**Spring - Introduction**

Before we cover up the different Modules of Spring, first let’s understand about Spring in a crux,

* What is Spring ?
* Why it is required?
* How does it help the developer’s community?

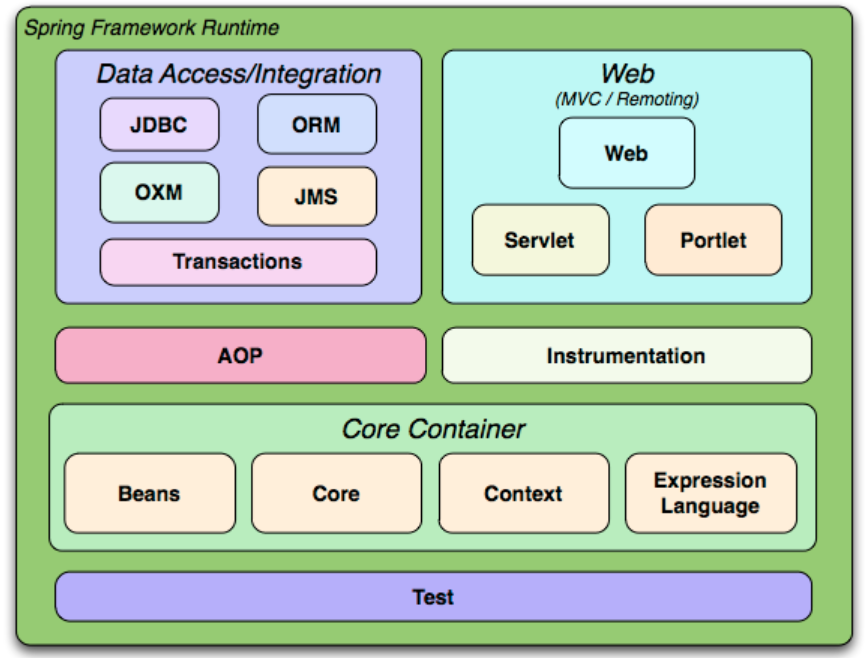
Spring is a Framework (meaning, not a different coding language) which is plugged/bonded on top of Java Platform by making use of Spring libraries. It helps the developers to focus mainly on business functionalities with minimal efforts in handling the dependencies and tight-coupling & avoiding the boiler plate coding.

Some of the features of Spring are,

* **Writing Loosely coupled** codes using **Inversion of Control (IoC)**
* **Dependency Injection** **(DI)** – a form or a principle adhering to IoC
* Reduce the number of lines of codes by **avoiding boiler plate coding** such as
  + Avoid Creation & Destruction of Objects for each classes
  + Avoid Coding for Opening & Closing of DB Connections
* Testing the code modules are quite easy with Mock Objects due to loose coupling

At this point, you will find it bit confusing about IoC or DI. Do not worry, we will figure it out while explaining the Modules of Spring.

**Spring Modules – Content of Architecture**



**Reference**: <https://docs.spring.io>

**Overview of Spring Modules - Introduction**

Spring Framework Modules can be highly classified into below parts.

* **Core Container** – This acts as the main/base module of the Spring which contains the below sub modules
  + Beans
  + Core
  + Context
  + Expression Language
* **Web Layer** – This module is responsible for creating Web Applications. Sub Modules of Web are
  + Web
  + Servlet
  + Portlet
* **Data Access/Integration Layer** – This module is responsible for interacting with the database as well as provides support for Integration with ORM frameworks. Sub Modules includes
  + JDBC
  + ORM
  + OXM
  + JMS
  + Transactions
* **Test Layer** –This layer has its own Testing framework which has good binding & support for testing the application using Junit or Mockito or TestNG.
* **Miscellaneous Layer** – This independent layer has two more different modules
  + AOP (Aspect Oriented Programming)
  + Instrumentation

**Spring Modules Explanation**

**Core Container – Beans & Core**

Core Container acts as the brain of the Spring Module which is responsible for implementing the **Inversion of Control (IoC) or Dependency Injection (DI)**. This helps in achieving the loose coupling between classes.

In a conventional way of writing code, developers will have the control over the code in creating the objects & injecting the dependencies. Here the Spring frameworks takes the control of doing the above-mentioned activities, that’s why the term is coined as ***“Inversion of Control”*** - i.e., the control is inversed!

**Dependency Injection** is a pattern which adheres to Inversion of Control for resolving the dependencies

So, in Spring Terminology – Objects, which acts as the heart of your application & are managed (i.e., instantiation, configuration & assembling) by the IoC Container are called as **Beans**.

Until here, you should be clear about IoC concepts. But how does the beans get instantiated? It is taken care by means of **Autowiring**. Spring framework makes use of Autowiring for instantiating the beans at the time of booting (starting) up the application.

