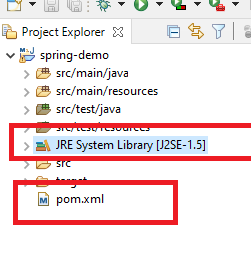
Step1:



Step2:

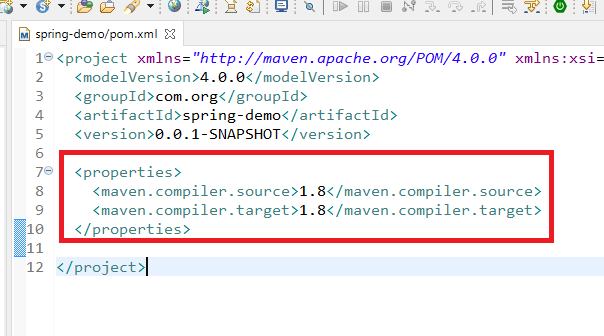


Step3 checking pom.xml & java version



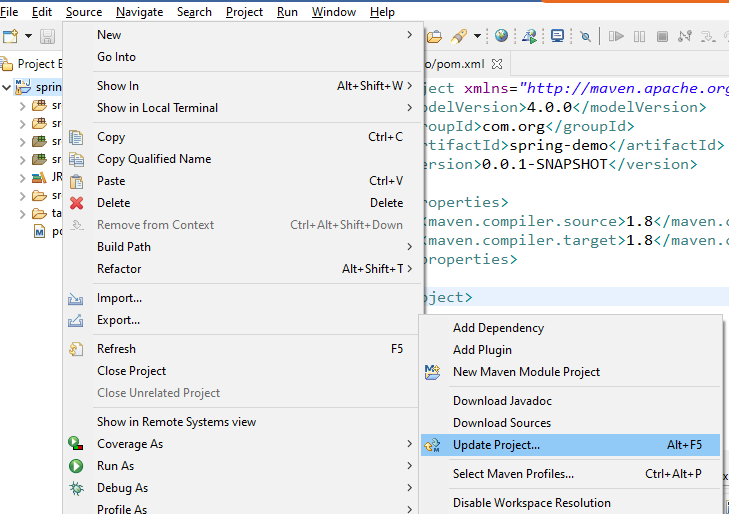
Since Maven project is having Java 1.5 we need to configure pom.xml to use 1.8

pom.xml

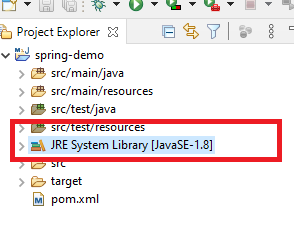


Update maven project

Project -> Maven -> Update Project



Now you can notice 1.8 in maven project



Adding Spring dependencies into pom.xml

List of dependencies we need are:

Spring Modules : dependency-names

1. Spring Core: spring-context
2. Spring MVC: spring-webmvc
3. Spring ORM: spring-orm
4. Spring AOP: spring-aop

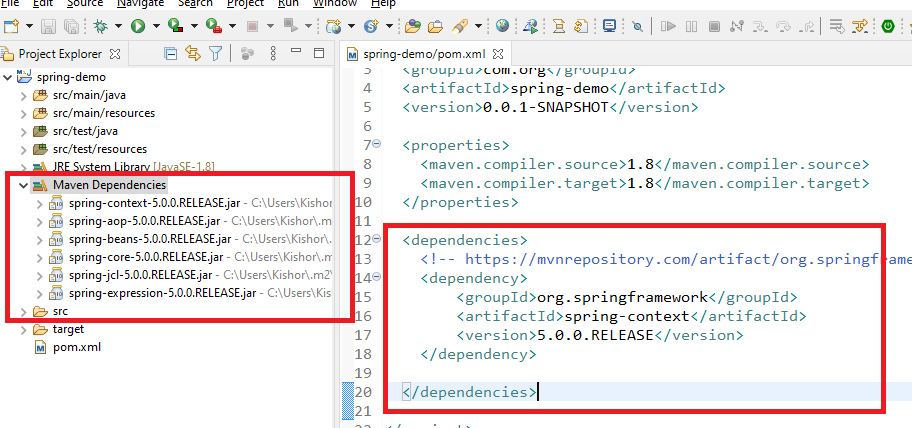
Spring Core: Here you can achieve Dependency Injection

