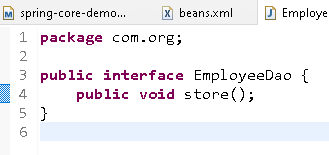
Spring Notes

Spring Framework implements lot of design patterns required to develop the applications like

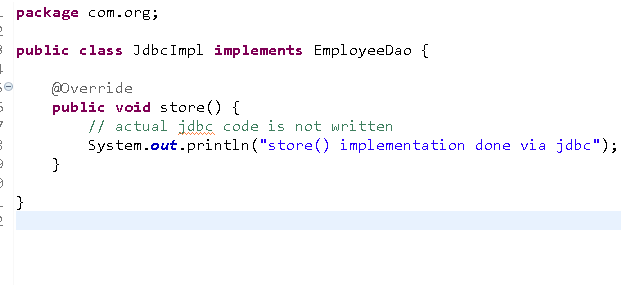
* Dependency Injection
* Singleton pattern
* Front controller pattern
* Proxy pattern
* Factory pattern

Old Approach

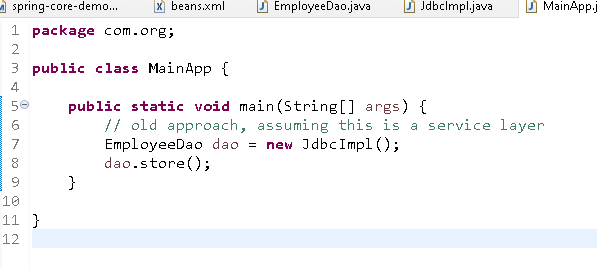
EmployeeDao.java



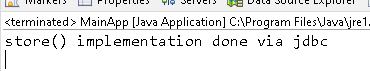
JdbcImpl.java



MainApp.java



Output:



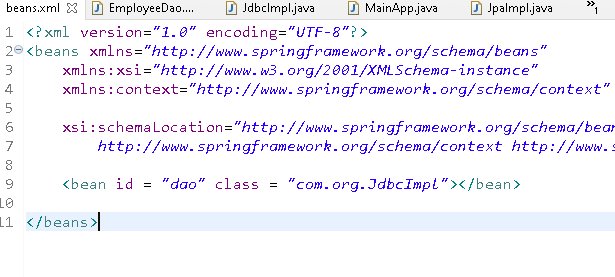
Suppose a new implementation you want to use then you must change the dependency code, which can be completely avoided through spring dependency injection feature

Dependency Injection: process of supplying the dependency to other objects

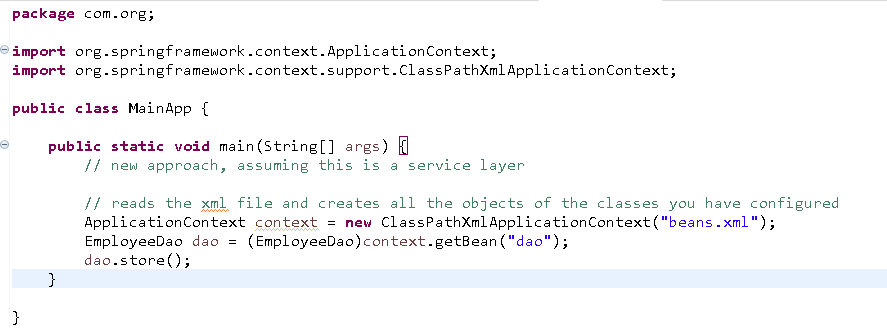
New Approach:

Mention the dependencies to be created in an xml file and use ApplicationContext to load the xml file that enables spring to create the object of all the classes present in the xml

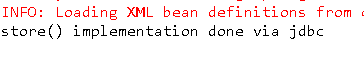
beans.xml



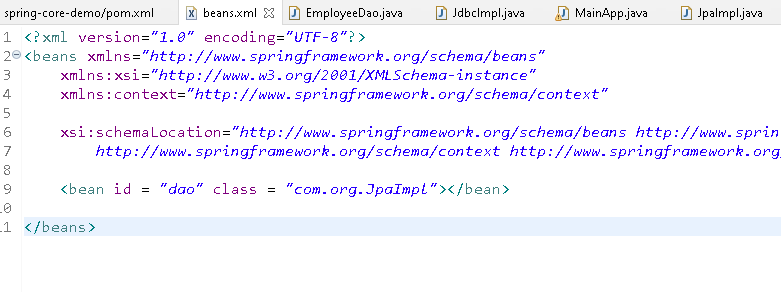
MainApp.java



Output:



Changing the xml to get JpaImpl



Output:



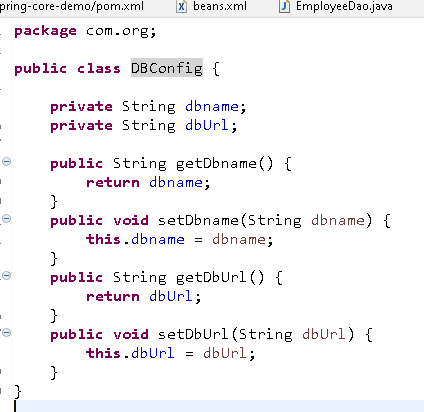
Dependency injection are of two types

1. setter injection
2. constructor injection

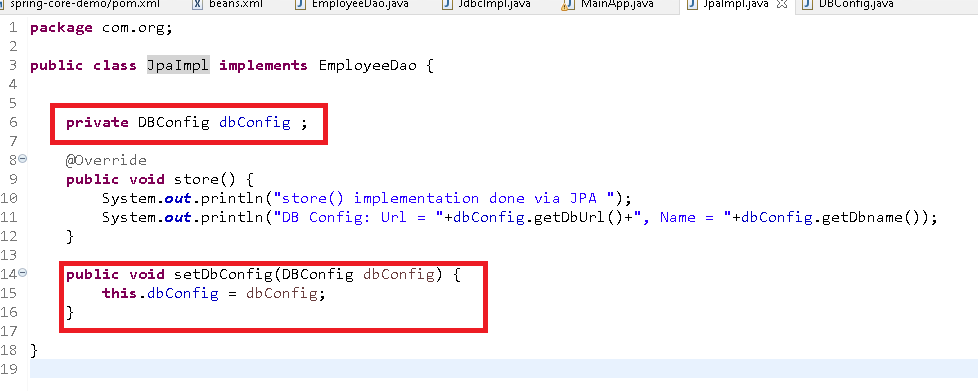
Setter injection means setter method will be called by passing a reference or a value

Constructor injection means parameterized constructor will be called by passing a reference or a value

