Spring Framework

Spring is a Java Framework is used to develop variety applications like desktop, web, enterprise, mobile, cloud and etc.

Framework: It is like a semi project which will have all the common features required in every application, the common feature every application needs are:

1. Type Conversion
2. Exception Handling
3. Security
4. Transaction
5. Design Patterns
6. Internationalization

Framework enables developers to quickly create the complex applications, any body can understand the application if it is using framework as it will be structured and follows lot of design principles.

Spring Framework has many modules

1. Spring Core: Base module
2. Spring MVC: Web/REST based applications
3. Spring JDBC: Interact with the database
4. Spring AOP: Aspect Oriented Programming
5. Spring Boot
6. Spring Microservices

Spring Core:

It mainly concentrates on the dependency injection feature

Dependency Injection: It is a process of supplying the dependent object to other objects, i.e., creating objects & supplying to other objects

In spring framework you will use an XML configuration where you write all the bean configurations, based on the bean configurations the spring container creates the object

beans.xml

<bean id = “b1” class = “com.A”> </bean>  
<bean id = “b2” class = “com.B”> </bean>  
<bean id = “b3” class = “com.C”> </bean>

Above configuration allows spring to create 3 objects i.e., A, B & C

beans.xml

<bean id = “d” class = “com.DAO1”> </bean>  
<bean id = “s” class = “com.Service1”>  
 <property name = “dao” ref = “d”>

</bean>

Assume Service1 has a property dao of a DAO type (interface type) and DAO1 is implementing the DAO, then the above configuration supplies the DAO1 to the dao property of Service1.

This process of Dependency Injection is also called as IoC (Inversion of Control)

IoC: Inversion of Control is the process of reversing the control of object creation i.e., container is creating the object and passing it to other objects

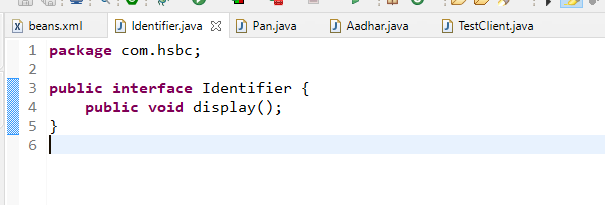
Note: You need to use maven project because you need spring libraries some of the important libraries are

* spring context
* spring webmvc
* spring jdbc

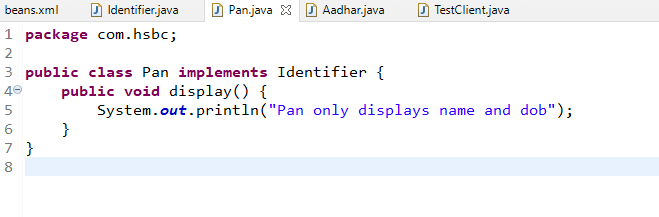
pom.xml



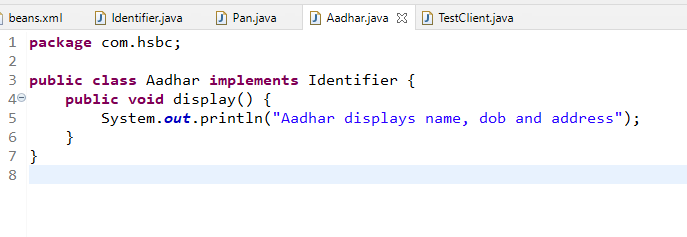
Identifier.java



Pan.java



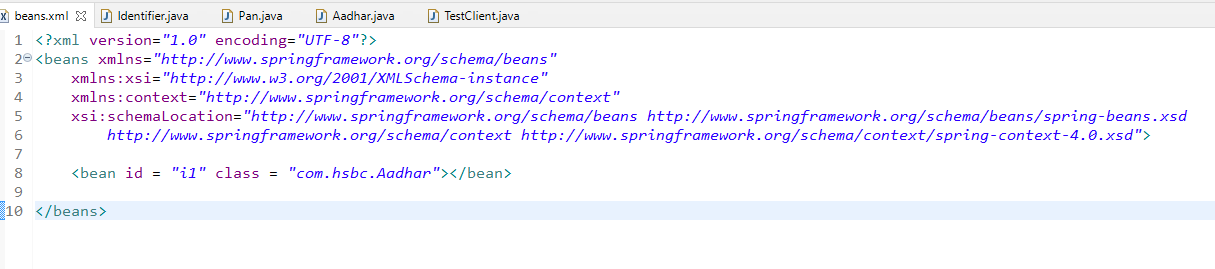
Aadhar.java



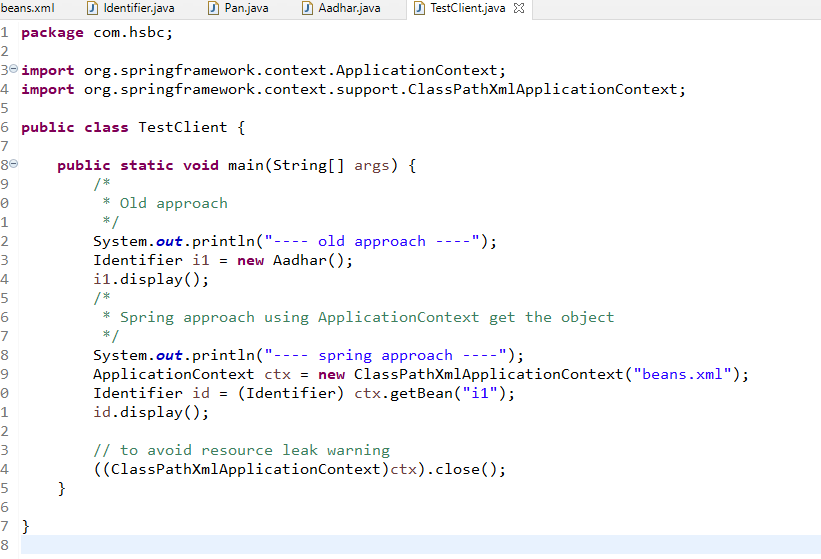
Bean configuration file

You need to keep the xml file in the classpath i.e., src folder

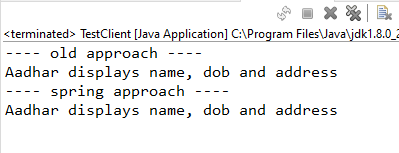
beans.xml



TestClient.java



Output:

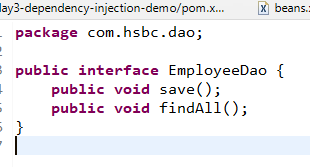


Types of Dependency Injection

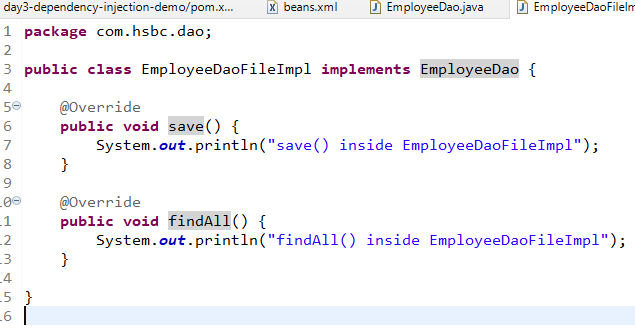
We have 2 types

1. Setter Injection <property>
2. Constructor Injection <constructor-arg>

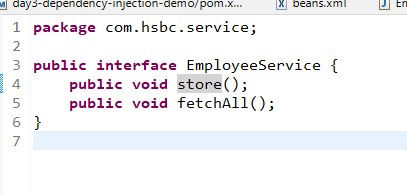
EmployeeDao.java



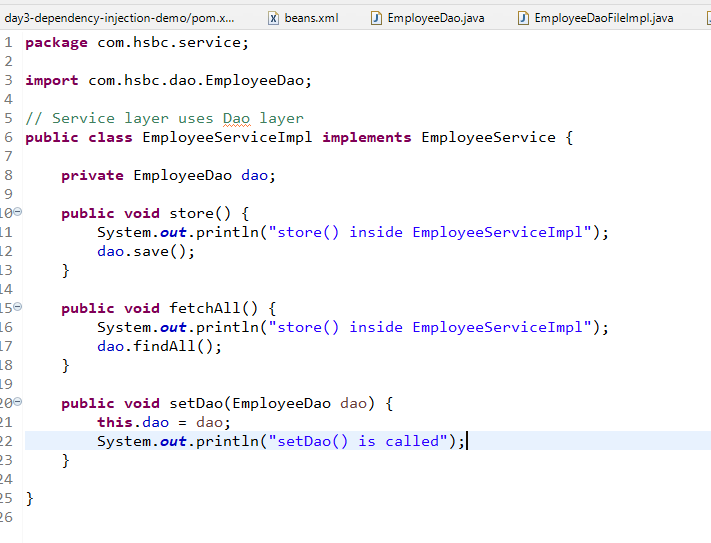
EmployeeDaoFileImpl.java



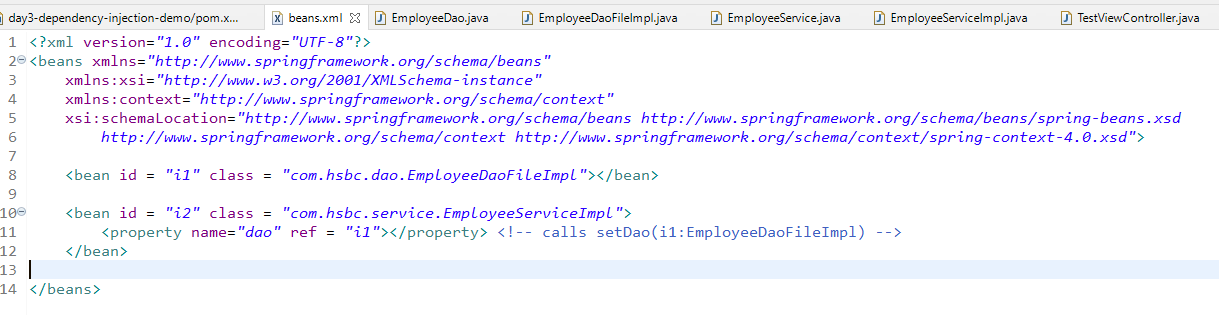
EmployeeService.java



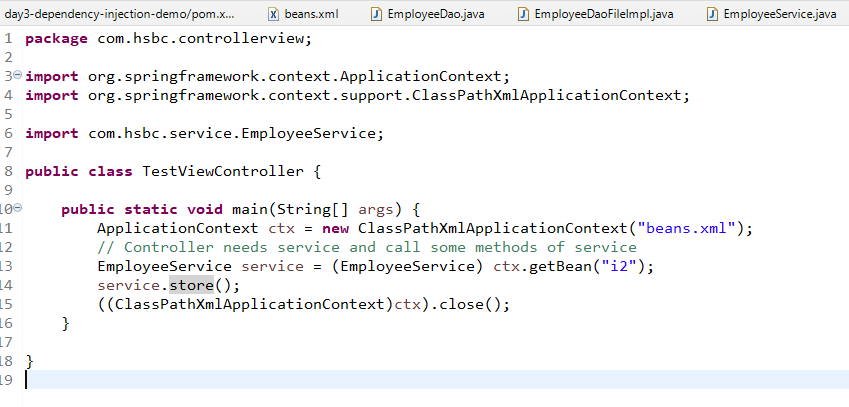
EmployeeServiceImpl.java



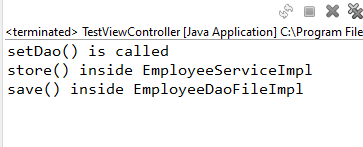
beans.xml



TestViewController.java



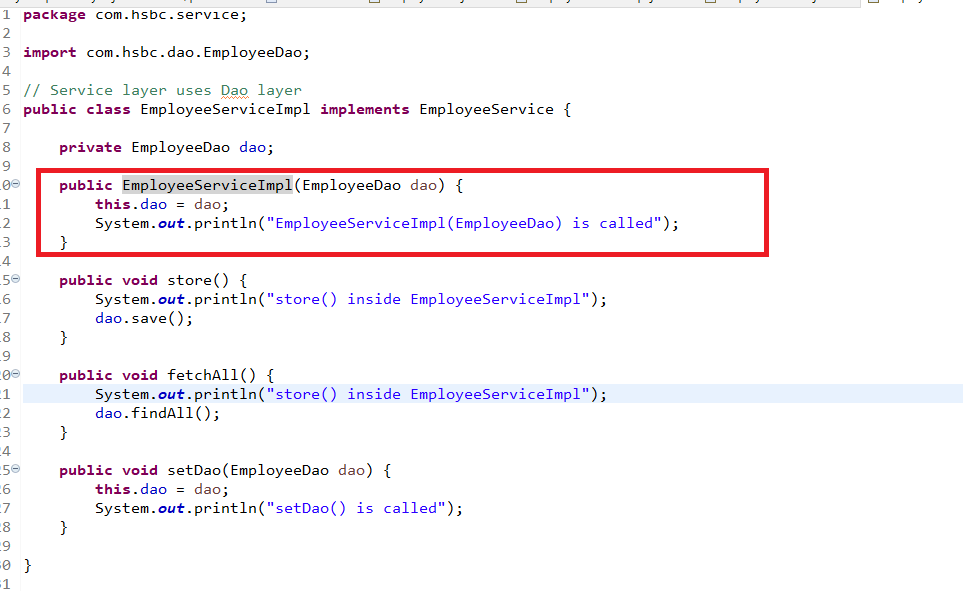
Output:



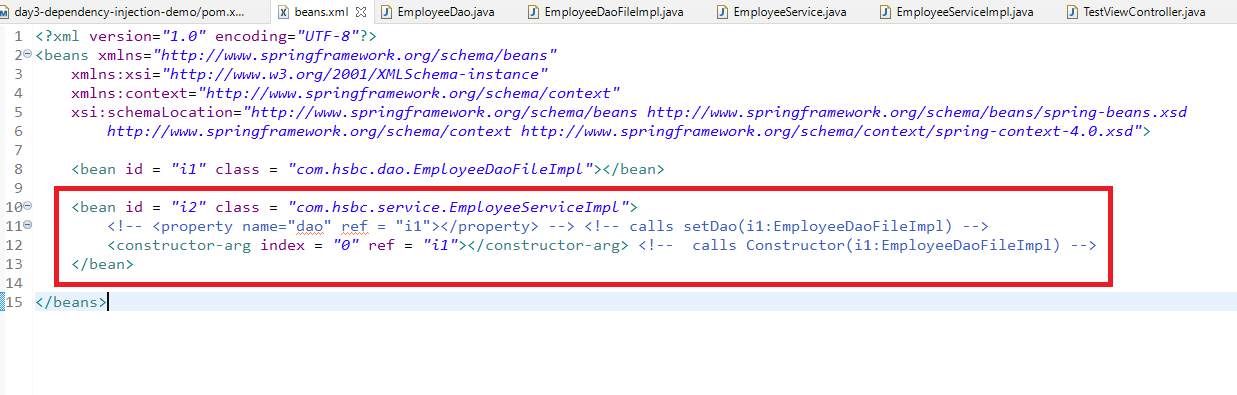
Constructor Injection

It is done when you want spring to pass the dependencies through the constructor parameter, the tag we need to use is <constructor-arg> and index attribute specifies the parameter position starting from 0