Spring MVC: You can develop MVC based application & REST APIs

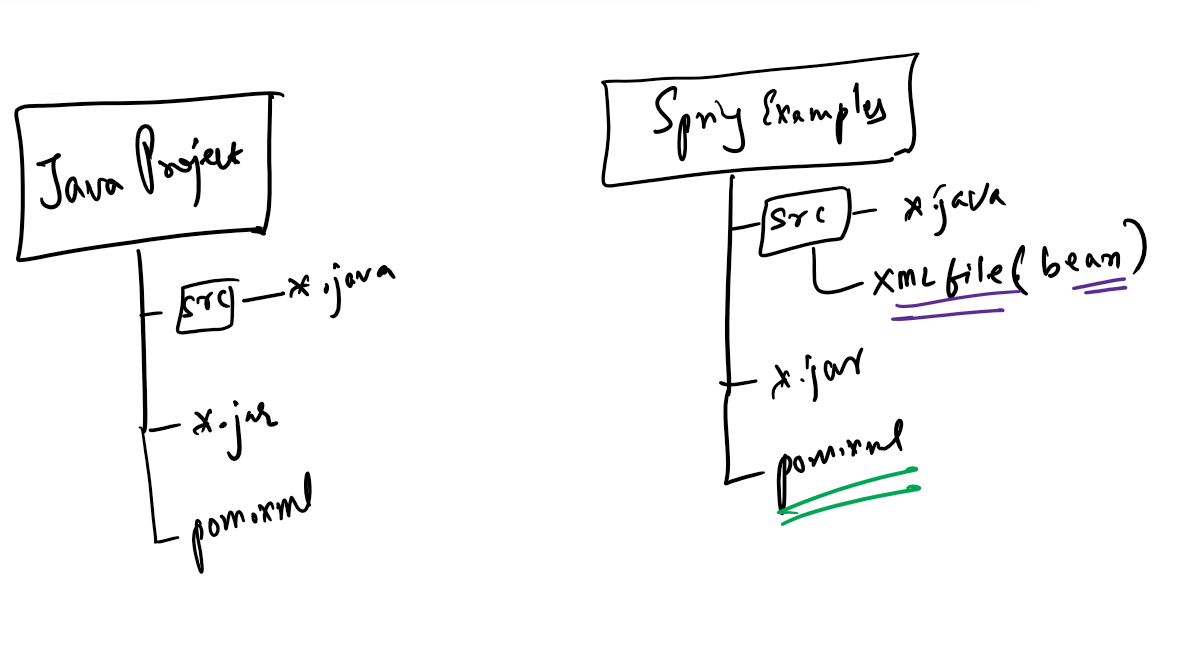
Spring ORM: You can interact with any database using Object Relational Mapping

Spring AOP: It is for Aspect Oriented Programming (Out of scope)

Update pom.xml to have dependency of spring-context



Project Structure for Spring Examples



Normal Java project doesn’t need xml file for bean configurations, however spring example use one xml file which will have bean configurations.