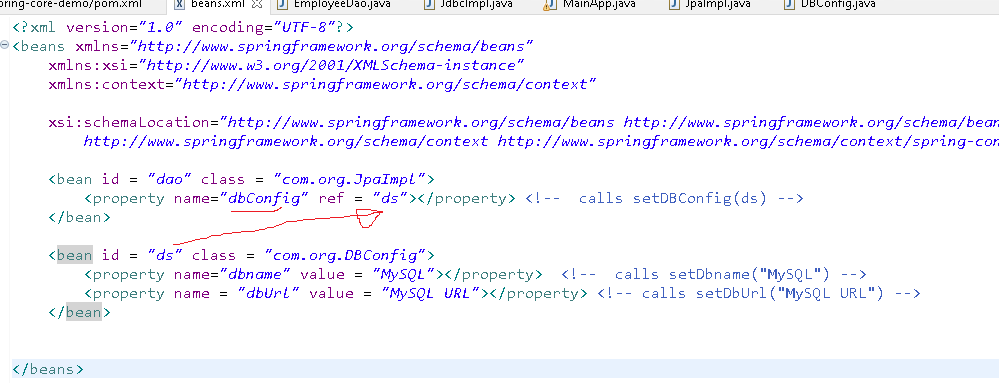
DBConfig.java



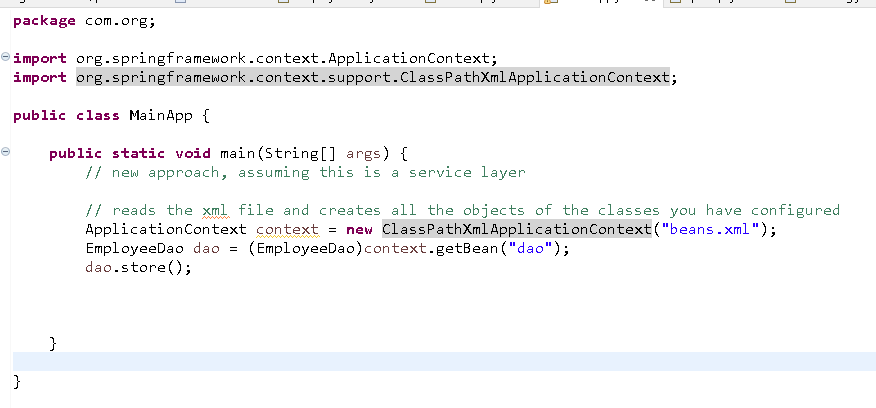
JpaImpl.java



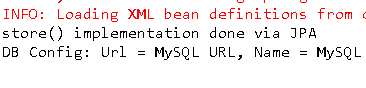
beans.xml



MainApp.java

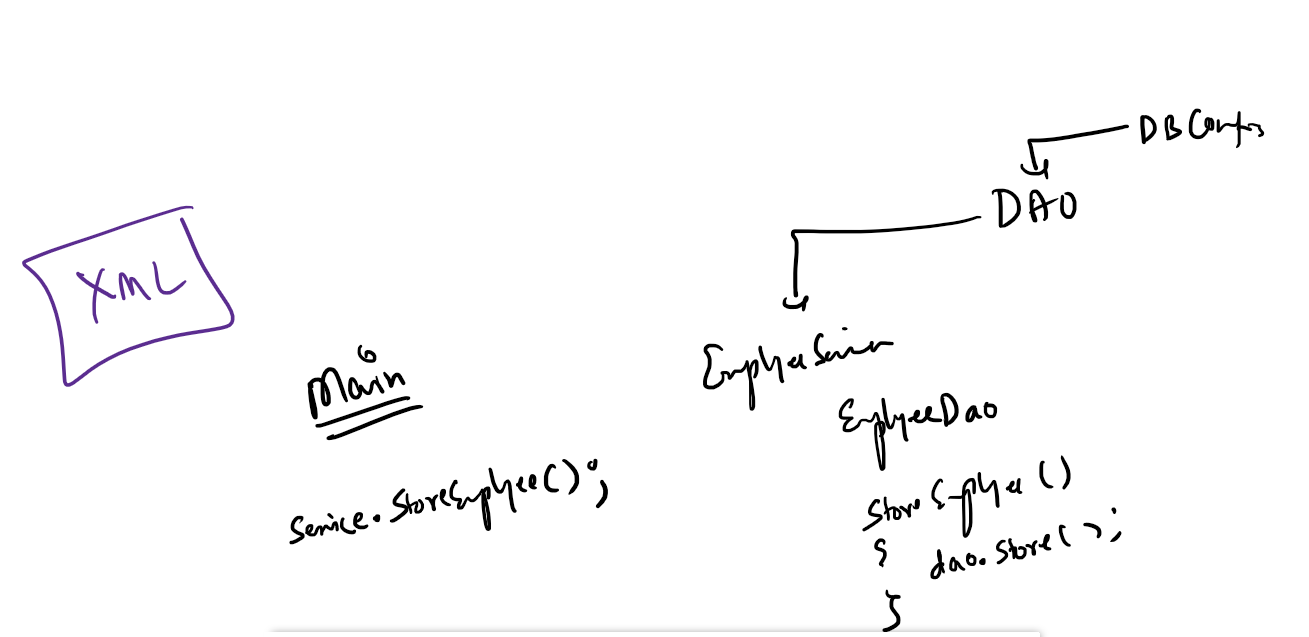


Output:



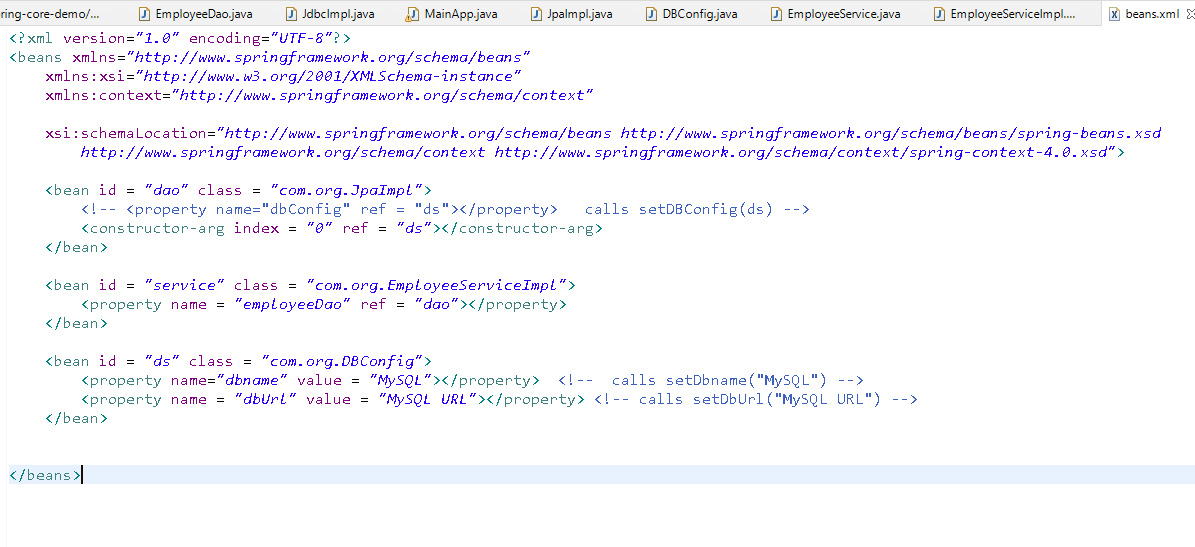
Exercise:

Create a Service class named EmployeeService and create a variable of type EmployeeDao and in main method you get the object of EmployeeService, the EmployeeService will have a method storeEmployee() which calls store() method of dao. From main you will call storeEmployee() that prints storeEmployee() method message as well as store() method message of EmployeeDao

