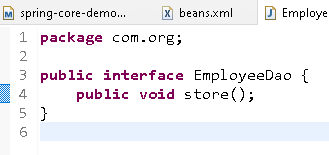
Spring Notes

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

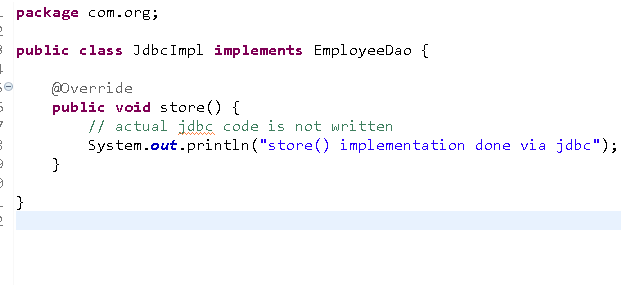
* Dependency Injection
* Singleton pattern
* Front controller pattern
* Proxy pattern
* Factory pattern

Old Approach

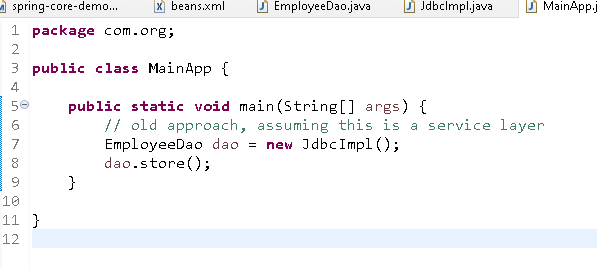
EmployeeDao.java



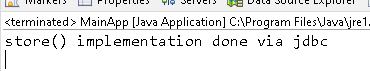
JdbcImpl.java



MainApp.java



Output:



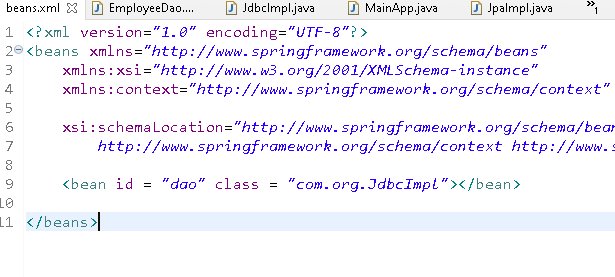
Suppose a new implementation you want to use then you must change the dependency code, which can be completely avoided through spring dependency injection feature

Dependency Injection: process of supplying the dependency to other objects

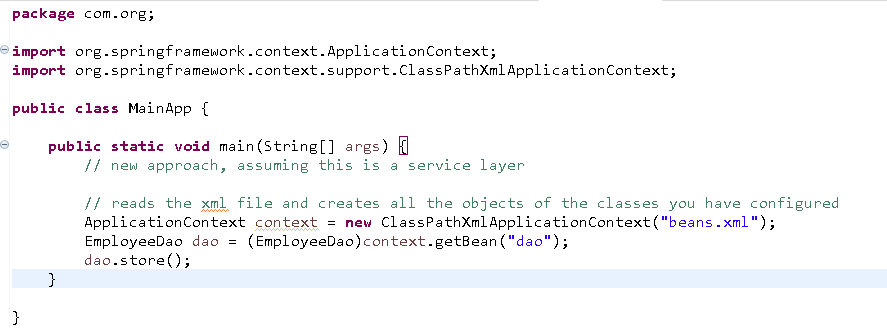
New Approach:

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

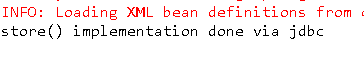
beans.xml



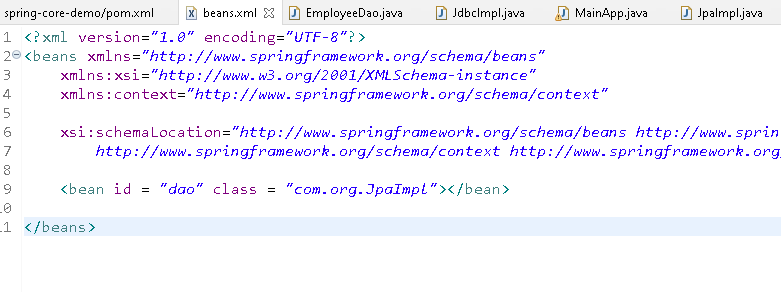
MainApp.java



Output:



Changing the xml to get JpaImpl



Output:



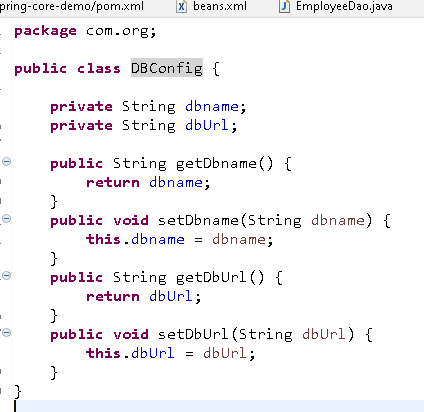
Dependency injection are of two types

1. setter injection
2. constructor injection

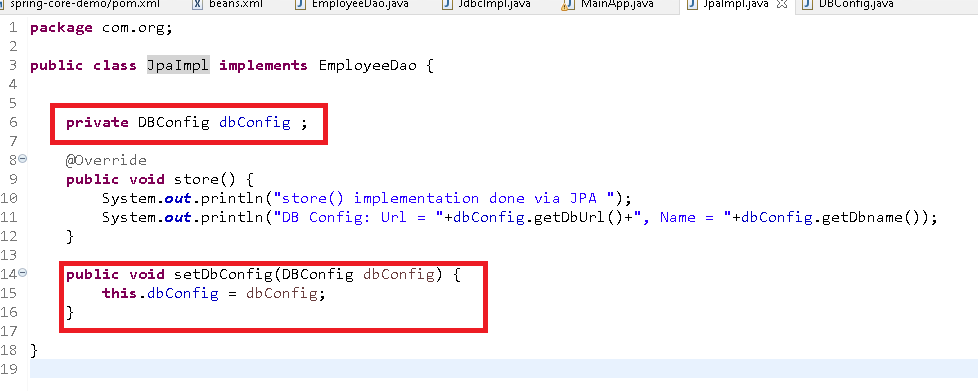
Setter injection means setter method will be called by passing a reference or a value