Bean Configuration:

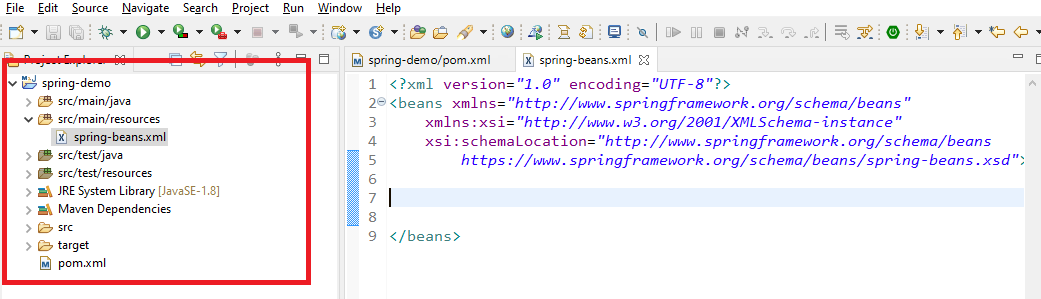
This configures the classes that needs to instantiated by spring framework to manage the objects that depends on other objects, this provides dependency injection feature to us

Sample code of bean configuration file

<beans>  
 <bean id = “x” class = “className”></bean>  
 <bean id = “y” class = “className”></bean>  
</beans>

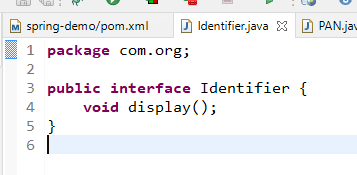
Note: The name of the xml file can be anything like spring-beans.xml, beans.xml, spring-config.xml, abc.xml and so on.

A simple bean configuration file with the name spring-beans.xml looks like

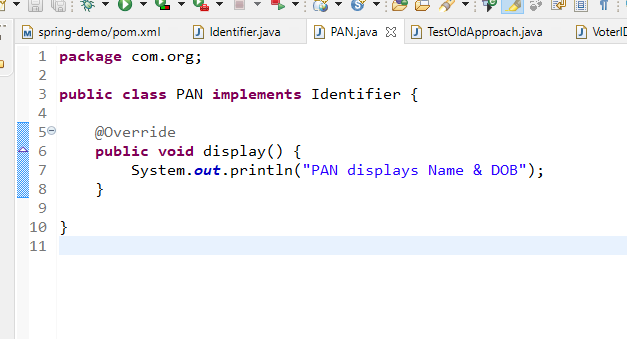


Note: This xml file has to be downloaded either from internet or from spring website