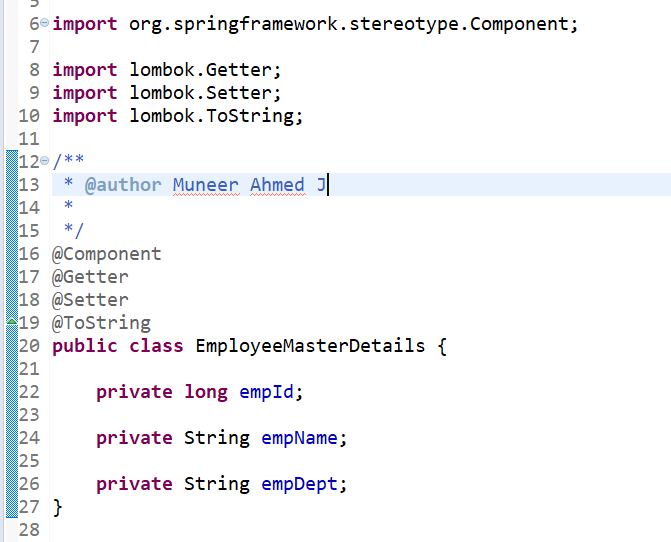
**Types of Dependency Injection**

* Field Based Dependency Injection
* Setter Based Dependency Injection
* Constructor Based Dependency Injection

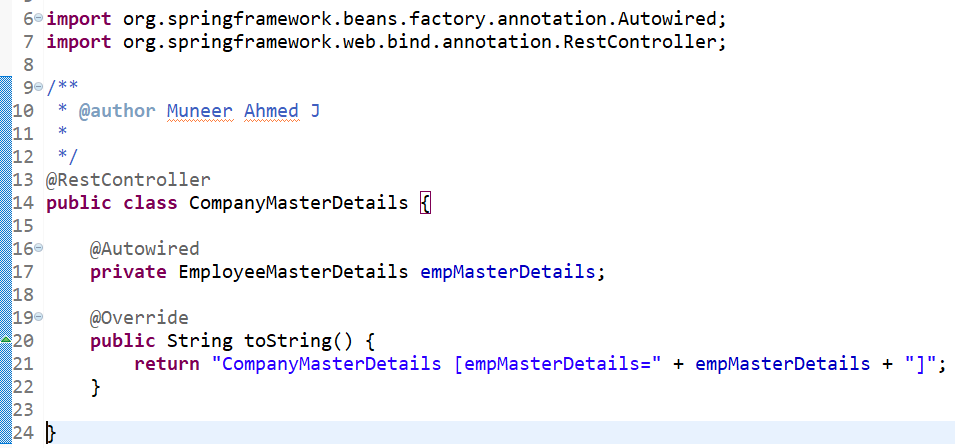
Let’s understand Dependency Injection with a real time example.

**Note**: we will be making use of Java configuration-based (Annotation) examples & not XML based configurations.

**Employee Master Details Class**



**Field Based Dependency Injection of EmployeeMasterDetails in CompanyMasterDetails Class**

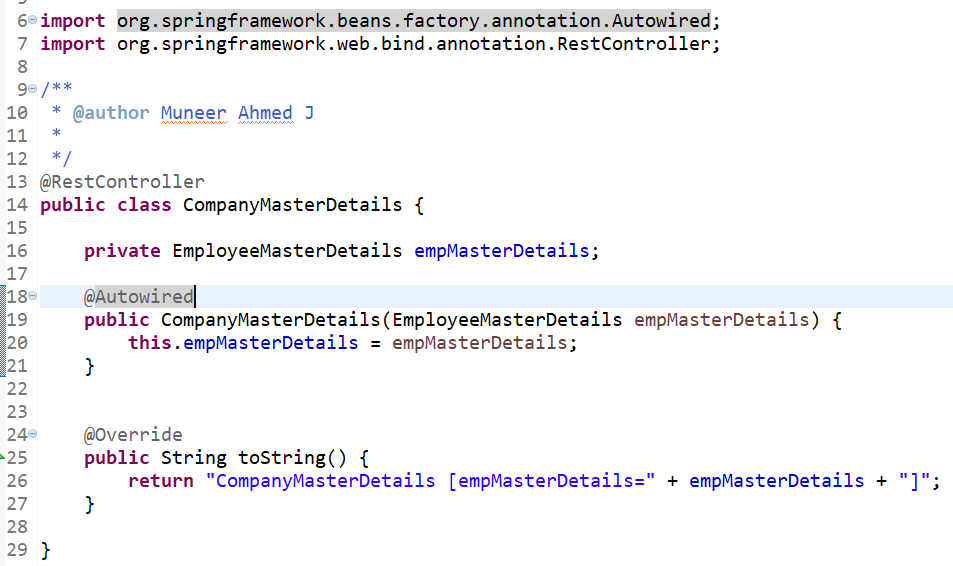


EmployeeMasterDetails class has been created as a Component using @Component (line 16). Spring will automatically recognize this during initialization of the application & will create a bean object for EmployeeMasterDetails class.