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

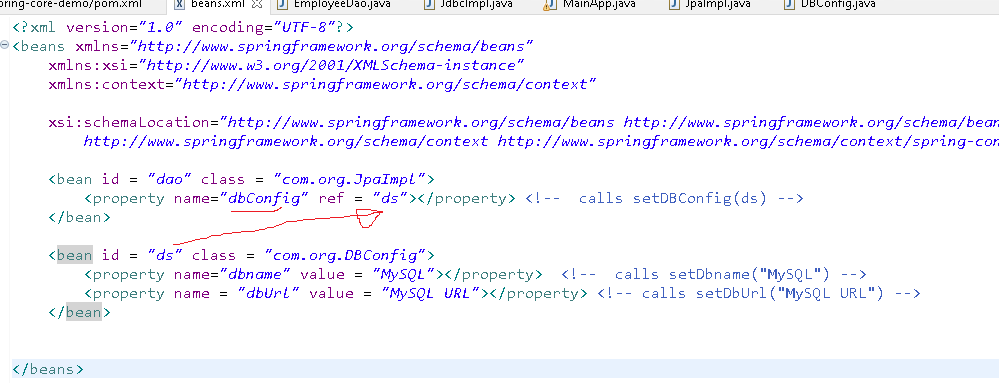
DBConfig.java



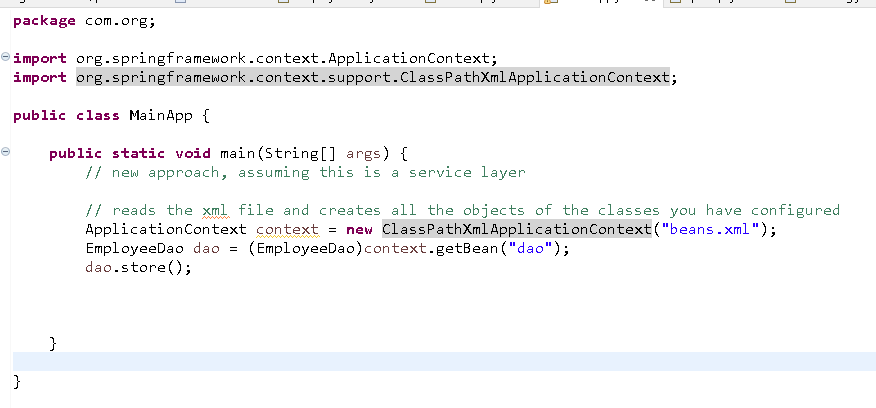
JpaImpl.java



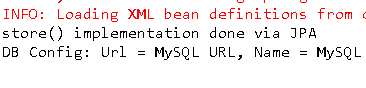
beans.xml



MainApp.java

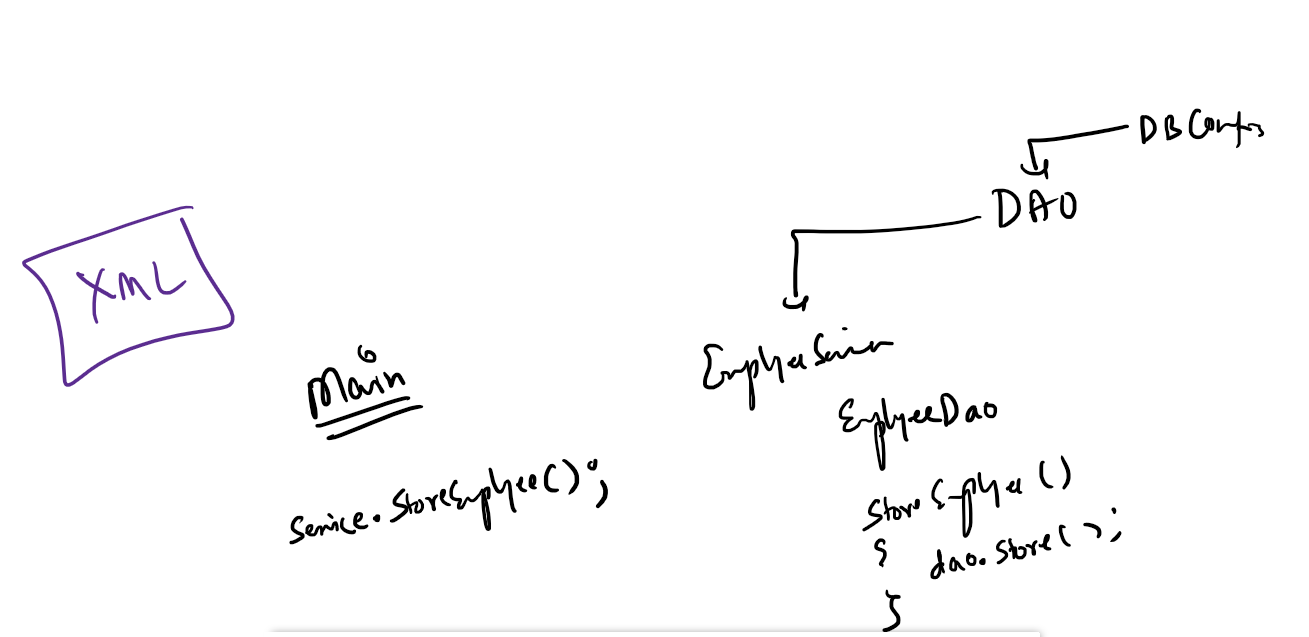


Output:



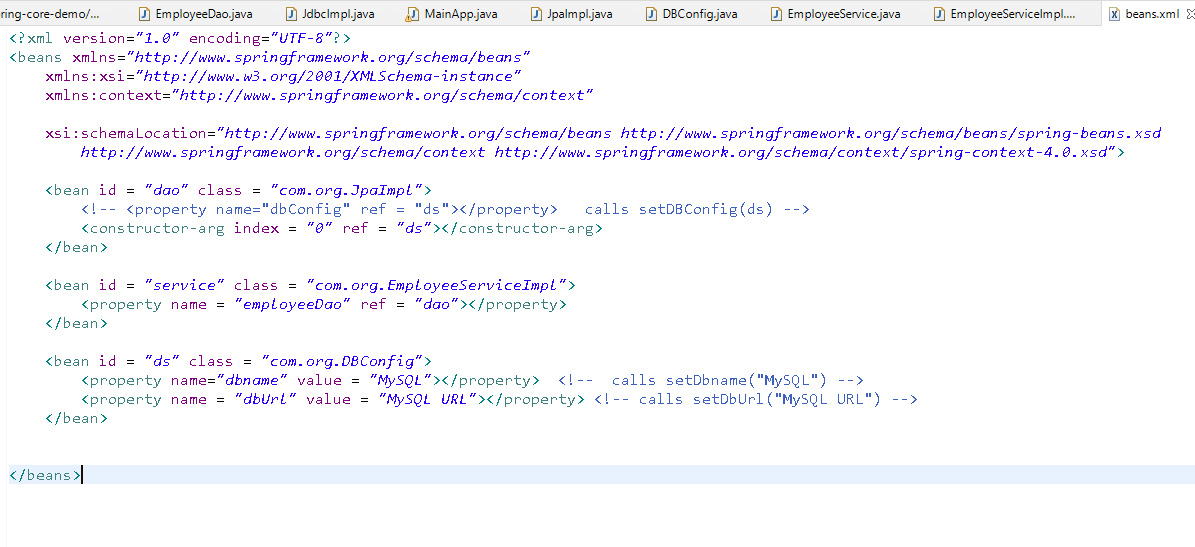
Exercise:

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

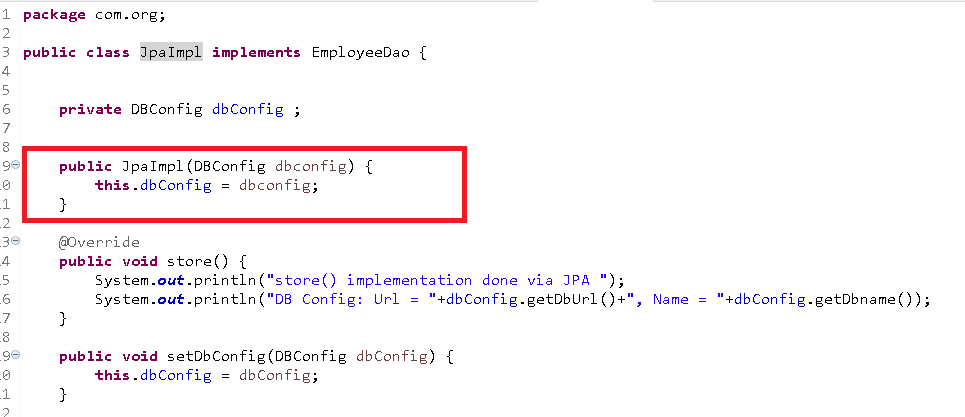


Setter & Constructor Injection

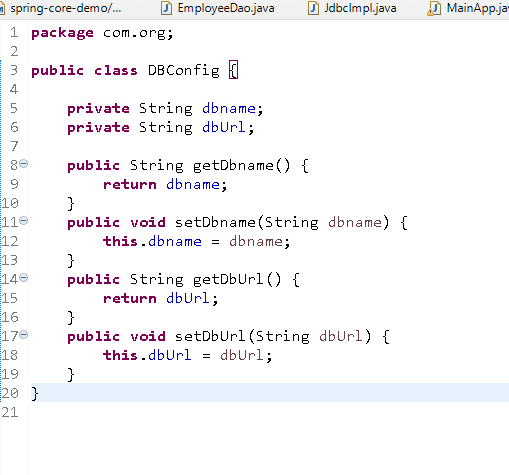
beans.xml



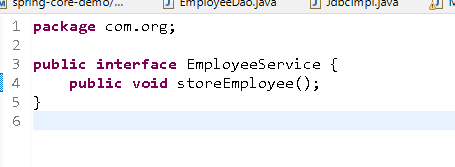
JpaImpl.java



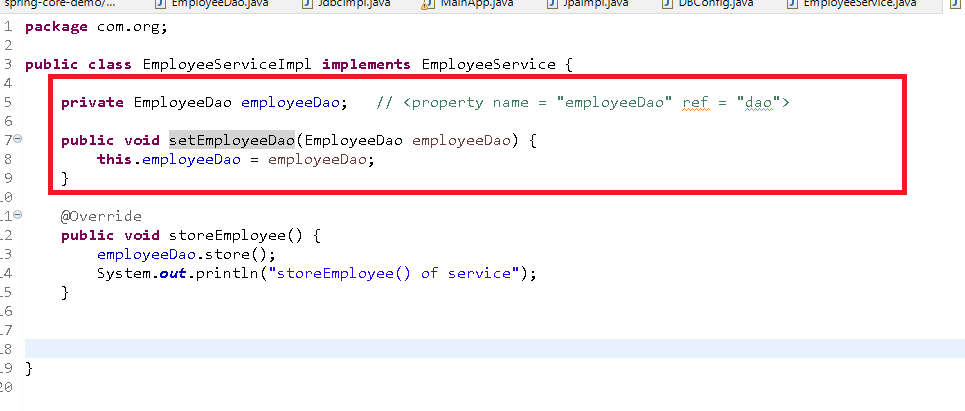
DBCOnfig.java



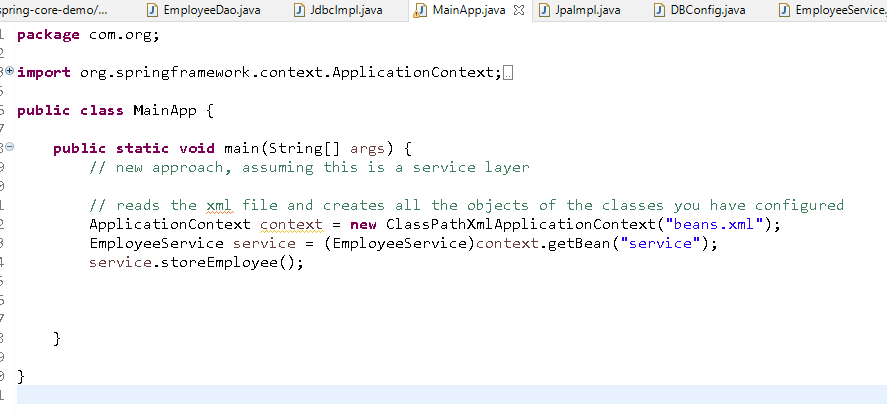
EmployeeService.java



EmployeeServiceImpl.java

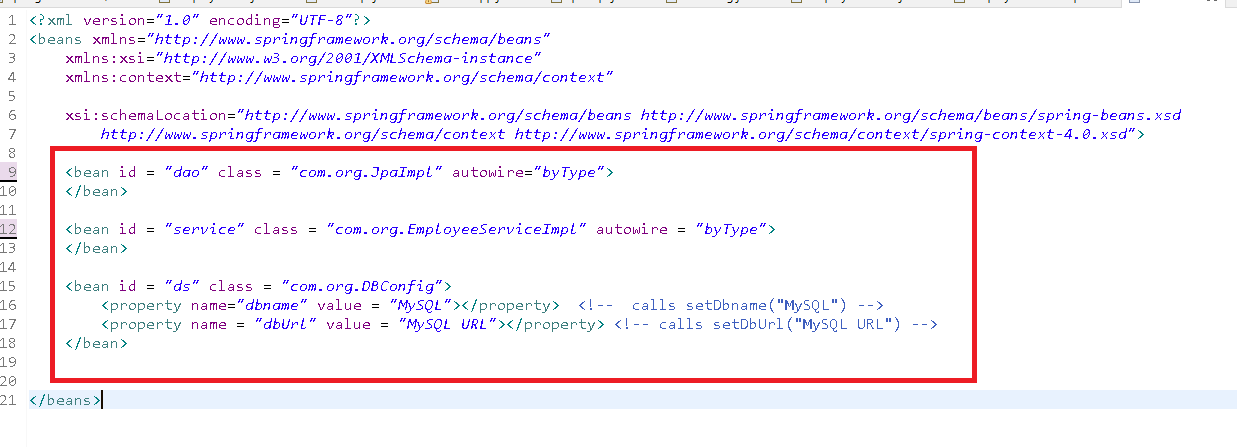


Main.java



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.



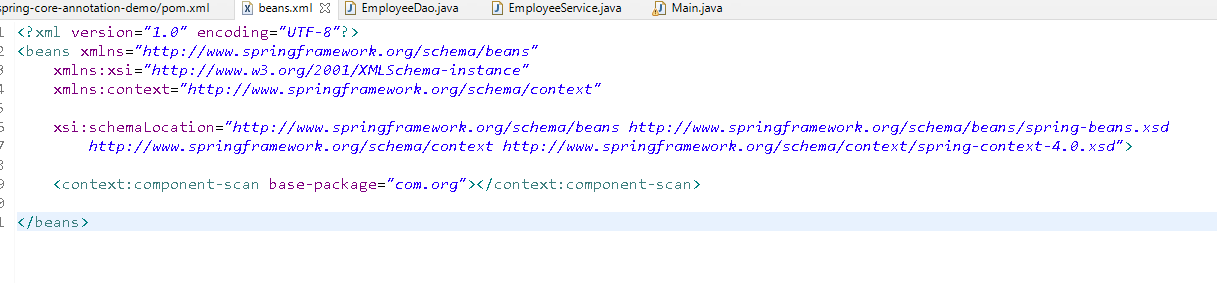
byType checks the datatype of the dependency and supplies to the dependent object

Annotation based configuration:

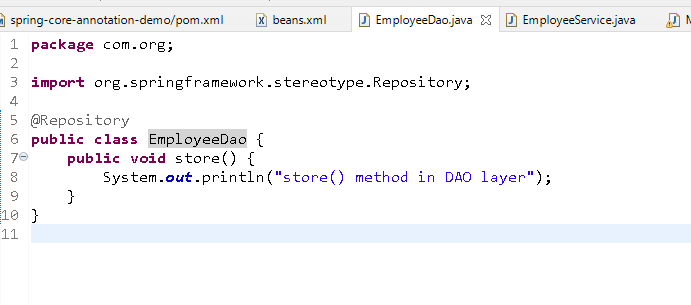
This simplifies writing the bean configuration where you can using annotations to enable spring to create objects and supply the dependencies.

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

beans.xml

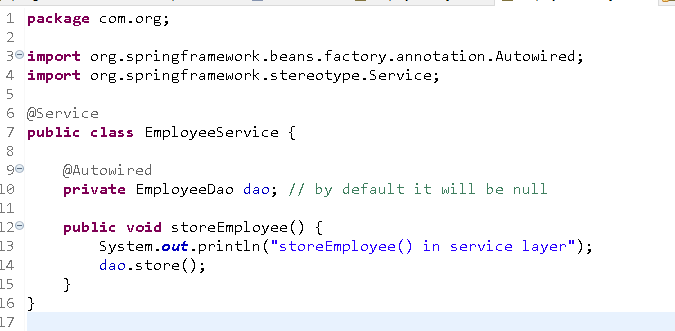


EmployeeDao.java



Note: @Repository is to recognise DAO classes

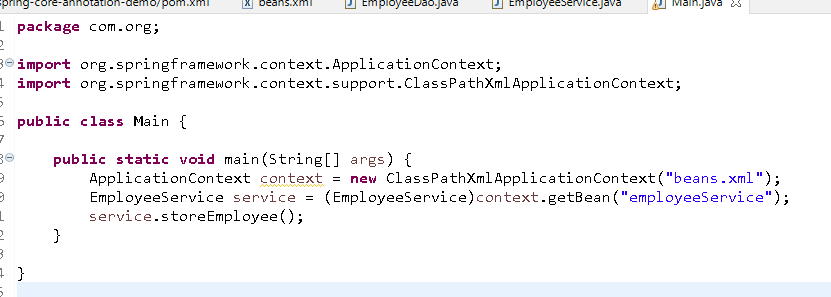
EmployeeService.java



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

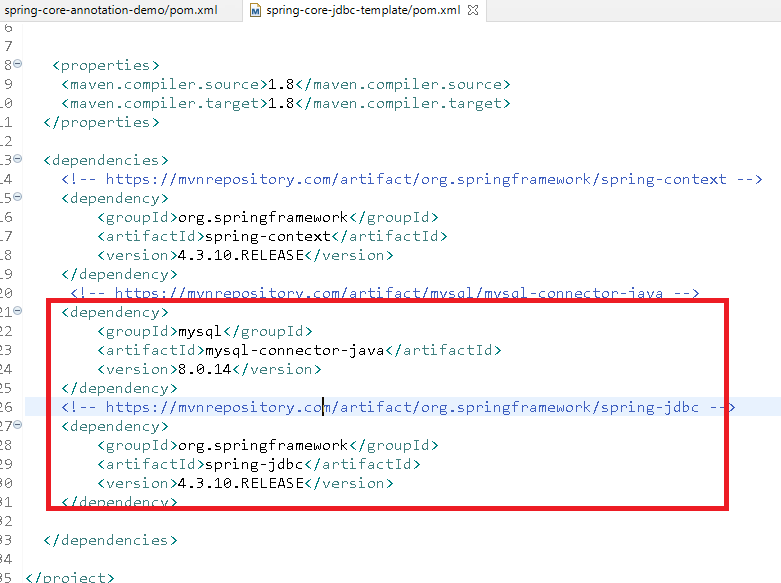
1. Configure DBConfig class in the XML File
2. Inject the DBConfig to the DAO via @Autowired