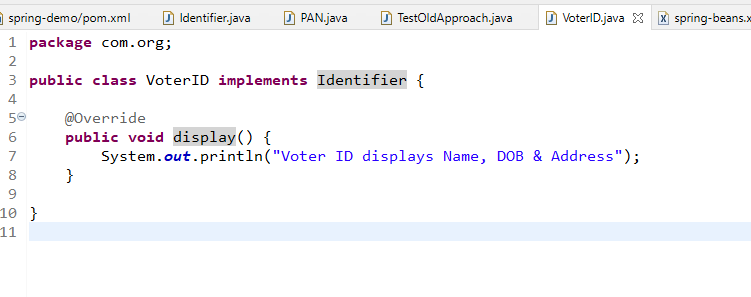
Identifier.java



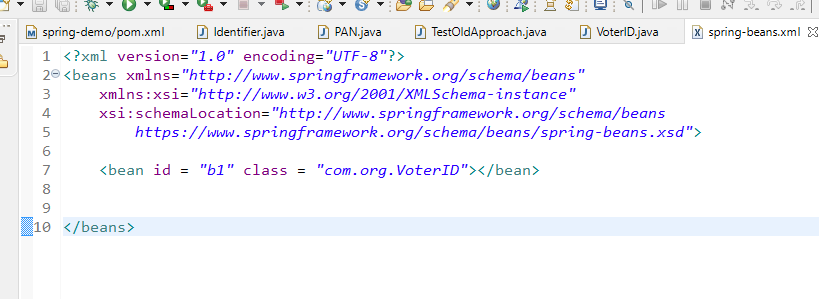
PAN.java



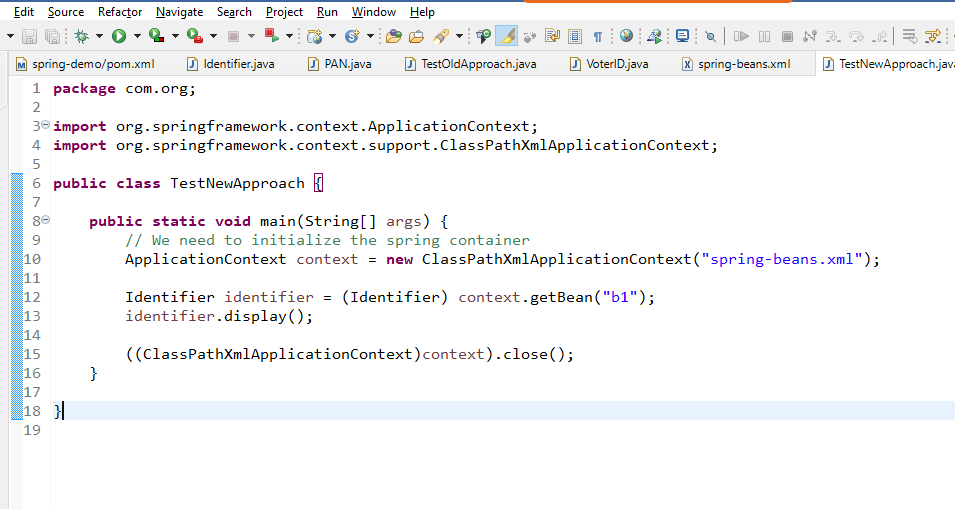
VoterID.java



spring-beans.xml

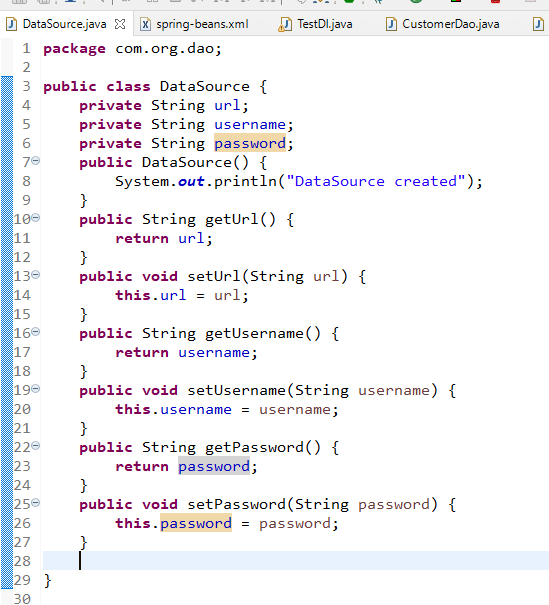


TestNewApproach.java

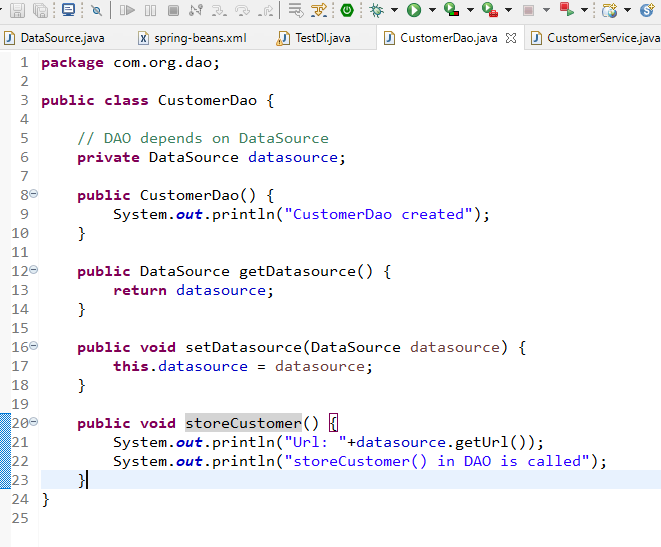


Dependency injection of Datasource->DAO->Service with XML

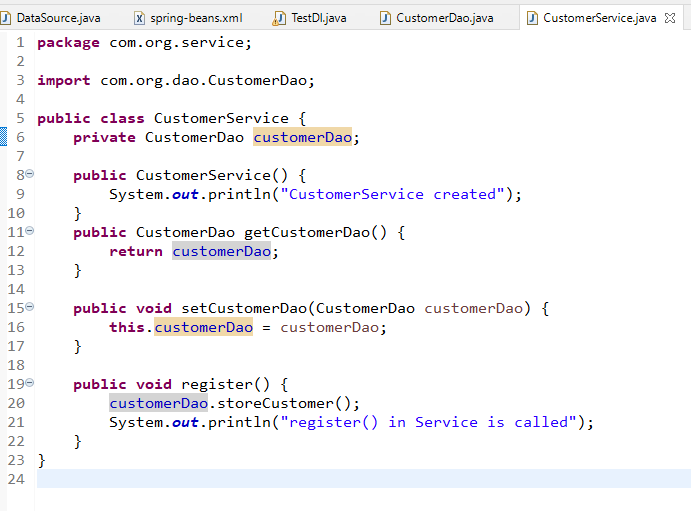
Datasource.java