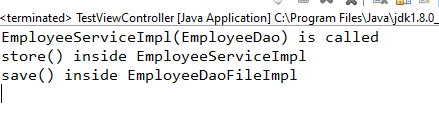
EmployeeServiceImpl



beans.xml



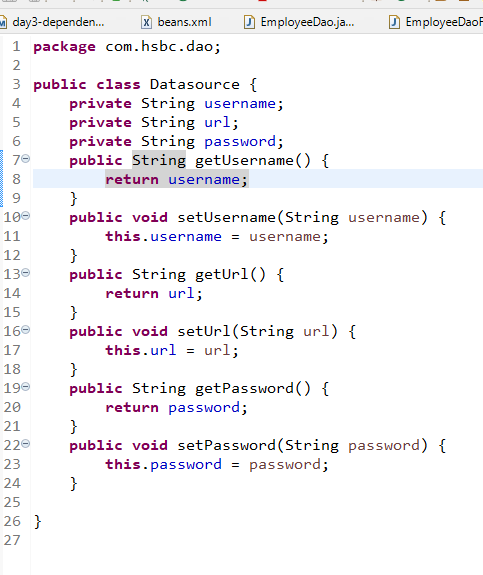
Output:



Note: when your bean class has more than one properties then you can use one or more <propery> or <constructor-arg> tags depending on the class structure

Passing values instead of objects also possible in dependency injection

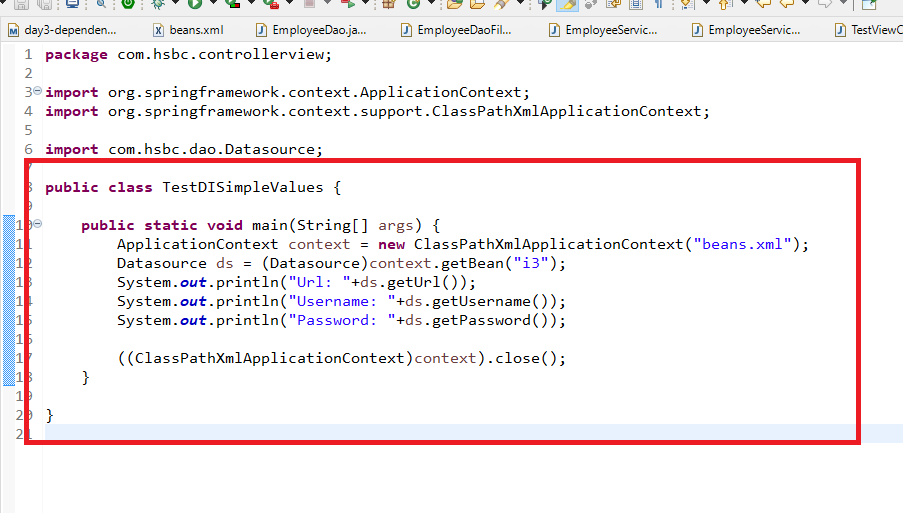
Datasource.java



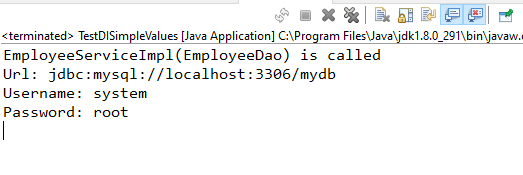
beans.xml



TestDISimpleValues.java



Output:



The above examples performed bean configuration through XML file, but spring also supports configuring bean using annotations some of the mainly used annotations are:

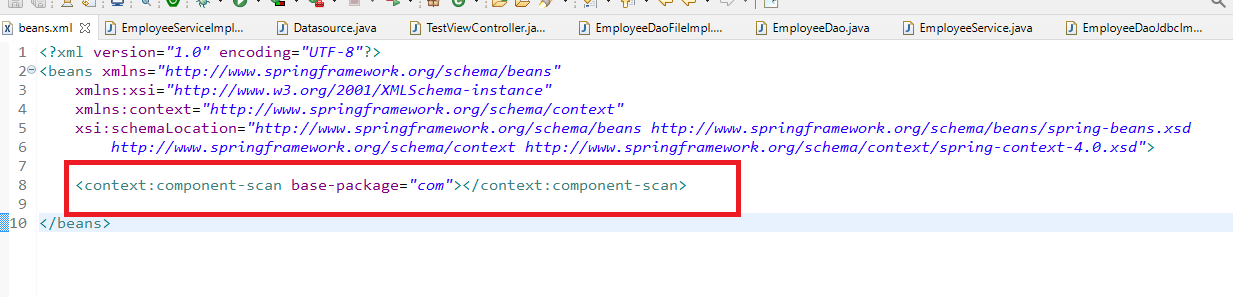
1. @Component
2. @Service
3. @Repository
4. @Configuration
5. @Autowired
6. @Controller
7. @RestController
8. @Bean
9. @Qualifier

Annotation based configuration helps in configuring the beans and the dependencies through annotations but we still need XML file to configure the spring container to use annotations, because in XML file you will write one tag to scan the classes with annotations.

i.e., <context:component-scan base-package = “com”>

The above tag scans all the annotations like @Component, @Repository, @Service, @Configuration, @Controller, @RestController and so on present from com package and their sub-packages

beans.xml



EmployeeDao.java

**package** com.hsbc.dao;

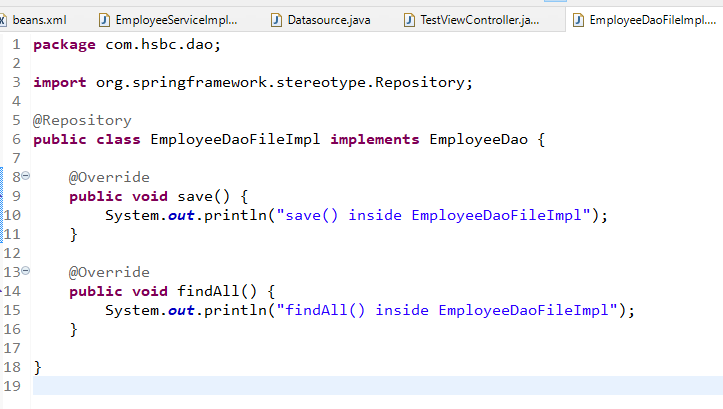
**public** **interface** EmployeeDao {

**public** **void** save();

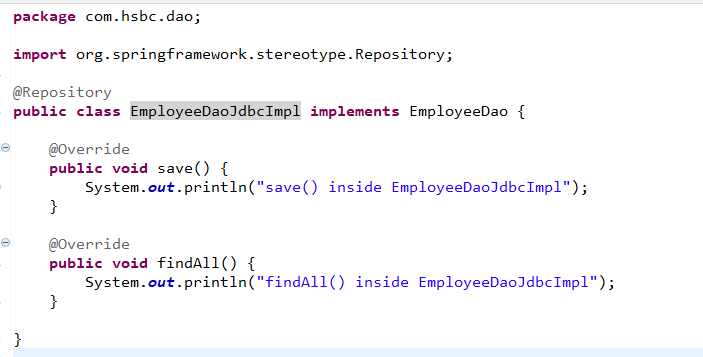
**public** **void** findAll();