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



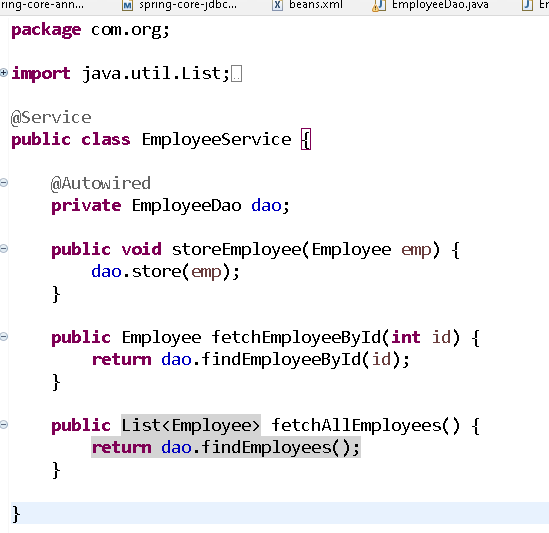
RowMapperImpl.java



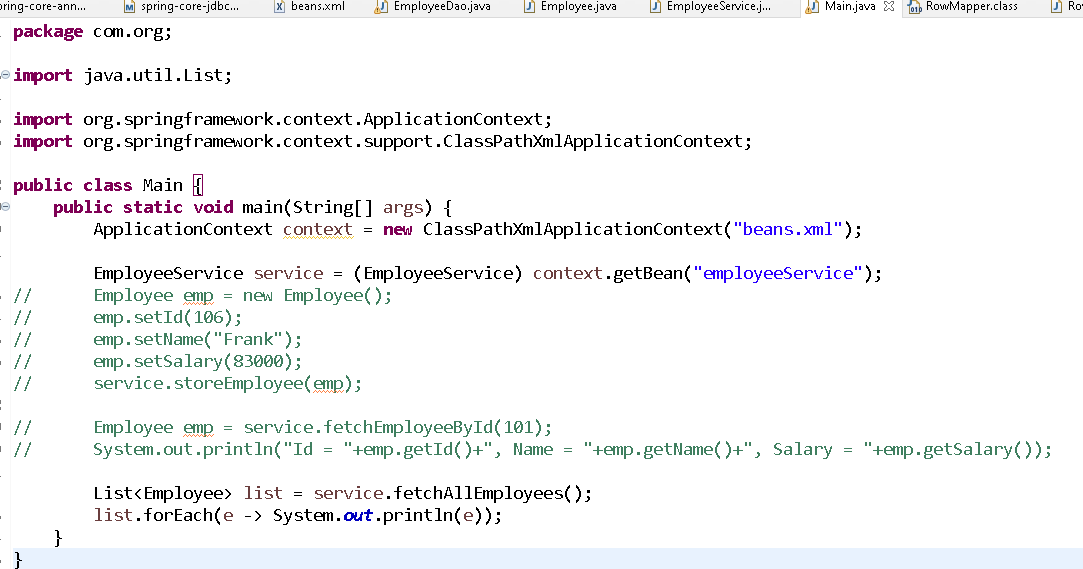
EmployeeDao.java



EmployeeService.java



Main.java



Spring MVC

It is used to develop web applications, it gives you Front Controller(DispatcherServlet) to take all the requests coming from the client and routes to the appropriate controller and also it takes care of instantiating the ApplicationContext for the web application.

In Spring MVC you will use 2 XML files:

1. web.xml: by default every web application will have this
2. spring xml: bean configurations

In Spring Core you wrote:

ApplicationContext ctx = new ClassPathXmlApplicationContext(..);

In Spring MVC you don’t need to create ApplicationContext

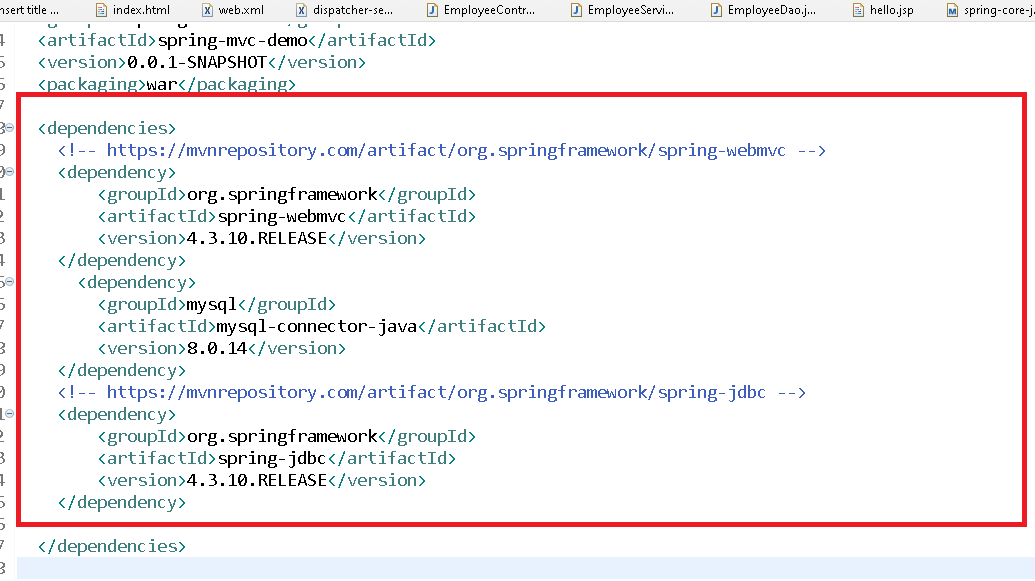
Because it is instantiated by DispatcherServlet and the loads the XML file using some naming approach present in web.xml

Steps:

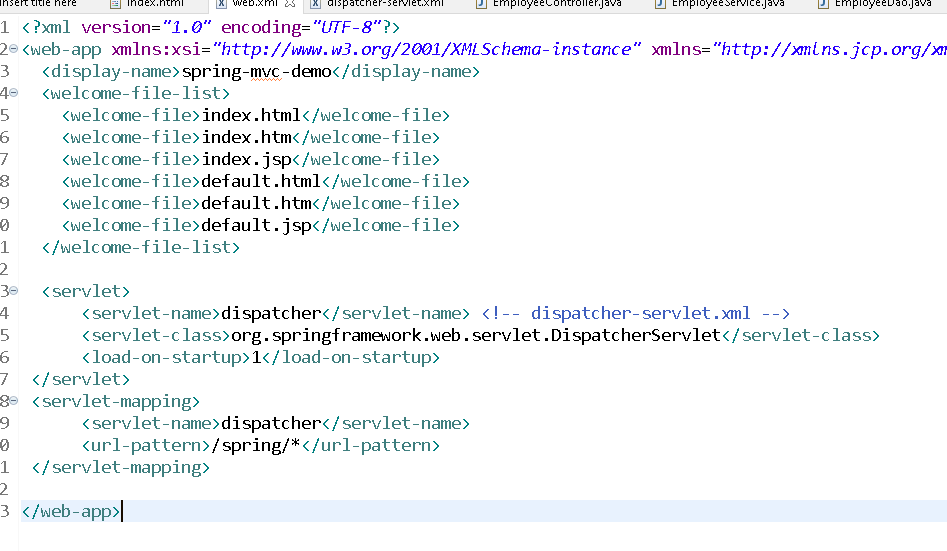
1. Create dynamic web project
2. Generate web.xml
3. Convert to Maven Project
4. Add spring web mvc dependencies in pom.xml
5. Configure Front controller in web.xml
6. Add spring configuration file