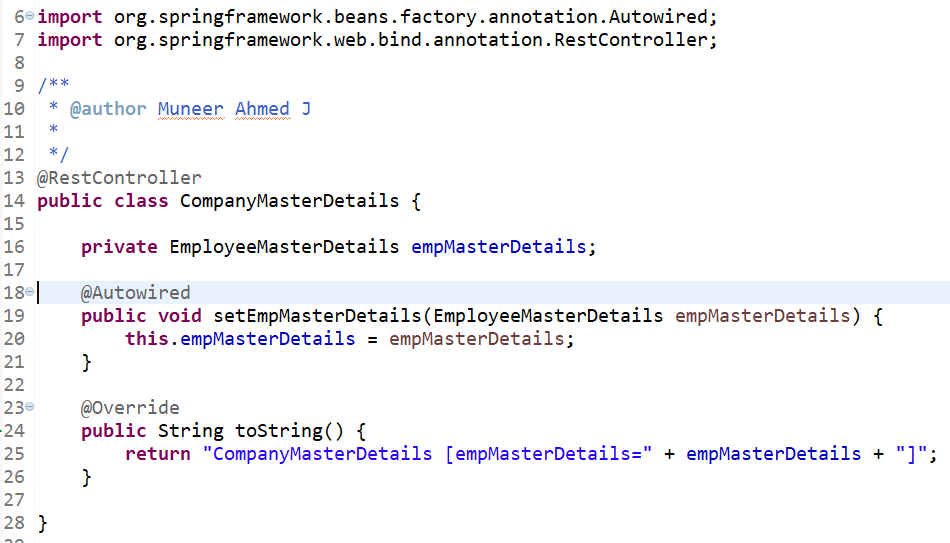
After this, when Spring detects that EmployeeMasterDetails bean object has been autowired into CompanyMasterDetails class (line 16 & 17), the bean object created for EmployeeMasterDetails will be injected into CompanyMasterDetails class.

In this way, Spring framework will perform the Dependency Injection automatically. So, if you notice here, the developer has not created any object by using “new” keyword in CompanyMasterDetails class.

**Constructor based Autowiring using same use case**



**Setter based Autowiring using same use case**



**Core Container – Context & Expression Language**

Spring Context is a sub-module of Core Container which has been built on the base of Spring Core & Beans. It is mainly responsible for instantiating & configuring the beans by scanning the configurations from Spring annotations or XML files.

Additionally, it provides support for Internationalization(i18N), resource loading & event-propagation.

**Internationalization (i18N)**

Spring Internationalization helps in designing the Web Application to be accessed/adapted to various languages across the world.

This will help the end users to access/walk through the application or website in their own language.

Since there are 18 words in between I & N in Internationalization, it is also termed as i18N.

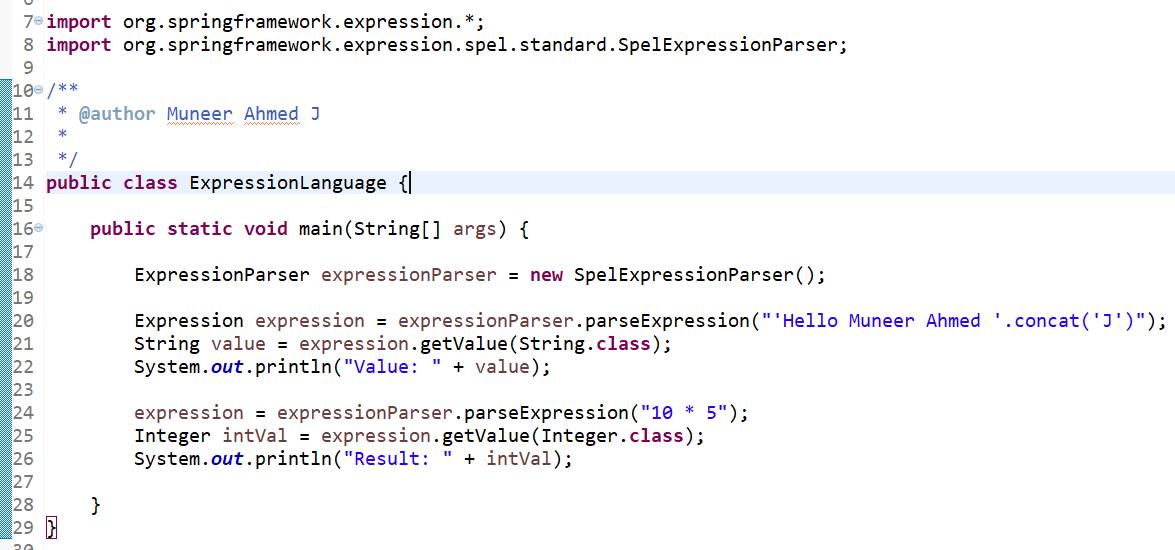
i18N can be achieved by configuring the messages/sentences in message-{language}.properties file.

Example, for English – the words & sentences can be maintained in message-en.properties whereas for Hungarian, it can be maintained in message-hgr.properties file.

**Spring Expression Language (SpEL)**

SpEL is a Spring based Expression Language for evaluating the expression such as Mathematical Operations, Logical & Relation Operations, Conditional Operations & RegEx.