}

EmployeeDaoFileImpl.java



EmployeeDaoJdbcImpl.java



EmployeeService.java

**package** com.hsbc.service;

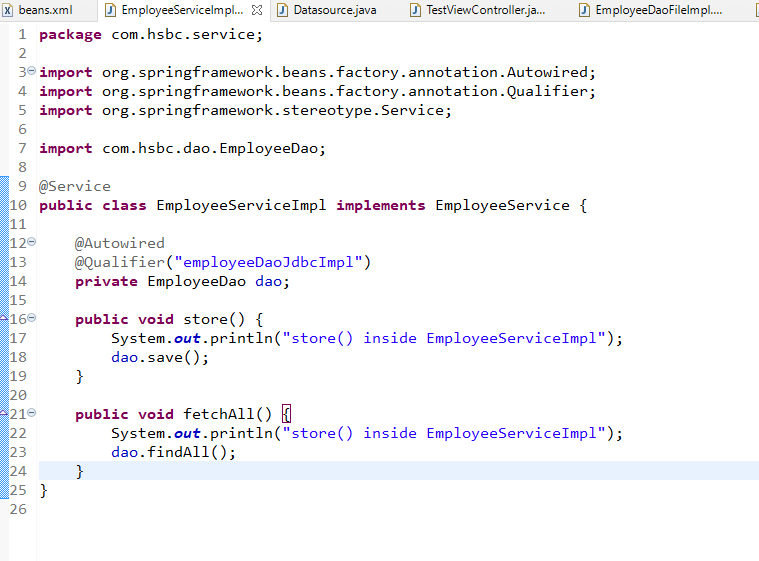
**public** **interface** EmployeeService {

**public** **void** store();

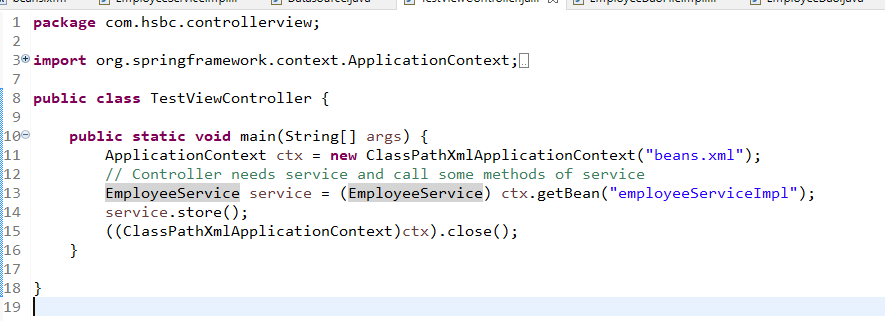
**public** **void** fetchAll();

}

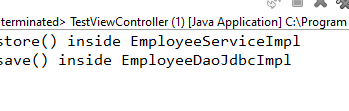
EmployeeServiceImpl.java



TestViewController.java



Output:



Exercise:

1. Configure the DAO using @Repository and Service in the XML file and supply the DAO instance in the service from XML and test whether you can get the service instance using getBean()
2. Configure the DAO in XML and Service using @Service and use @Autowired to the DAO reference in the Service.
3. Use @Value to assign for the username, url and password of the Datasource class and @Component to instantiate the Datasource and test whether you can get the datasource instance from getBean and call the getter methods to see values of username, url & password

@Configuration: Java based configuration where you create object & allow spring to maintain it.

It is used to configure all the beans in the class and provide to the spring container.

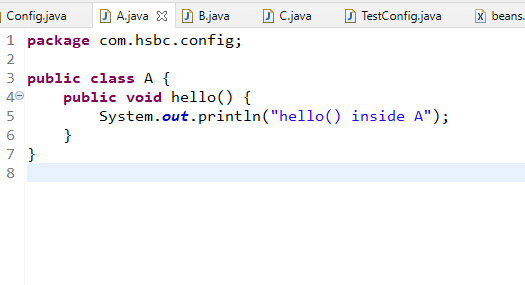
@Configuration

public class Config {

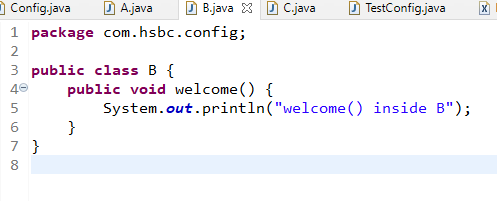
@Bean  
 public Xyz xyz() { return new Xyz(); }

@Bean  
 public Abc abc() { return new Abc(); }   
}

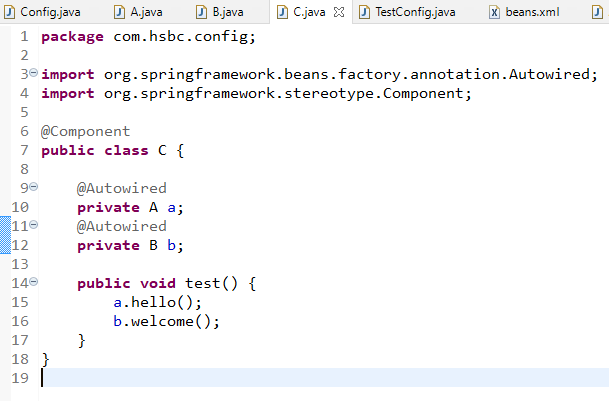
A.java



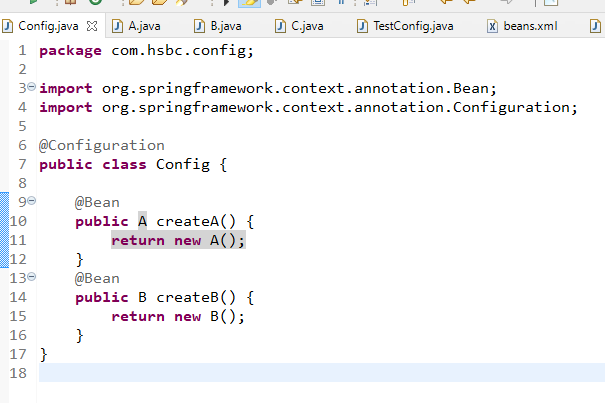
B.java



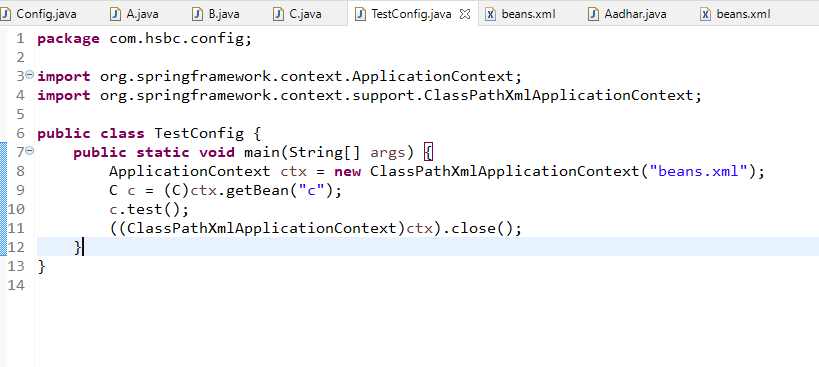
C.java



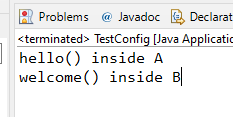
Config.java



TestConfig.java



Output:



AOP:

Aspect Oriented Programming, it allows a task to be done when some methods are executed without explicitly calling

Spring JDBC

It is a module which is used to interact with the database, it takes care of providing the connection to the application & also closing the database resources, we need to configure the datasource properties like url, username, password, driver-class in the XML file and you will use a template called JdbcTemplate that can perform CRUD operations, JdbcTemplate needs datasource information hence datasource information instance is injected to the JdbcTemplate

Spring Framework takes care of establishing connection you only need to configure the DriverManagerDataSource instance which will have information like url, username, password, driver-class.