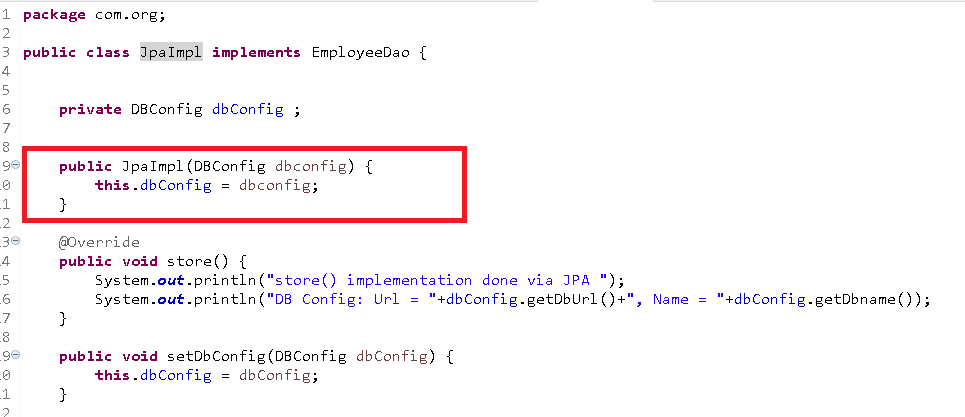


Setter & Constructor Injection

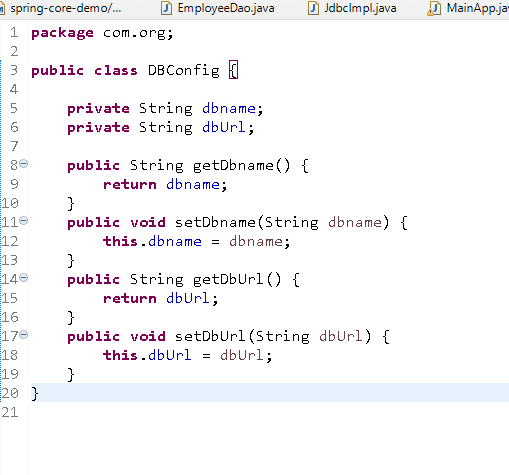
beans.xml



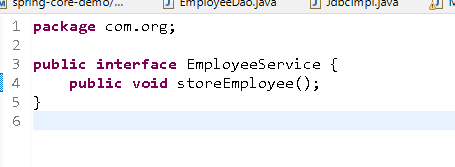
JpaImpl.java



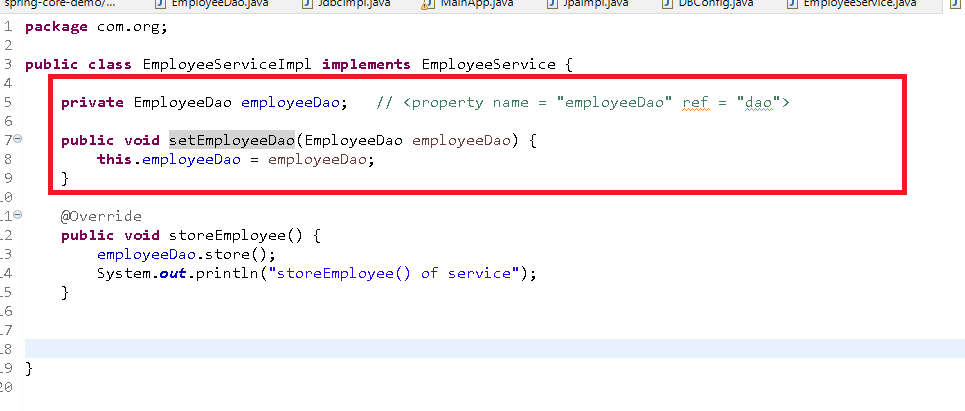
DBCOnfig.java



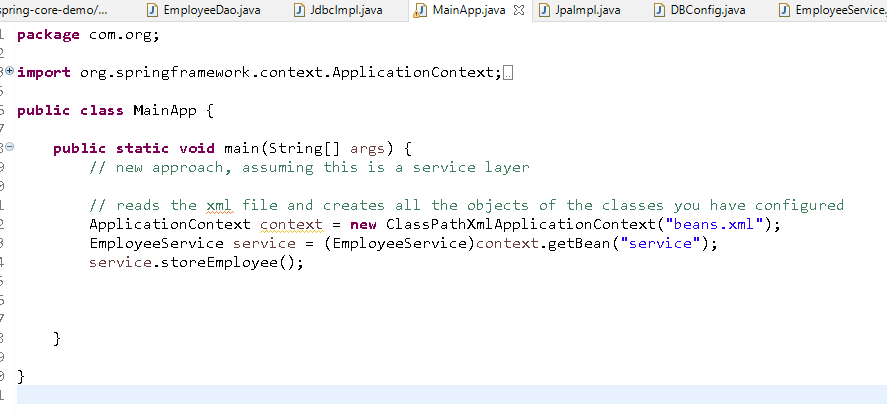
EmployeeService.java



EmployeeServiceImpl.java

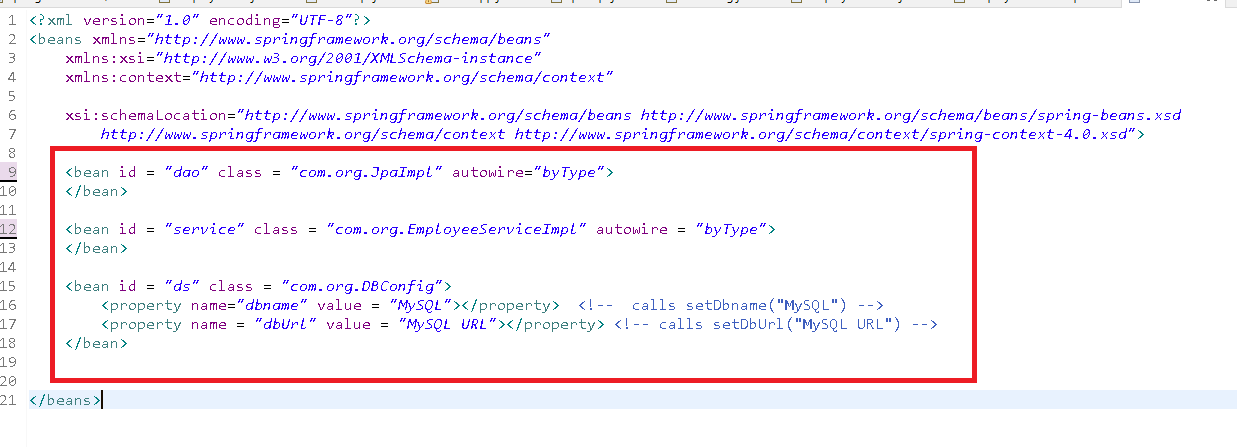


Main.java



Autowired:

It is another way of achieving the dependency injection where you can use auto-wire attribute in the bean so that you can avoid number of <property> or <constructor-arg> tags.