Real time example for SpEL,



Here the output will be,

* Value: Hello Muneer Ahmed J
* Result: 50

**Data Access/Integration Layer**

Data Access/Integration Layer contains the below

* JDBC
* ORM
* OXM
* JMS &
* Transaction Modules

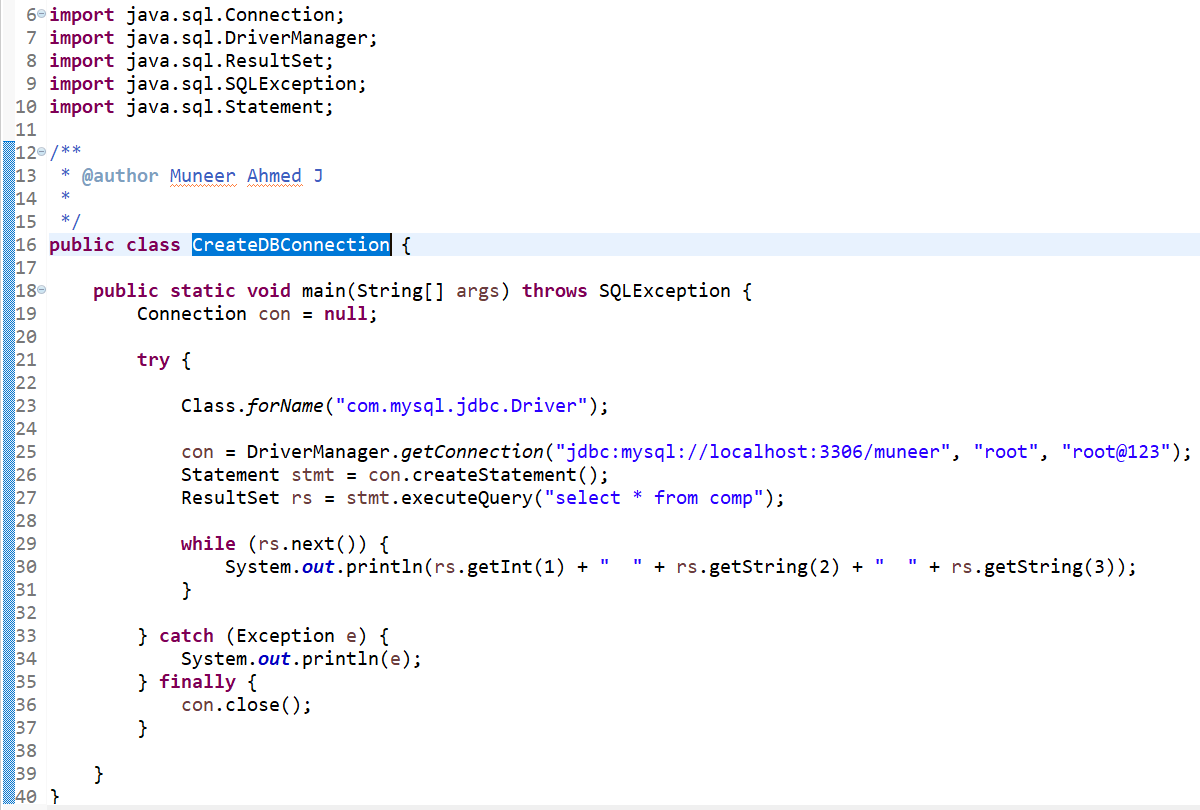
**JDBC Layer**

JDBC Layer provides a wrapper or an abstraction over the conventional JDBC.

While using conventional JDBC Connection, developers needs to write lot of boiler plate coding such as Creating, Opening & Closing of DB Connection, Executing Queries etc., But with the help **JDBC Template**, (i.e., the layer provided by Spring), the Spring Framework itself takes cares of the above-mentioned tasks.

Additionally, JDBC Templates are thread safe, i.e., only with a single instance of JDBC Template, we can make use of it in different DAO classes.

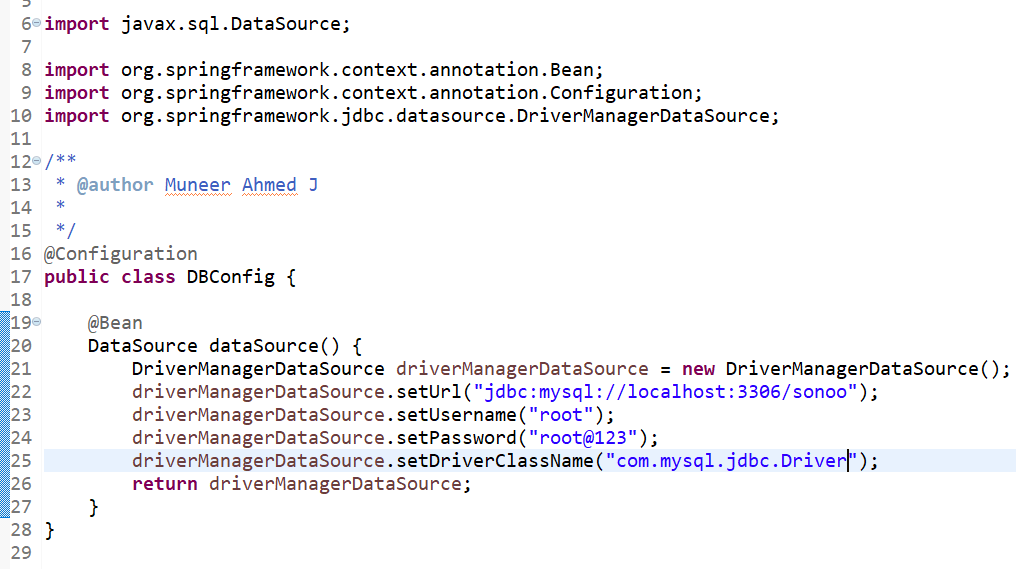
**Conventional way of establishing a JDBC Connection**