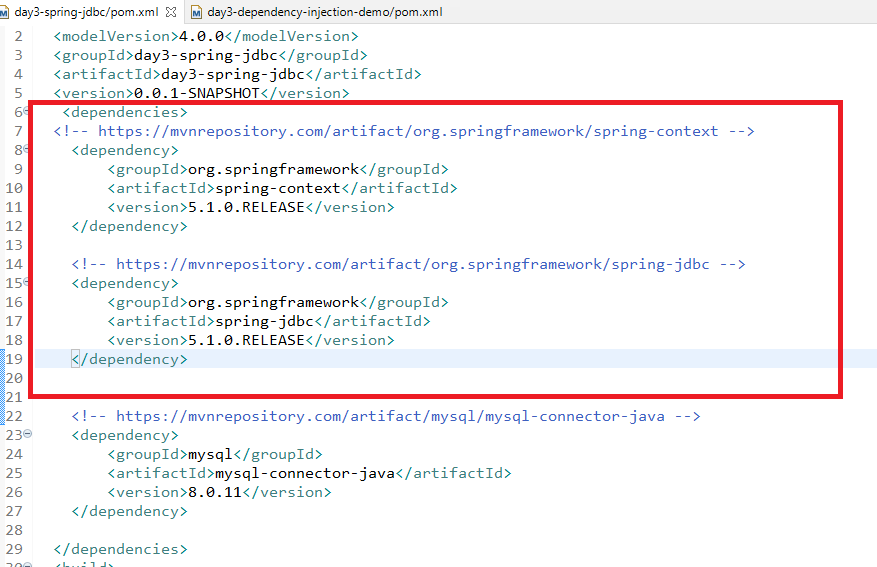
Spring JDBC VS JDBC

|  |  |
| --- | --- |
| JDBC | Spring JDBC |
| You need to create connection, prepared statement, resultset | spring framework does all these |
| You need to handle exceptions | spring framework handles all the datasource exceptions |
| You need to close the resources | spring framework takes care of closing the resources |
| Better for simple applications | Better for complex applications |

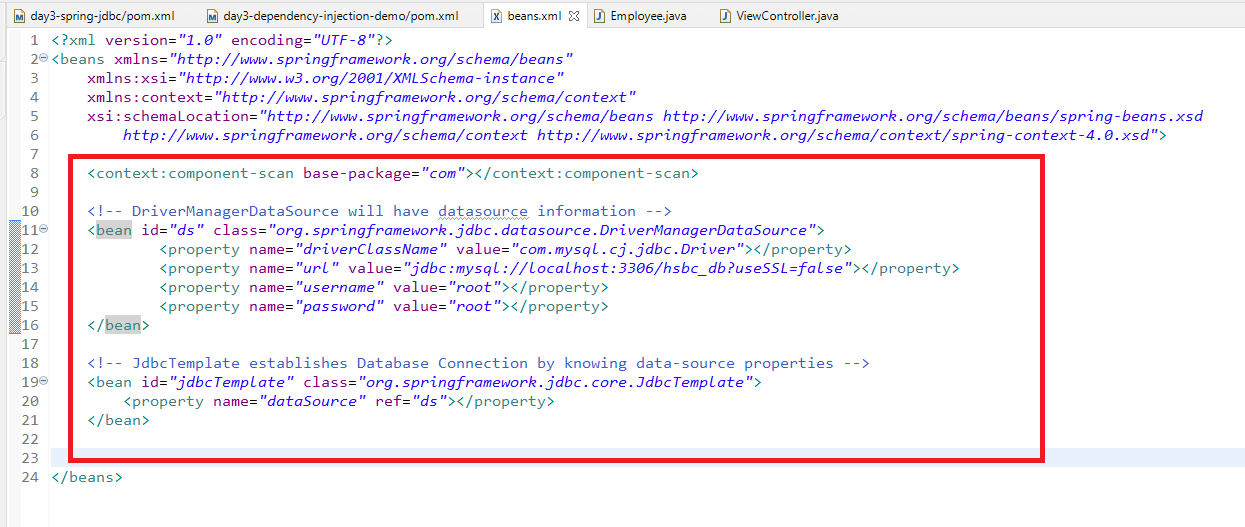
Dependency Required

1. spring-context
2. spring-jdbc
3. mysql-connector

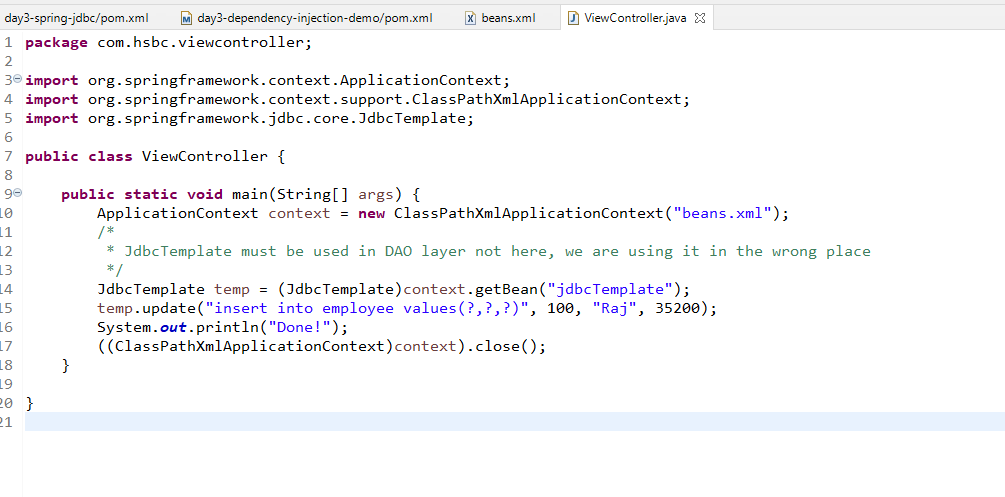
pom.xml



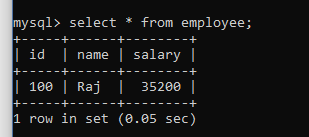
beans.xml



ViewController.java



Output:



Objects Scope:

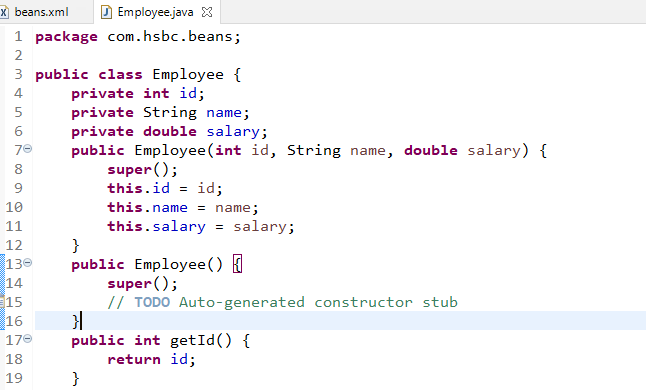
By default spring creates singleton object for every bean configuration, singleton means only one object per <bean> configuration, but if you want multiple instances per <bean> configuration then you should change the scope from singleton to prototype

* singleton: Single object is returned for every getBean
* prototype: For every getBean() new instances of the bean is returned

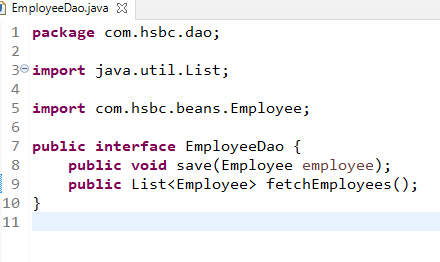
Implementing Layered architecture for the employee CRUD operation using JdbcTemplate