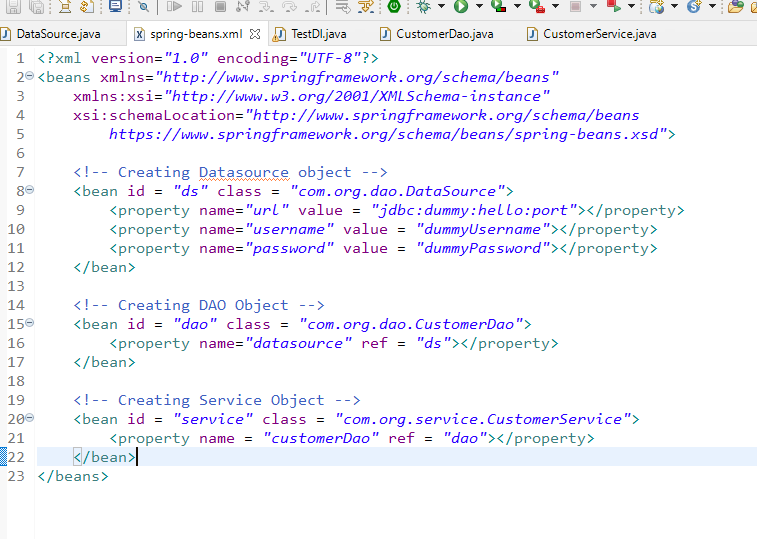
CustomerDao.java



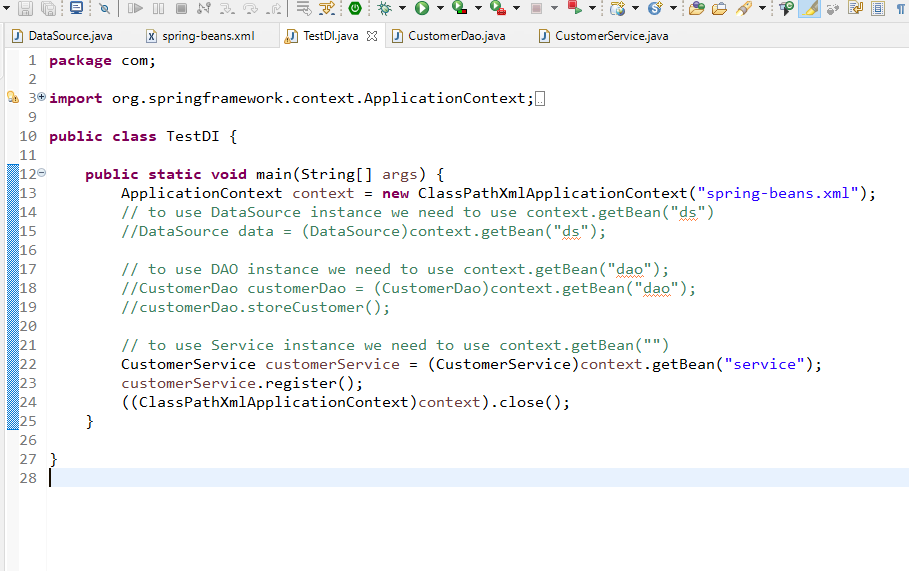
CustomerService.java



spring-beans.xml



TestDI.java



Spring Framework uses annotation instead of <bean> configuration, so that spring container can maintain the object

Some of the important annotations

1. @Component: similar to <bean>
2. @Controller: to mark the controller class
3. @RestController: to mark the RESTApi class