JdbcTemplate can be configuration in the dispatcher-servlet.xml, but you must add spring-jdbc and mysql-connector library in pom.xml

pom.xml



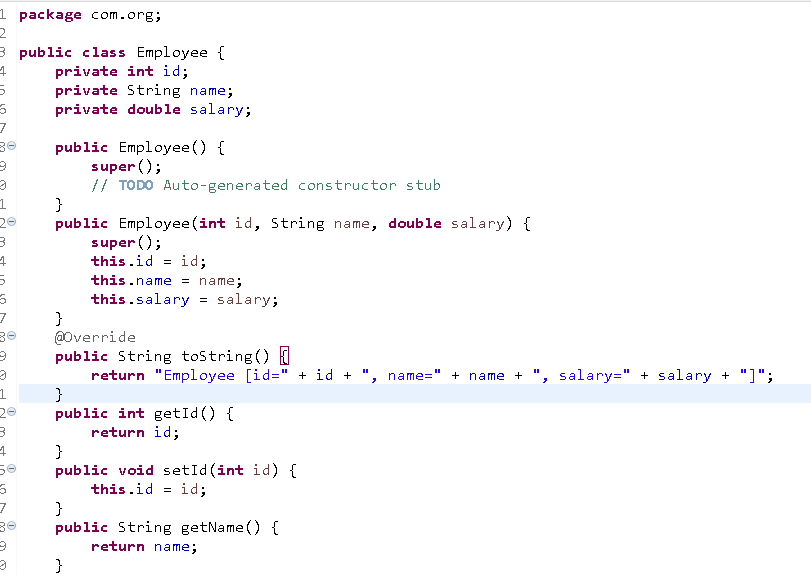
web.xml



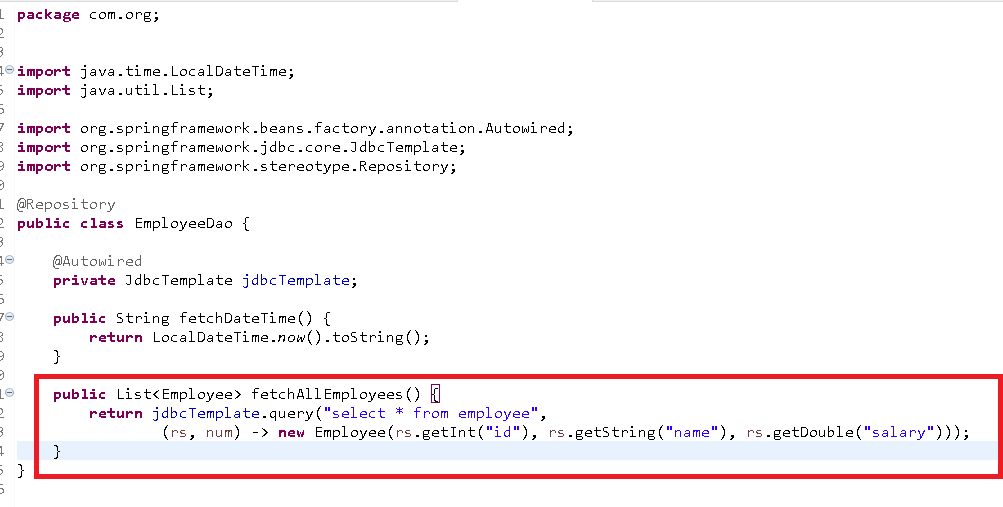
dispatcher-servlet.xml



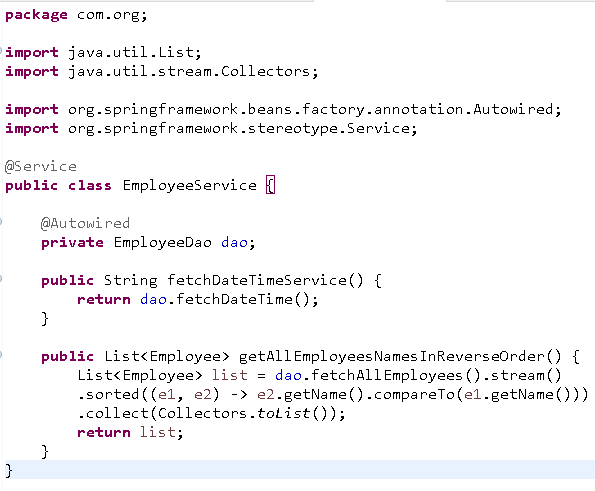
Employee.java



EmployeeDao.java



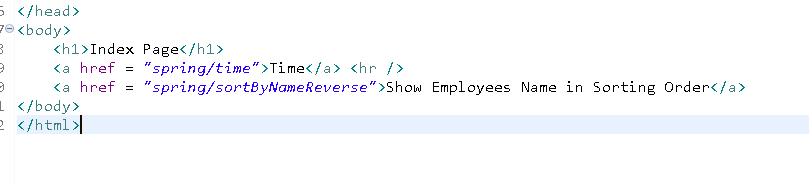
EmployeeService.java



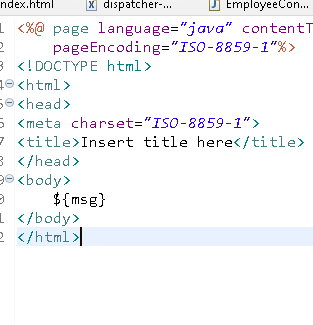
EmployeeController.java



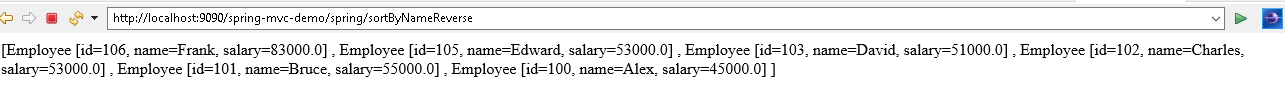
WebContent/index.html



WebContent/WEB-INF/pages/hello.jsp



Output:



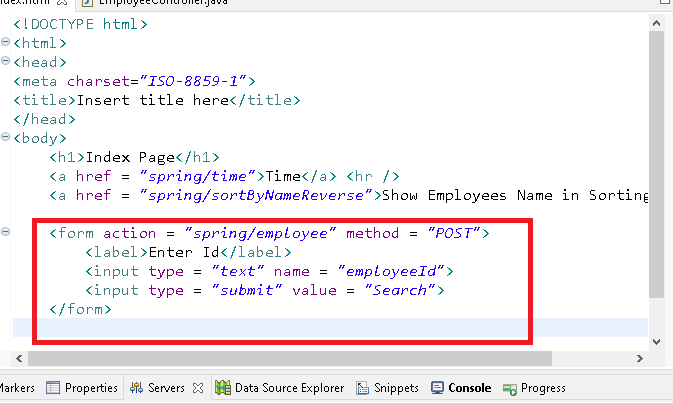
Exercise:

index.html, must have multiple links for sorting employees

1. Employees Name in Ascending Order
2. Employees Id in Ascending Order
3. Employees Id in Descending Order
4. Employees Salary in Ascending Order
5. Employees Salary in Descending Order

If you want employee information based on id, end user need to pass employee id and your web-application must return the employee matching to the id