1. Employee.java: java bean
2. EmployeeDao.java: interface
3. EmployeeDaoImp.java: implementation
4. EmployeeService.java: interface
5. EmployeeServiceImpl.java: implementation
6. EmployeeViewController.java: user-interface & controller

Employee.java



EmployeeDao.java

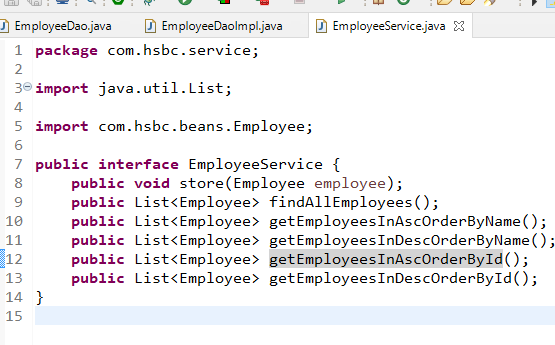


EmployeeDaoImpl.java

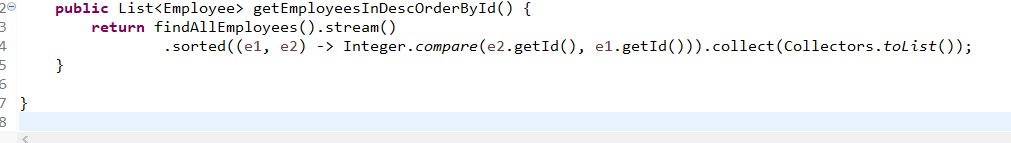


Note: the query method takes sql query & row mapper, which is a functional interface that is called for each row present in the resultset. The query method returns List of object you are returning

EmployeeService.java

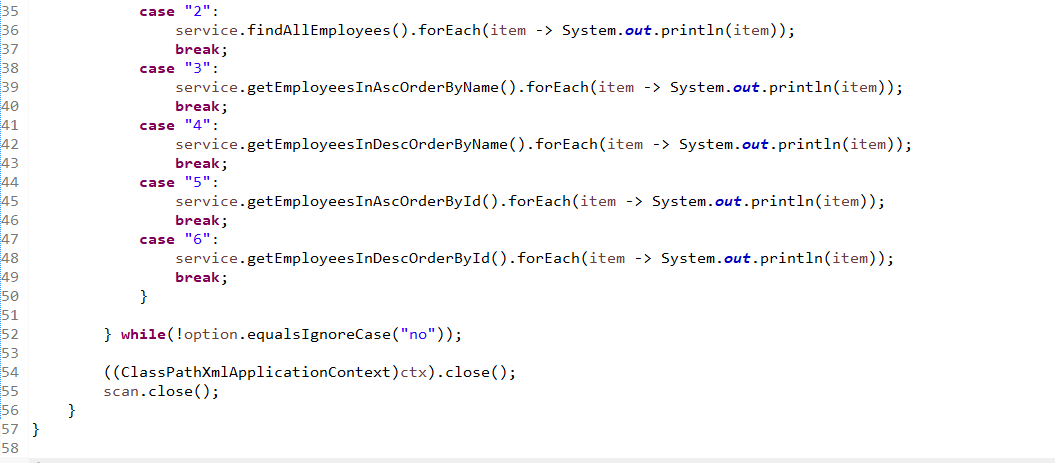


EmployeeServiceImpl.java



ViewController.java





Exercise:

Add few methods in dao & service to perform other operations like

* find employee by id
* update salary by id
* delete employee by id

Spring Web: It simplifies developing web application which provides an inbuilt front-controller called DispatcherServlet which takes care of

* Accepting all the requests and routing to the appropriate controller
* Initializing the container i.e., ApplicationContext
* Return the Model to the appropriate view
* Enables you to have a single controller class to have the navigation logic

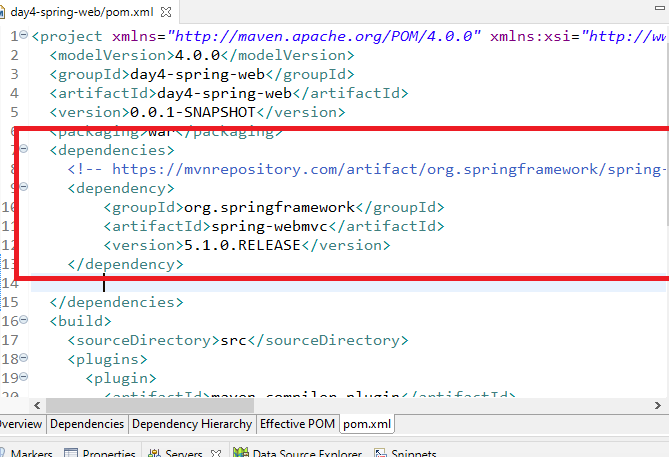
Dependency required for spring mvc

* spring web mvc

Steps:

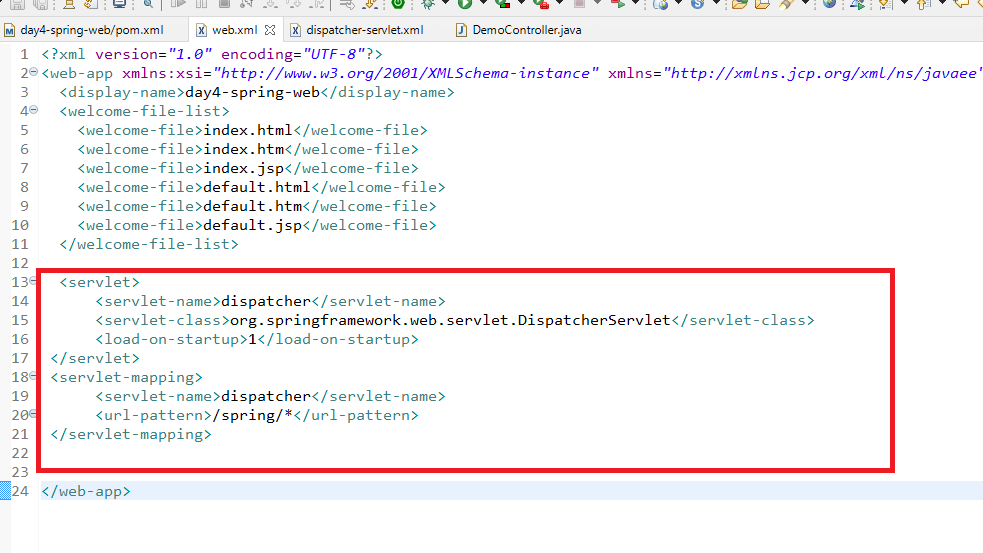
* Configure tomcat server in eclipse
* We need to create dynamic web project and then convert it to maven project
* Add spring web mvc dependency in pom.xml
* configure web.xml to have front-controller configuration i.e., DispatcherServlet configuration
* configure spring beans xml configuration file which will have the bean configurations

pom.xml



Generate web.xml file and configure the front-controller

web.xml

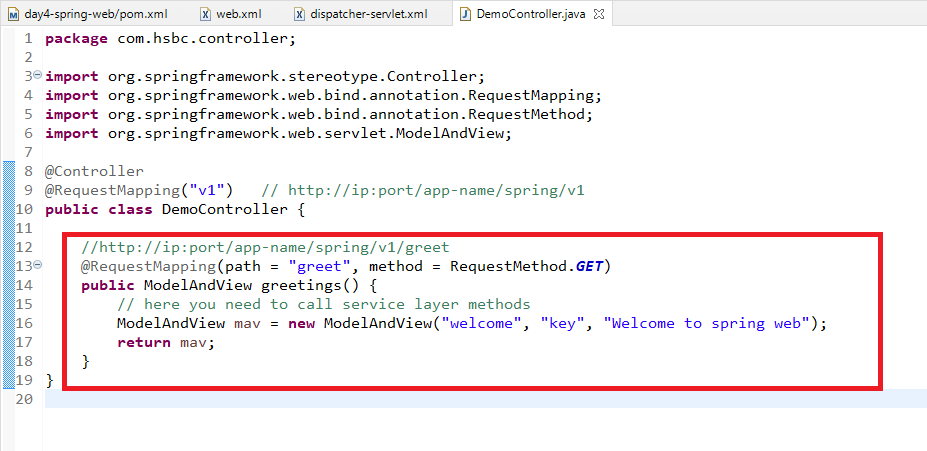


<servlet-name> suggests the DispatcherServlet to locate the bean configuration file using which the spring container will be initialized.