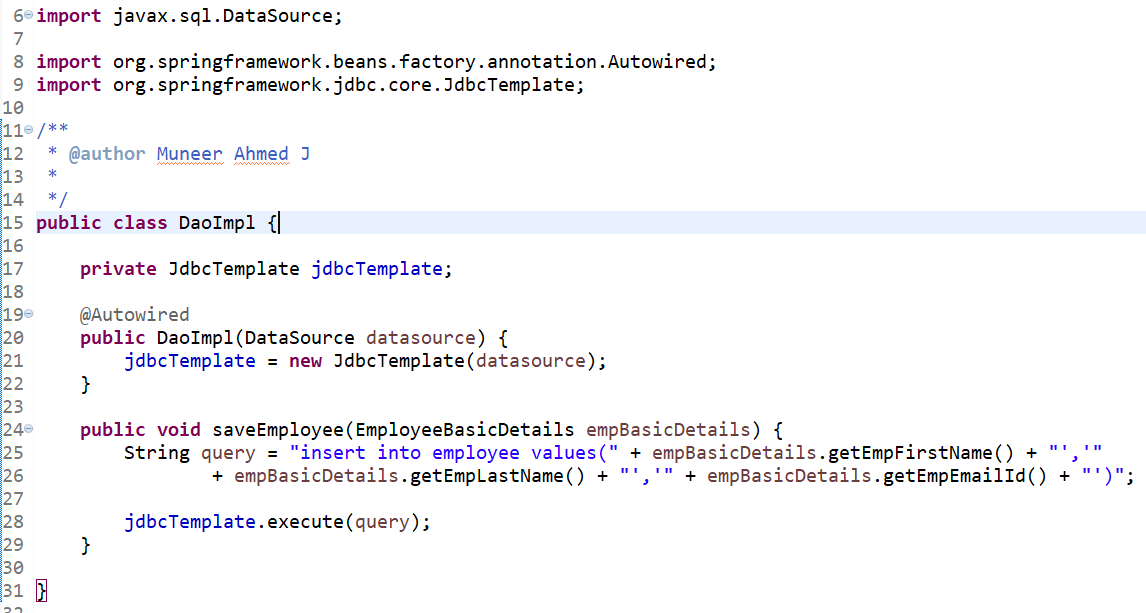
Here, developers needs to register the DB driver, manually create the connection and a statement, then execute the queries and get in a result set & apply the business logics. Once done, it should be closed to avoid Connection Pool Exhaustion.

All the above process contains boiler plate coding. Now lets see how Spring JDBC takes care of this.

**Using Spring JDBC Template**



DBConfig class for configuring the DB Configurations such as Driver, URL, Username & Password.



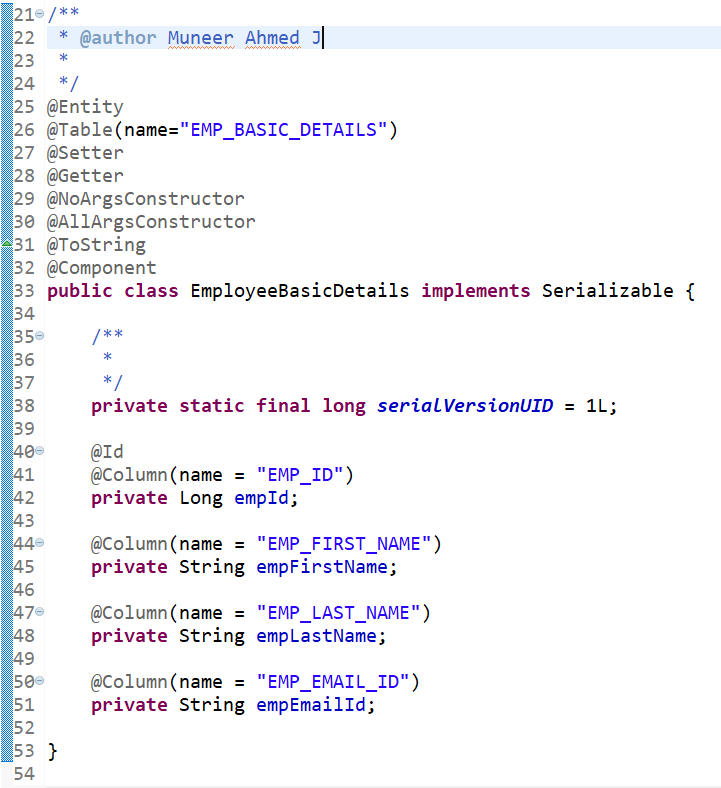
DaoImpl class which makes use of JDBC Template for executing the queries. Here you can notice that, we are not writing any boiler plate codes i.e., explicitly opening/closing the DB Connection.