4. @Service: to mark the service layer classes
5. @Repository: to mark the DAO layer classes

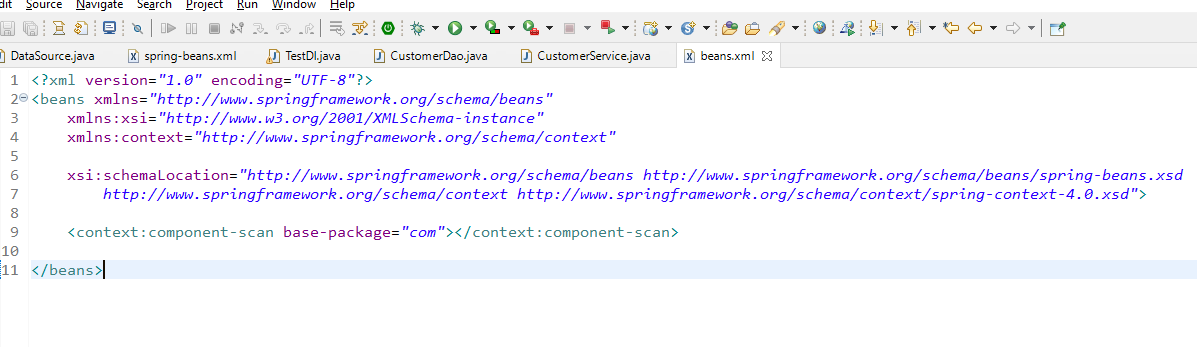
All these annotations should be written on top of the classes, the spring container will create object for these classes

To configure the beans through annotations you need to use a tag called

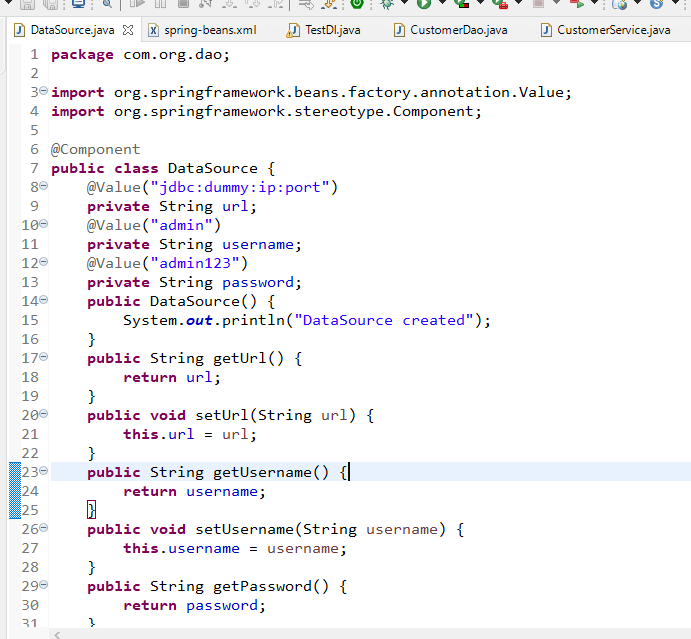
<context:component-scan package = “com”>