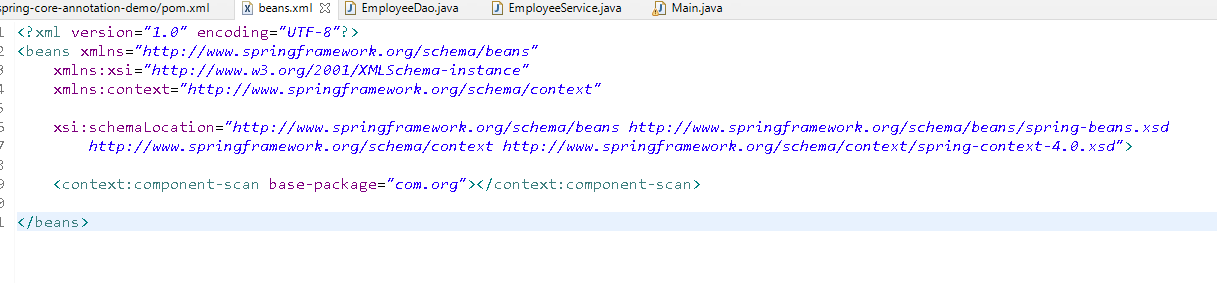
byType checks the datatype of the dependency and supplies to the dependent object

Annotation based configuration:

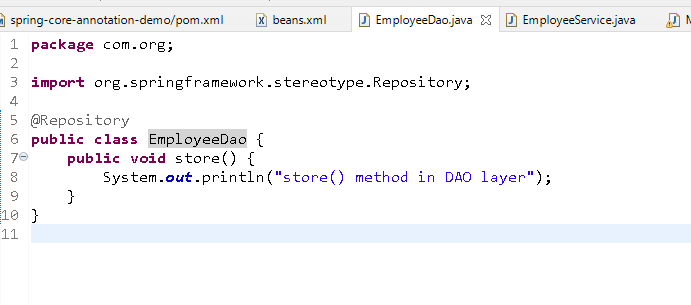
This simplifies writing the bean configuration where you can using annotations to enable spring to create objects and supply the dependencies.

Note: You can also declare the <bean> in the xml and supply them through @Autowired annotation