WEB-INF/dispatcher-servlet.xml



DemoController.java



Now the controller should render the page with some model, hence ModelAndView will have 3 parameters

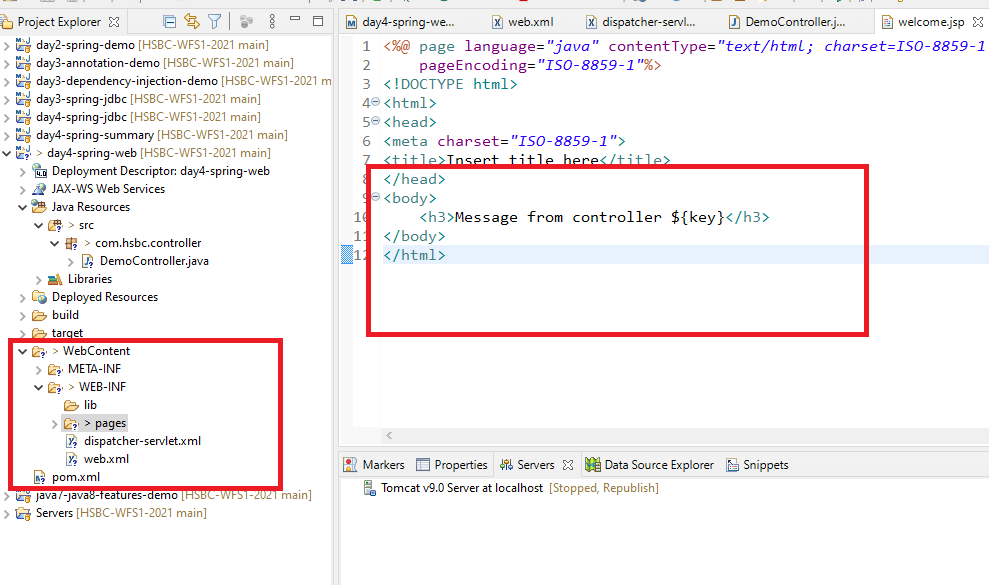
1st parameter is view name

2nd parameter is the model name

3rd parameter is the model

Now the view resolver locates the welcome.jsp in WEB-INF/pages as per the View resolver configuration done in the dispatcher-servlet.xml