**ORM Layer**

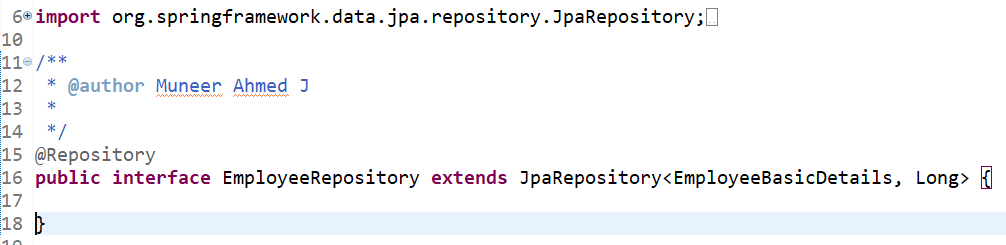
Spring ORM (Object Relational Mapping) Layer provides support for integrating with various ORM frameworks such as Hibernate, JPA (Java Persistence API) etc., Spring doesn’t provide any changes in the ORM layer, but it provides some additional features for handling the ORM layer efficiently.

To make things clear, JPA is only a specification. To make use of it, you will need an implementation for it. Few of the implementation frameworks are Hibernate & Spring JPA. Now the most advanced version from Spring is Spring Data JPA which is widely used along with Spring Boot.

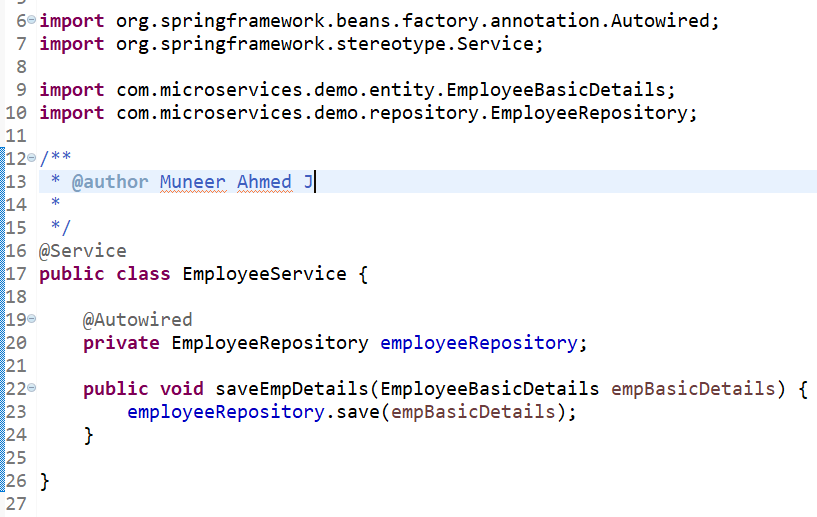
Now let us see a real time example of Spring Data JPA.



EmployeeBasicDetails is an entity class which maps the Java class with the Database entity.



\*\*\*\*\*\*\*\*\*



EmployeeRepository acts as the Repository layer which in-turn extends to JpaRepository. With this, Spring Data JPA Layer provides more powerful & readymade methods such as save, findById, deletebyId etc.,

So, the basic CRUD Operations are provided in ready made fashion. As you can see in EmployeeService class, we have not written any query for saving the Object, instead we are making of in-built save method for saving the employee basic details object into database.

With this, we can identify that Spring provides more powerful abstraction over ORM Layer for interacting the java objects with the database layer.

**OXM Layer**

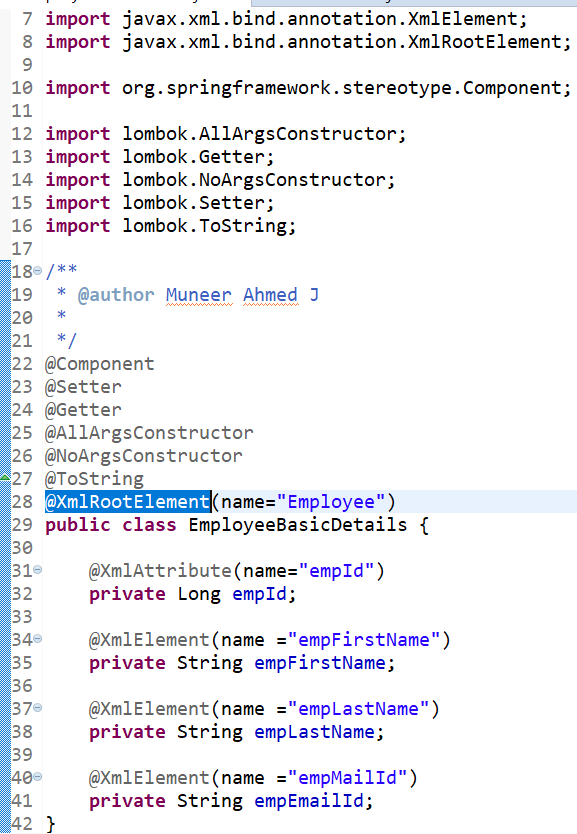
OXM Layer of Spring provides a wrapper or an abstraction layer for supporting Java Object to XML Document mappings & vice versa (i.e., XML Marshalling & Unmarshalling) with various implementation frameworks such as JAXB, XStream, Castor etc.,