index.html

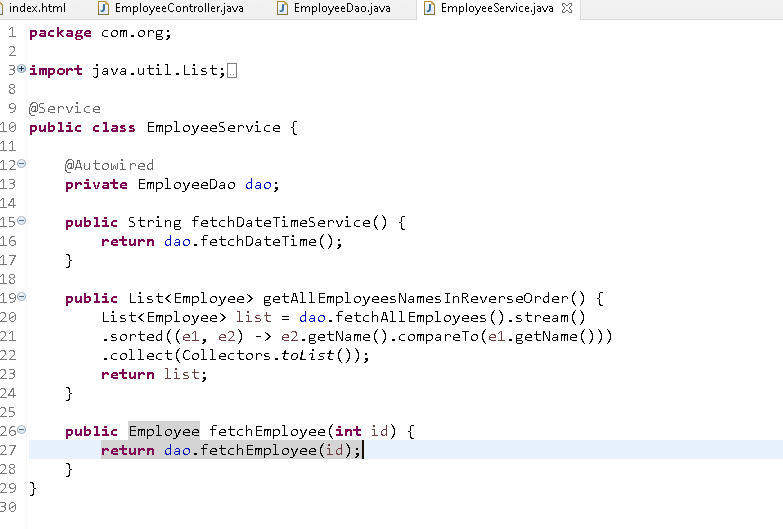


Controller has to extract the value associated with employeeId

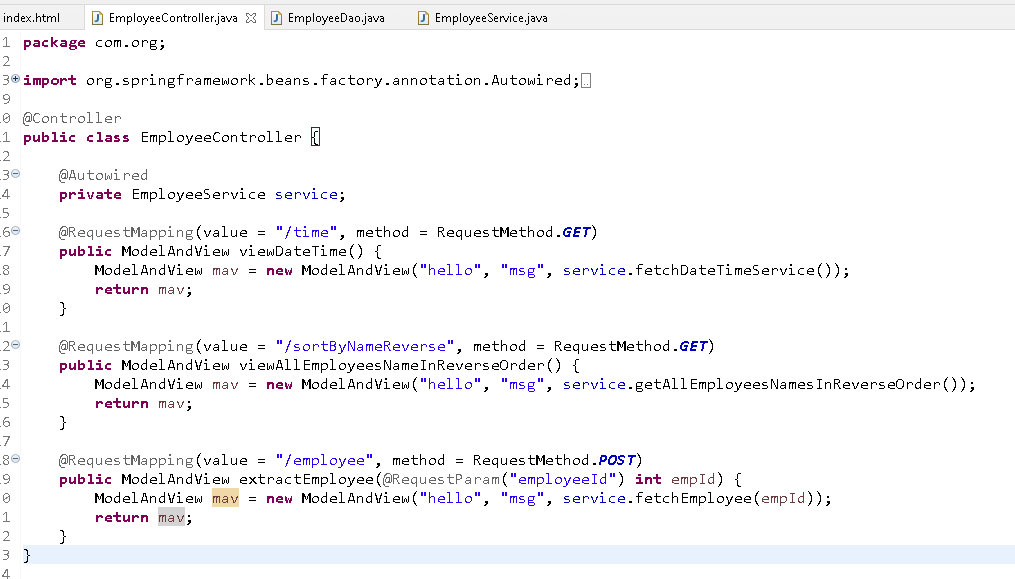
EmployeeDao



EmployeeService

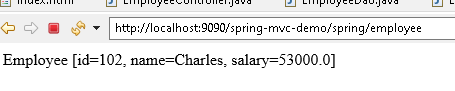


EmployeeController



Output:

After entering the id in the index.html you will see employee information



Few topics:

Internationalization

ModelMap: to maintain multiple keys to share the data to the view

@ControllerAdvice: It is like global exception handler internally uses AOP concept

Model Map:

When you have multiple data to be shown by the view then you need to maintain those multiple data in model map each data can have some unique key

It uses put method to add the key and the value

ModelMap map = new ModelMap();  
map.put(key1, value1);  
map.put(key2, value2);  
...

ModelAndView(“hello”, “mappedModel”, map);

The view can use ${mappedModel.key1} and ${mappedModel.key2}

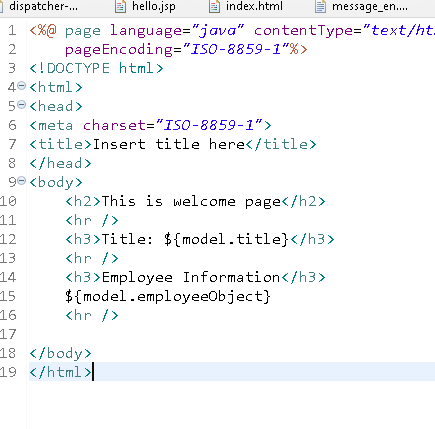
hello.jsp



EmployeeController.java



welcome.jsp



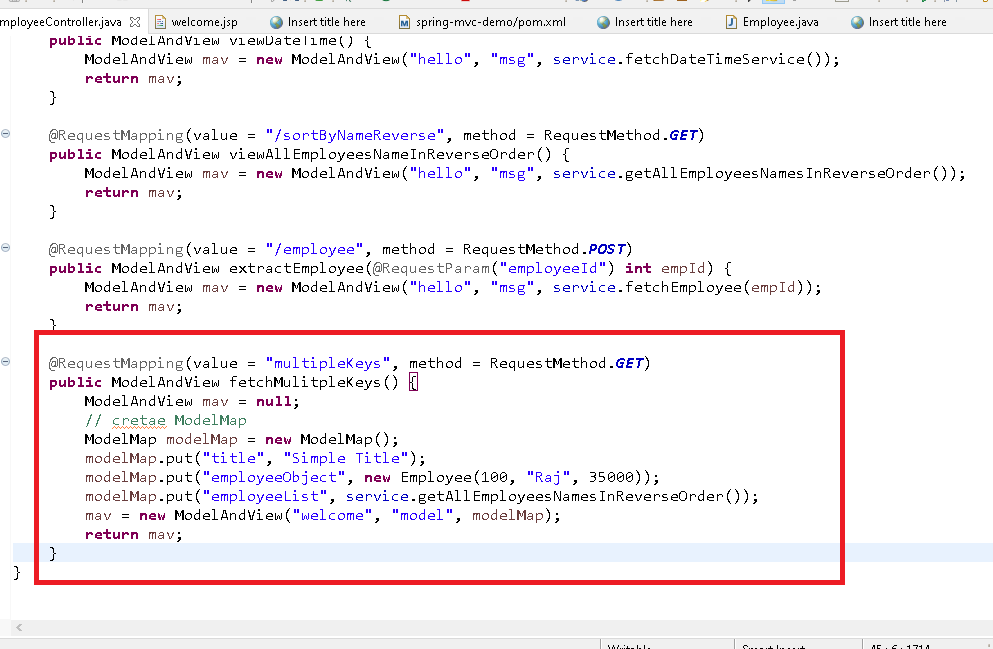
Output:



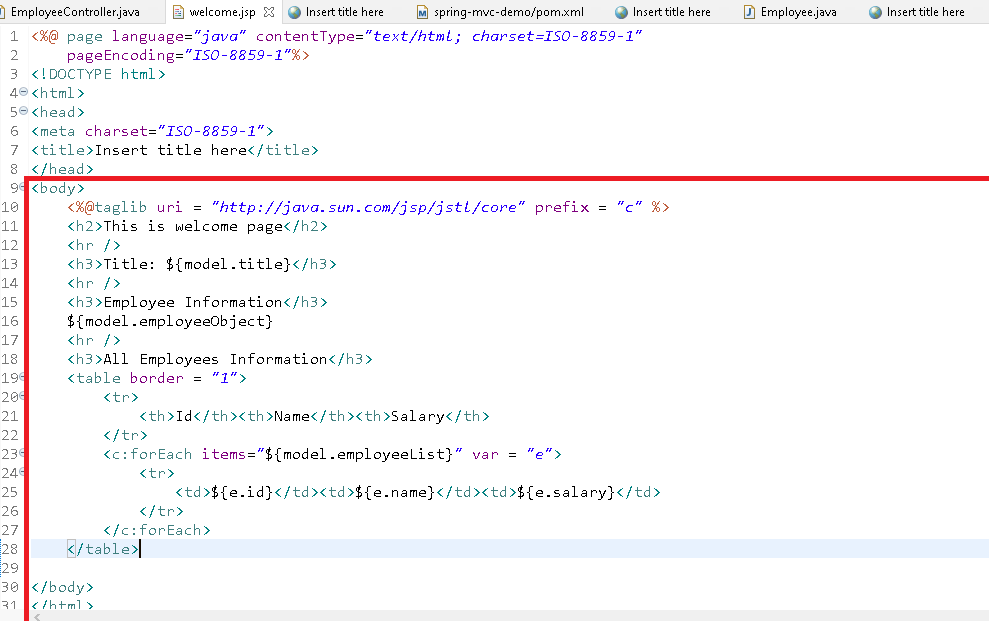
To display each item present in the collection you can user forEach tag of JSTL, for that you need to add jstl library



EmployeeControll.java



welcome.jsp



AOP:

Aspect Oriented Programming, it is mainly for cross cutting concerns, means executing some logics when a particular method is getting executed without actually calling that method.

join point: a particular business logic that needs cross cutting concerns

advice: a method that is executed when a join point is executed

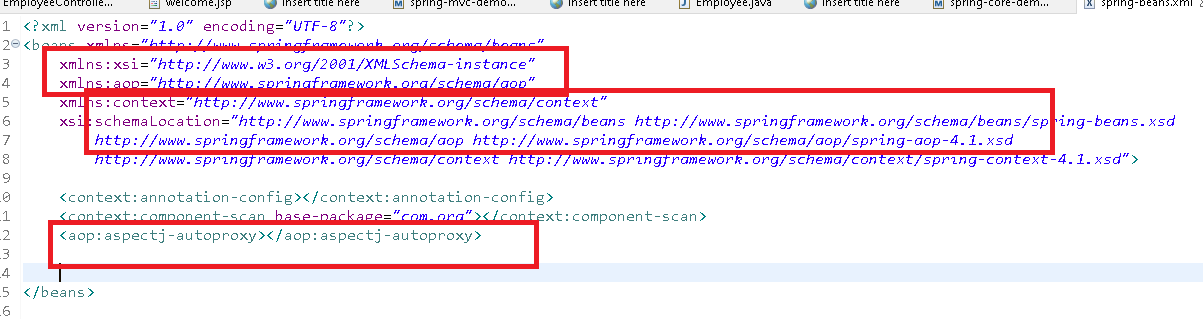
aspects: a class which will have advices

point cut: these are expressions that specify on which join point an advice has to be called.