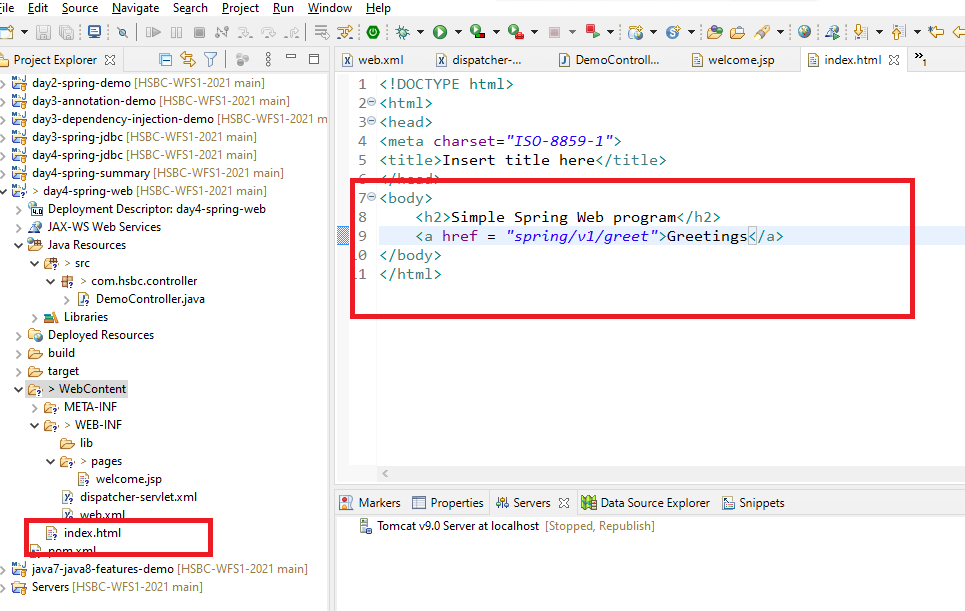
welcome.jsp



You can create index.html and send request to the controller

Note: index.html must be inside Web Content folder

index.html

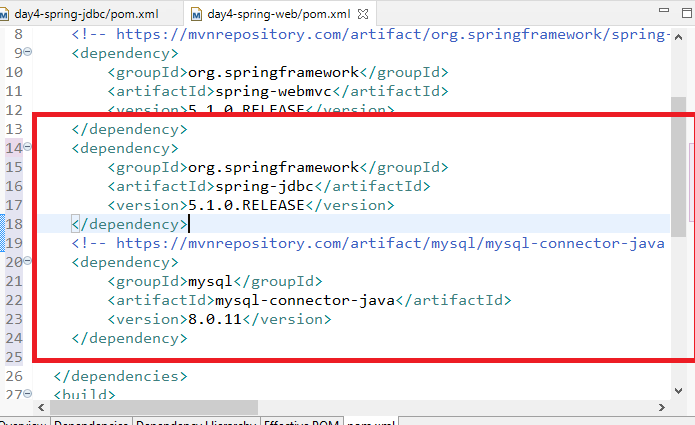


Integrating spring jdbc with spring web

Dependencies required

* spring webmvc
* spring jdbc
* mysql-connector

pom.xml

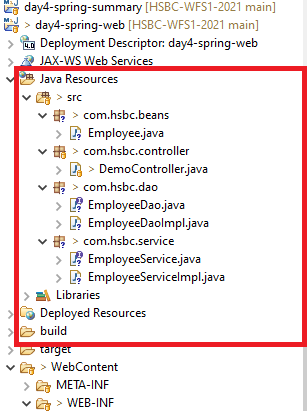


configure the jdbc template in the bean configuration file

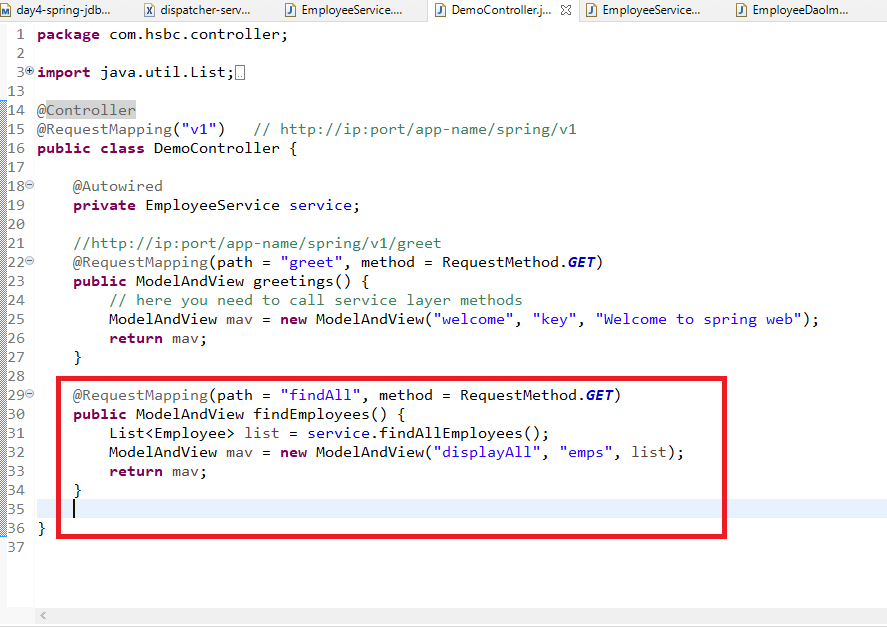
dispatcher-servlet.xml



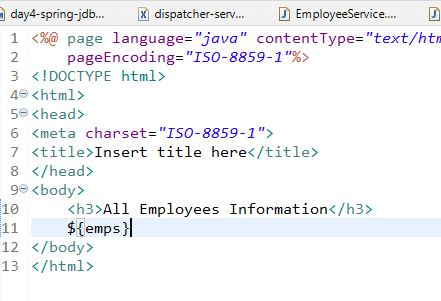
We need Service, DAO and some html and jsp file

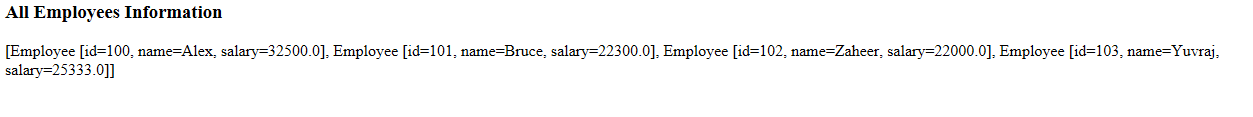


DemoController.java



displayAll.jsp



Output  


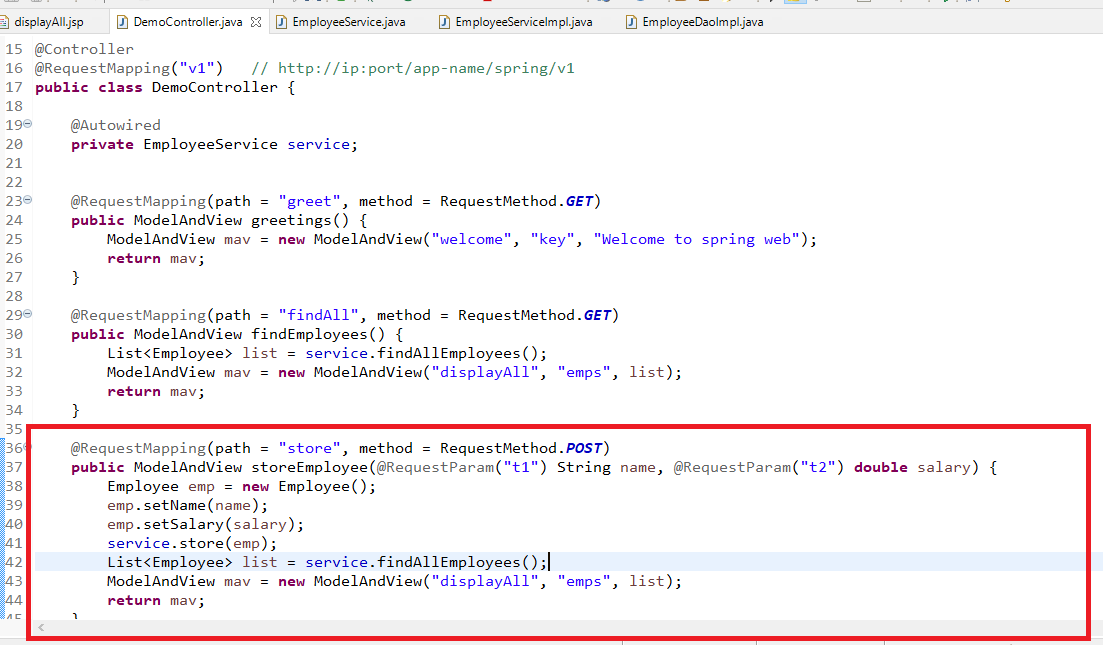
To pass the data from the user you can create a form either in html or jsp

displayAll.jsp



Since jsp is part of spring/v1 in the action we need to mention only the sub-path i.e., store instead of spring/v1/store

DemoController.java



Exercise:

Create html page or jsp page to

* enter the id and display the employee matching the id
* enter the id and update the salary and
* also delete the employee based on the id
* handle EmployeeNotFoundException if in case the employee you are searching is not found

Hint: Propagate the EmployeeNotFoundException from service to controller and let the controller show the exception message in another jsp file using ModelAndView

ReST Webservices: ReST stands for Representational State Transfer, it allows heterogenous applications to exchange the data in one common format it could be in JSON, XML, CSV, Text, HTML and so on, but the more preferred one is JSON.

You need to use an annotation for creating ReSTful webservices

@RestController

@GetMapping: For retrieve the existing resource

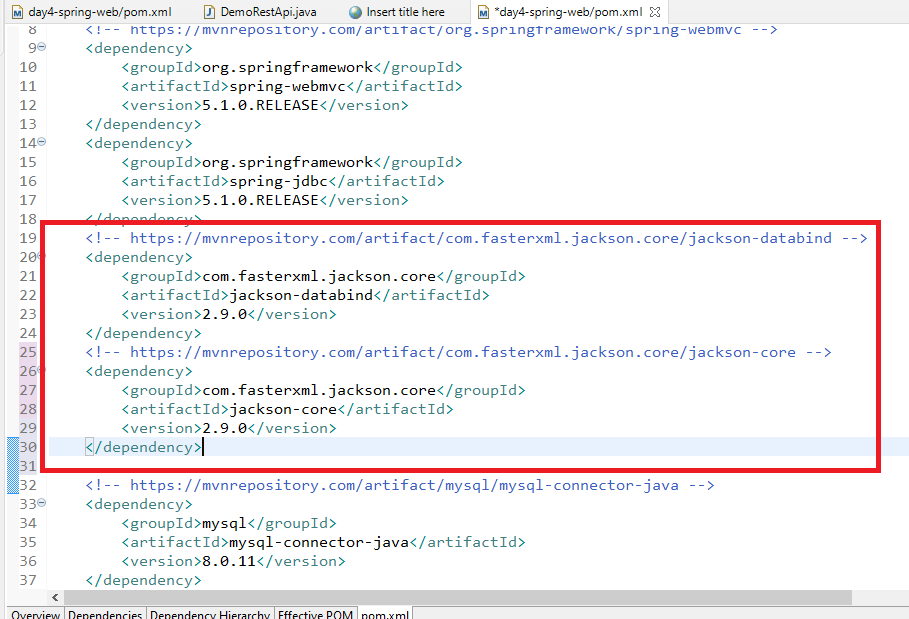
@PostMapping: For creating new resource

@DeleteMapping: For deleting the existing resource

@PutMapping: For updating the existing resource

Java to JSON and vice versa is done by Jackson databind library, you need to add this library in your classpath to convert java to json to java

pom.xml



DemoRestApi