@Component  
public class Test { }   
@Controller  
public class Demo { }   
@Service  
public class EmpService { }  
@Repository  
public class EmpDao {}  
@RestController  
public class EmpController {}

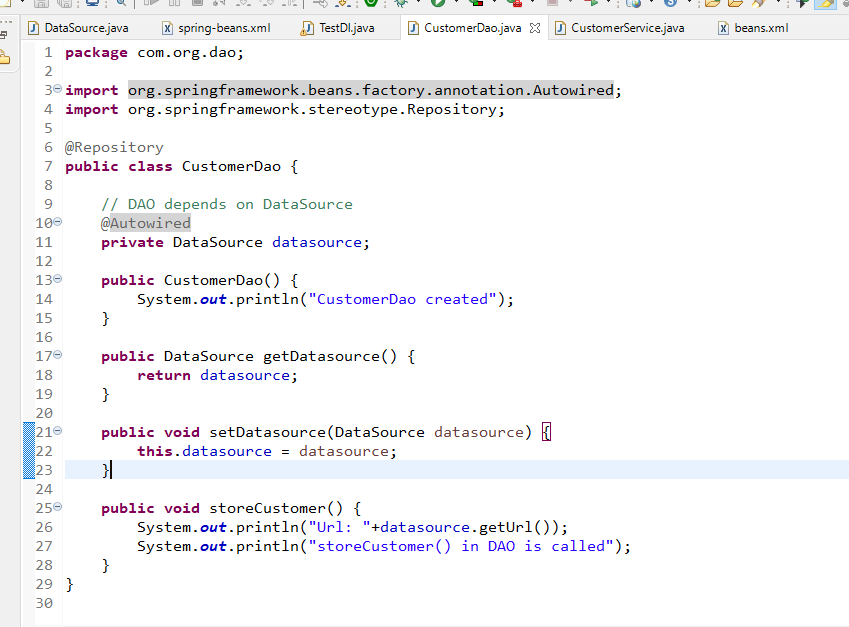
beans.xml



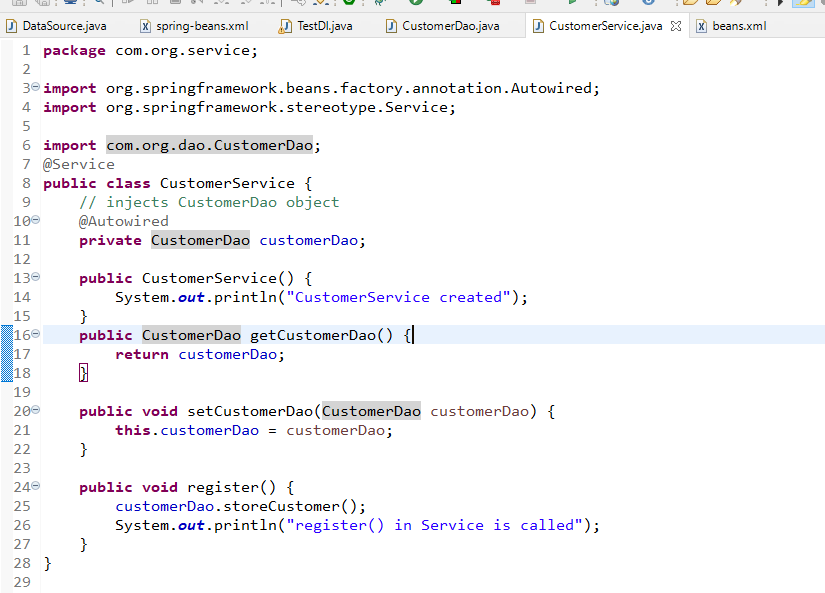
DataSource.java



CustomerDao.java



CustomerService.java



TestDI.java