**Real time Use Case:**

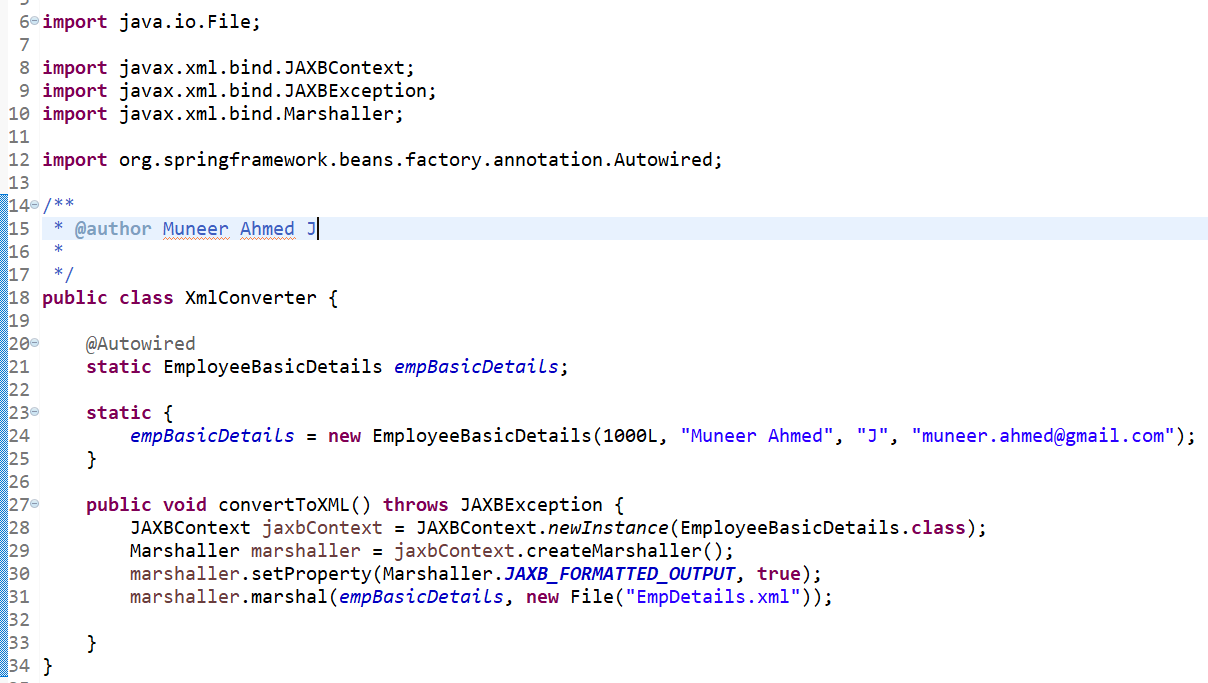
In a traditional SOAP Web Service Environment, we need to accept the requests as well as return the response to the clients in XML format. In this case, developers needs to convert the XML into Java Objects for processing (Marshalling) and after processing, the Java Object needs to converted back into XML Response & send it back to the Clients.

In this case, we will be making use of Spring OXM Layer with (maybe) JAXB for Marshalling & Unmarshalling.

**Code snippet for converting Java Objects into XML**

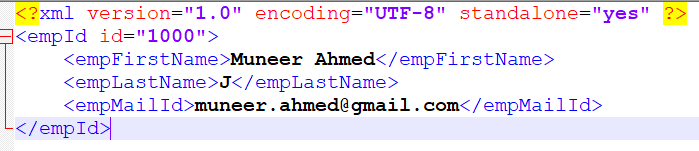


\*\*\*\*\*\*\*\*\*\*\*\*\*



EmployeeBasicDetails class is the POJO class which has the attributes for XML Element. XmlConverter class converts the EmployeeBasicDetails Java Object into an XML Document by making use of JAXB Libraries.