beans.xml

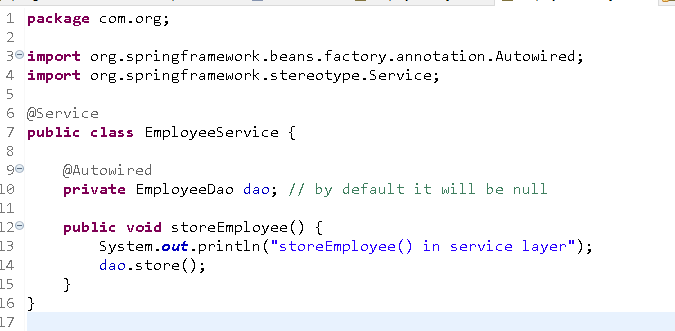


EmployeeDao.java



Note: @Repository is to recognise DAO classes

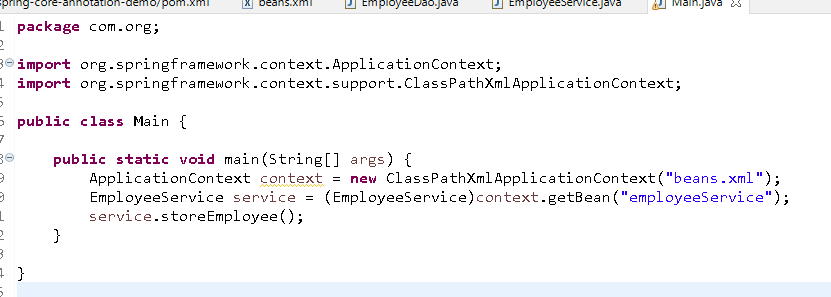
EmployeeService.java



Note: @Service is used to recognize the Service classes

Similarly you have @Controller to recognize classes and @RestController to recognize REST API classes

Main.java



Note: When objects are created via annotation, their id will be same as the class names except the first letter will start by lowercase.

Exercise: Same above example you have to modify

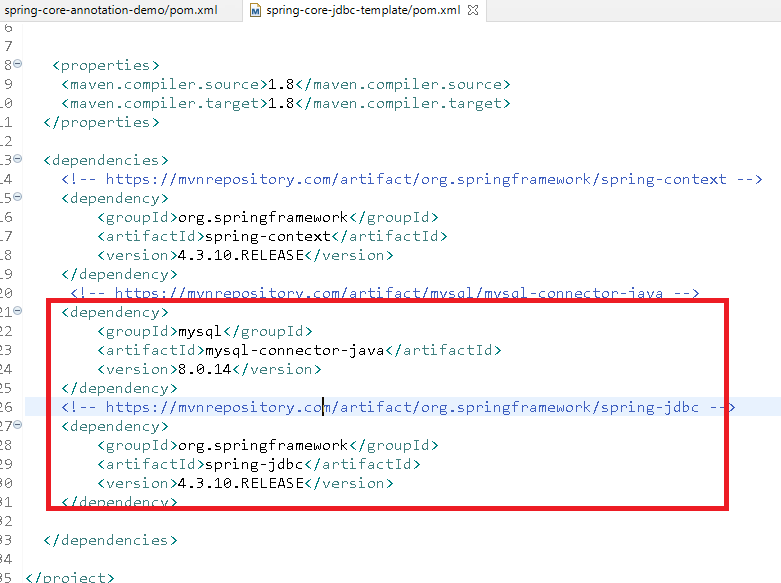
1. Configure DBConfig class in the XML File
2. Inject the DBConfig to the DAO via @Autowired

JdbcTemplate: It is an instance used to interact with database it depends on the DriverManagerDataSource instance you must inject it in the XML, and you can autowire JdbcTemplate in dao, it provides some methods to execute SQL Query.

We need two dependencies to work with Database & JdbcTemplate

1. MySQL connector
2. spring-jdbc

You must configure the xml file with database informations & inject the instance to the JdbcTemplate



beans.xml



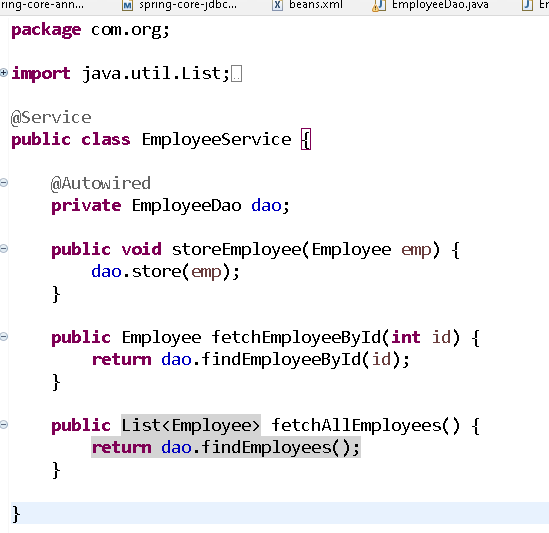
RowMapperImpl.java



EmployeeDao.java



EmployeeService.java



Main.java