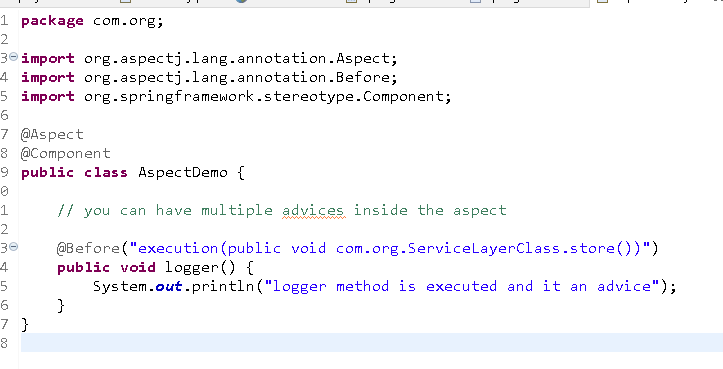
You must add aspecjweaver library to use AOP



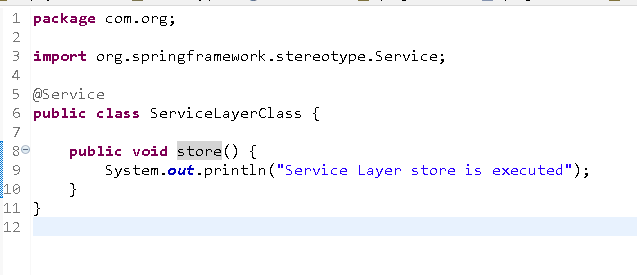
You need to have an xml with AOP namespace



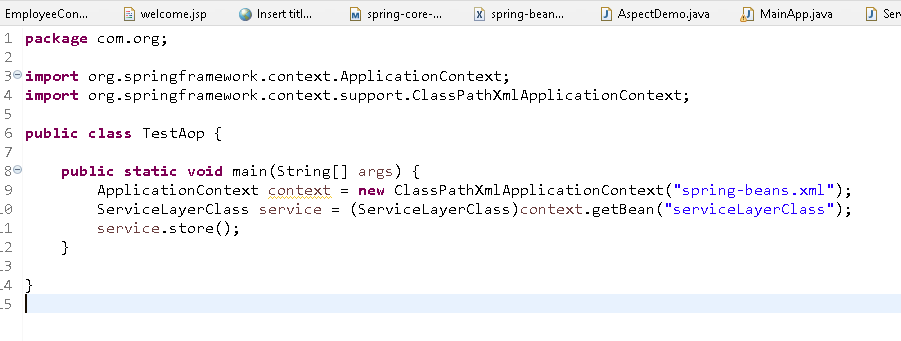
AspectDemo.java



ServiceLayerClass.java

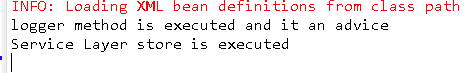


TestAop.java



Output:

You can see the logger executing without actually invoking inside store



You have other types of advices in AOP

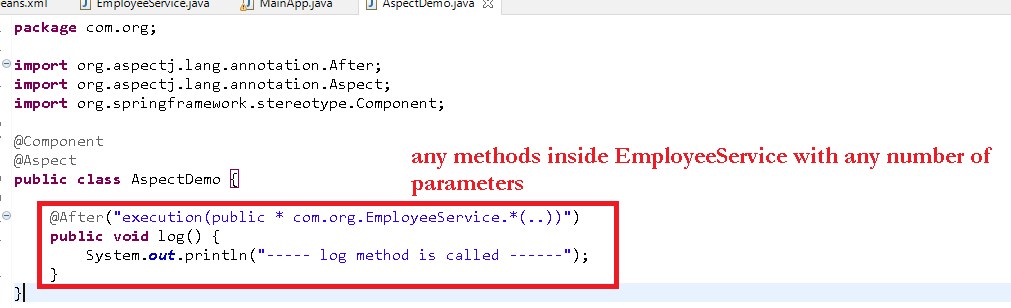
@After

@Around

@AfterThrowing

@AfterReturning

Point Cut expression



Some of the point cut expressions:

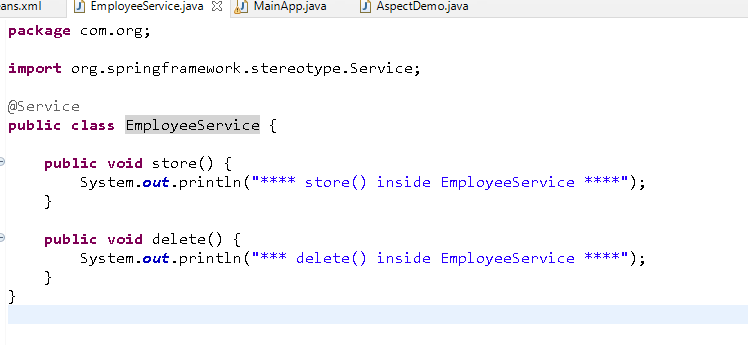
public void com.org.EmployeeService.\*(..): all methods inside EmployeeService having void return type

public void com.org.EmployeeService.set\*(..): all the methods starting with set followed any name and any parameter to the set, like setId(), setName()..

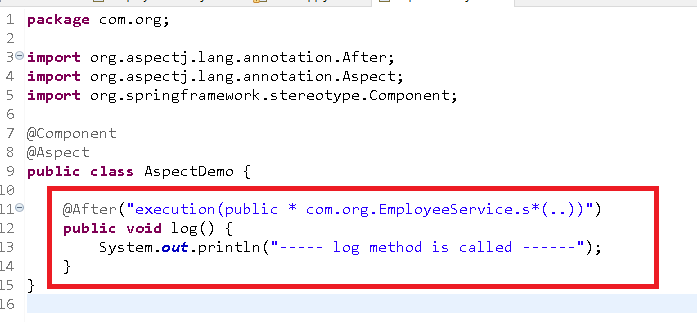
beans.xml



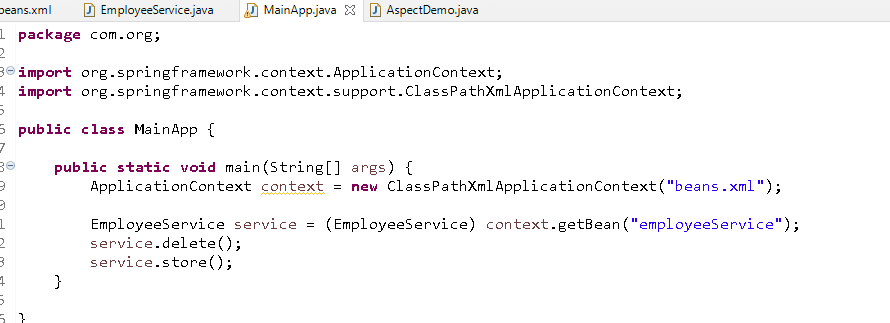
EmployeeService



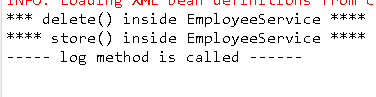
AspectDemo



MainApp



Output:



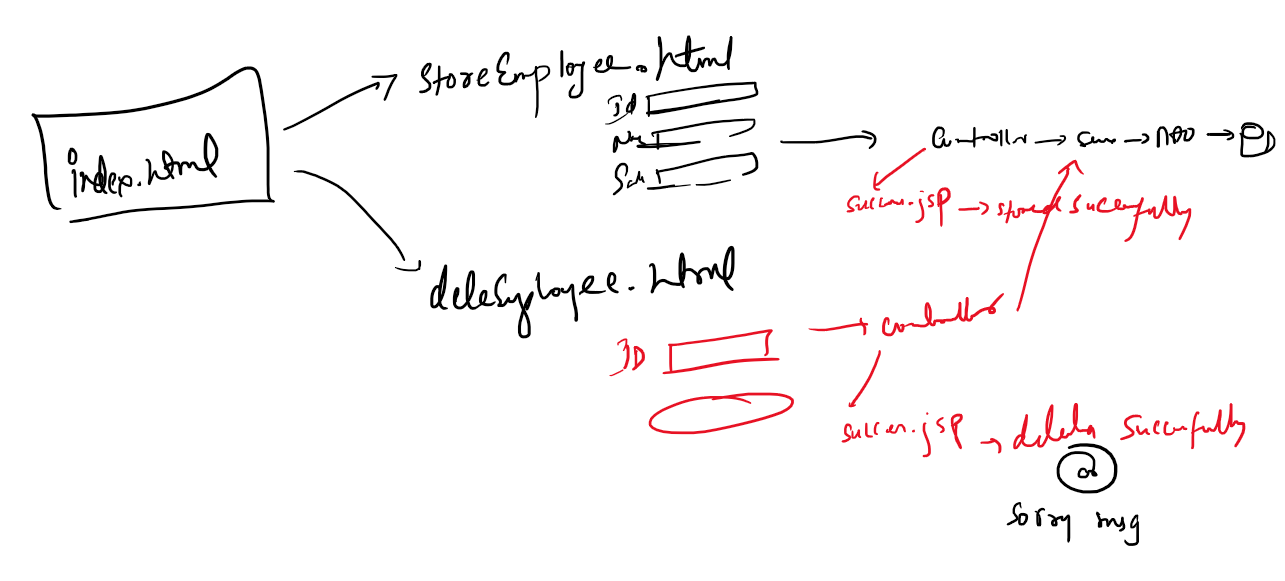
Add some more methods inside the EmployeeService with some parameters as well like

getEmployee(), getAllEmployees(), updateEmployee(), ....

Try to apply the point cut expression on each join point

1. expression to call only the methods taking int parameter
2. expression to call only the methods taking 2 parameter like int, String parameter
3. expression to call only the methods taking 2 parameters int, any parameter
4. expression to call only the methods having String return type

Spring MVC Assignment



Using ORM framework in Spring:

ORM stands for Object Relational Mapping which directly maps the java objects to the database, it simplifies performing the CRUD operation without writing any SQL statements or minimum SQL statements.

We have different implementations for ORM like:

JPA, Hibernate, TopLink, iBatis ....

Spring provides a template for different implementations of ORM like

JpaTemplate, HibernateTemplate, ...

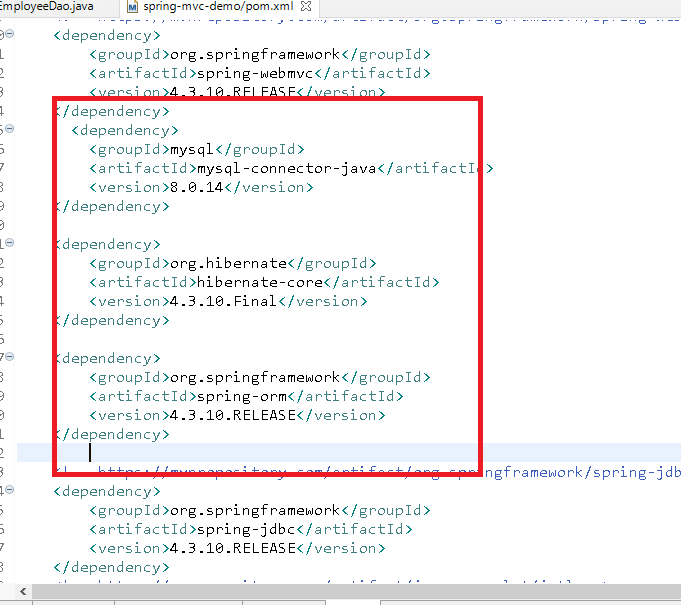
You need HibernateTemplate object which has to be configured in the xml file.

Dependencies required

1. hibernate-core library
2. spring-orm library
3. mysql-connector library



pom.xml



dispatcher-servlet.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd"*>

<context:component-scan base-package=*"com.org"*></context:component-scan>

<mvc:annotation-driven/>

<tx:annotation-driven/>

<bean id=*"viewResolver"* class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"*>

<value>/WEB-INF/pages/</value>

</property>

<property name=*"suffix"*>

<value>.jsp</value>

</property>

</bean>

<bean id=*"dataSourceBean"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name=*"driverClassName"* value=*"com.mysql.cj.jdbc.Driver"*></property>

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/cts\_db?useSSL=false"*></property>

<property name=*"username"* value=*"root"*></property>

<property name=*"password"* value=*"root"*></property>

</bean>

<bean id=*"sf"* class=*"org.springframework.orm.hibernate4.LocalSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSourceBean"*></property>

<property name=*"annotatedClasses"*>

<list>

<value>com.org.Employee</value> <!-- we had to mention only those classes having @Entity -->

</list>

</property>

<property name = *"hibernateProperties"*>

<props>

<prop key = *"hibernate.dialect"*>org.hibernate.dialect.MySQL5Dialect</prop>

</props>

</property>

</bean>

<bean id = *"hibernateTemplateBean"* class = *"org.springframework.orm.hibernate4.HibernateTemplate"*>

<property name = *"sessionFactory"* ref = *"sf"*></property>

</bean>

<bean id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>

<property name=*"dataSource"* ref=*"dataSourceBean"*></property>

</bean>

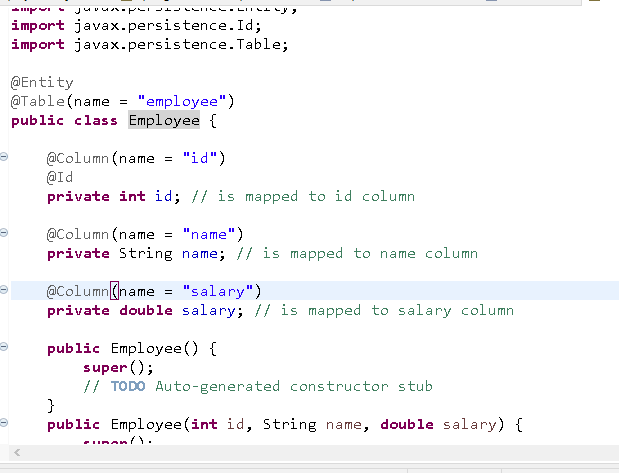
<bean id=*"transactionManager"* class=*"org.springframework.orm.hibernate4.HibernateTransactionManager"*>

<property name=*"sessionFactory"* ref=*"sf"*></property>

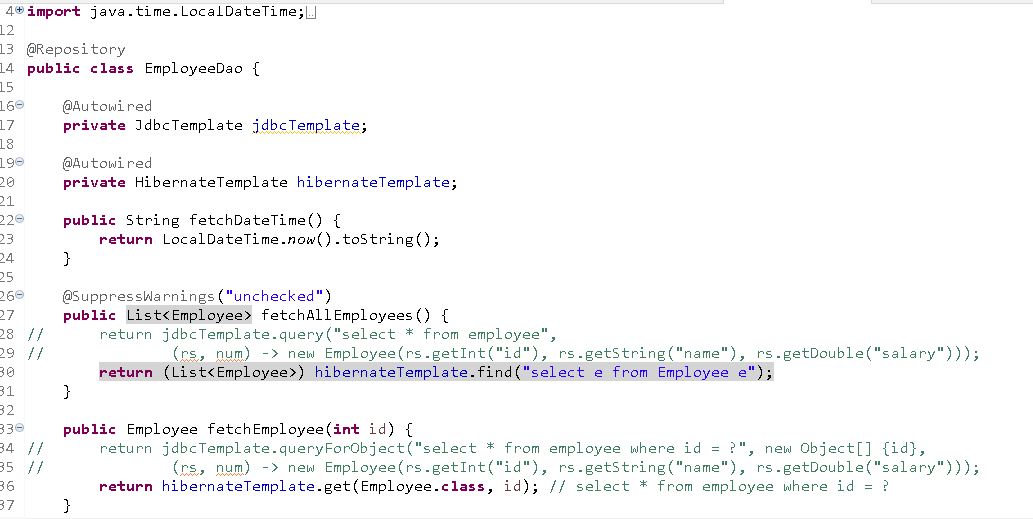
</bean>

</beans>

Employee.java



EmployeeDao.java



You can use save(employee), delete(employee) and so on.