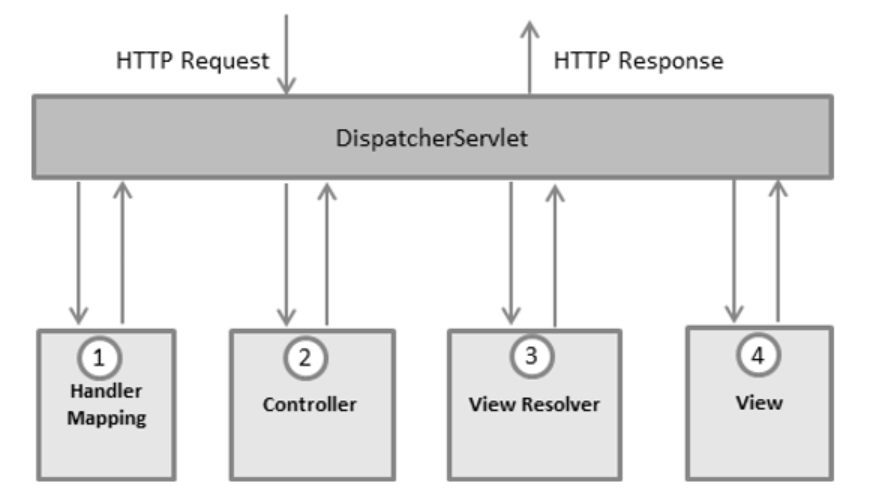
**Output Response**:



**Spring Web/MVC Layer**

Spring Web/MVC Layer refers to Model, View & Controller.

**Architecture of Spring MVC**



**Reference**: <https://tutorialspoint.com>

Below are the generic steps involved in a Traditional Spring MVC Model.

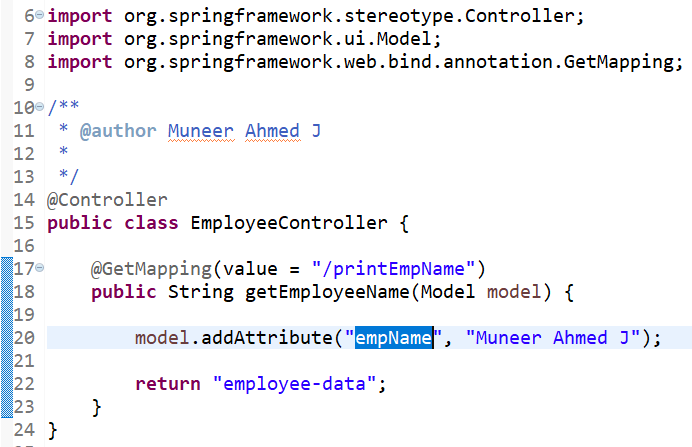
* Once a HTTP request is made from the Client layer, Dispatcher Servlet will provide the request endpoint (API Endpoint) to the Handler Mapping to route it to the required controller.
* Once routed to the required Controller, it is then routed to the Service Layer to implement any business logics & then to the DAO Layer (in case of any DB transactions).
* Once done, data will be set into Model Objects & then will be routed to the View Layer.
* Here, Dispatcher Servlet will provide the details to the View Resolver in finding the corresponding view.
* Once done, the HTTP response along with Model data objects will be routed to the View (HTML file) which will be rendered in the browser.

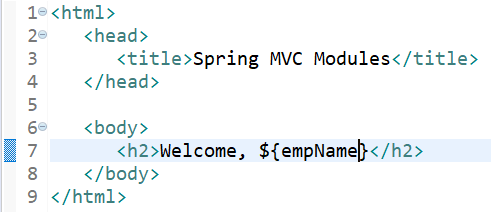
Here to help the Dispatcher Servlet to handle the HTTP request, mapping of the URL will be maintained in web.xml file (i.e., Deployment Descriptor).

Now, after the release of Spring 2.5, annotation-based configurations have eliminated the usage of XML based configurations.

**Real time use case**

Lets consider a HTTP request from a Client to a Web Application to get a simple print message. So, a REST call will be made from Client side to the Server side. Here, Server side will add the message & set the message data to a HTML page and will return the same to the Client side.





Rest call from Client side will be : <http://localhost:8080/printEmpName>

* Here EmployeeController is annotated with @Controller. This will make Spring understand that this is the controller layer which needs to invoked when a REST call is made.
* After looking up the endpoint from Client, it is referring to /printEmpName, so the method which has this endpoint will be invoked by Spring.
* @GetMapping makes sure that any HTTP request with the endpoint as /printEmpName with GET HTTP Request method will be consumed by this method.
* Inside the method, we are adding some data to the Model Object in a key-pair manner using addAttribute method.
* Finally, we are routing the response to a HTML/JSP page i.e., employee-data file.
* In employee-data JSP file, we are retrieving the data which we have set by providing the Key prefixed with dollar symbol.
* This page will get rendered in the browser with the final output as,

Welcome, Muneer Ahmed J

**Spring Test Module**

Spring Test Module has a testing framework which provides support for testing the spring components independently using Junit or TestNG.

Since Spring framework uses Dependency Injection which internally provides loose coupling, it helps the developer to perform the Unit Testing easily with the help of Junit Framework.

Also, it proves to be more powerful during Integration Testing. In a traditional way, Integration testing can be performed once after the application is deployed. But Spring provides an easy way to Mock or Stub the objects & perform the Integration Testing.

**Conclusion**

From this article, we have covered in depth of the importance of Spring Modules with real time scenarios & code snippets. So, in a nut shell - Spring Modules is not a different coding language, but a framework coupled with java for providing easy implementation of most complex items by providing an abstraction layer thereby powering the framework taking care of the control rather than the developer.