spring

# INDEX

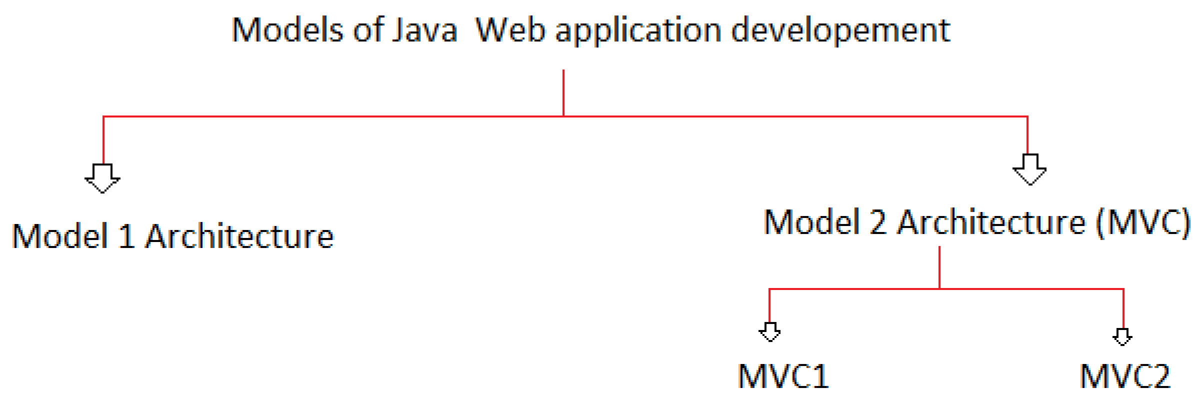
# Spring MVC

# Introduction

## Important logics in web application development:

a. Request data gathering logic (read form data (req params), header values, misc. info).
b. Form validation logic (verifying the pattern and format of the form data).
c. Request processing / business logic (main the logics dealing calculations, analyzation).
d. Persistence logic (like JDBC code/ Spring JDBC/ ORM/ Data code).
e. Middleware services logic (like security, Tx management, logging and etc.).
f. Presentation logic (the UI code that makes end user to supply inputs and to view outputs)
and etc.

## Different models of Java Web Application development:


Models of Java Web application developement
Model 1 Architecture
Model 2 Architecture (MVC)
MVC1
MVC2

## Model 1 Architecture:

- Here we use either Servlet or JSP as the main web components to place the above logics i.e. if servlet comps are used then JSP components will not be used and vice-versa.
- It is more of JSP based web application development.

Note:

✓ It is old and suitable only for small scale projects (max of 10 webpages).
✓ Here we mix up multiple logics in single Servlet/ JSP component so code becomes clumsy because no clean separate between logics.

## Model 2 Architecture (MVC):

- Develop web app as layered app having multiple comps developed in multiple technologies.

MVC:
M -> Model Layer - Data (business logic +persistence logic) (Accounts officer)
V -> View Layer - Represents Presentation logic (for UI for end user) (Beautician)
C -> Controller layer - Monitoring logics (controls the flow) (Traffic police/ Super visor)

Note:

- ✓ MVC 1 architecture we take sperate comps for model layer (like service, DAO classes) but we take single component either Servlet/ JSP component having both view, controller layer logics.
- ✓ MVC 1 is good compare to Model 1 architecture towards separation of logics, if want to get more separation between logics then go for MVC 2.

- In MVC2 architecture we take sperate components for View (HTML/ JSP/ ...), separate component for controller (Servlet/ Servlet Filter) and also separate components for model (Java classes + Java beans) will be taken to get cleaner separation between logics.

MVC 2 architecture advantages:

a. More layers, so there is clean separation between logics.
b. Modifications done in one layer does not affect other layers.
c. Maintenance and enhancement of the web application becomes easy.
d. Parallel development is possible, So the productivity is good.
e. It is industry standard architecture to develop the Applications.

Limitations:

a. Knowledge on multiple technologies is required.
b. For parallel development more programmers are required.

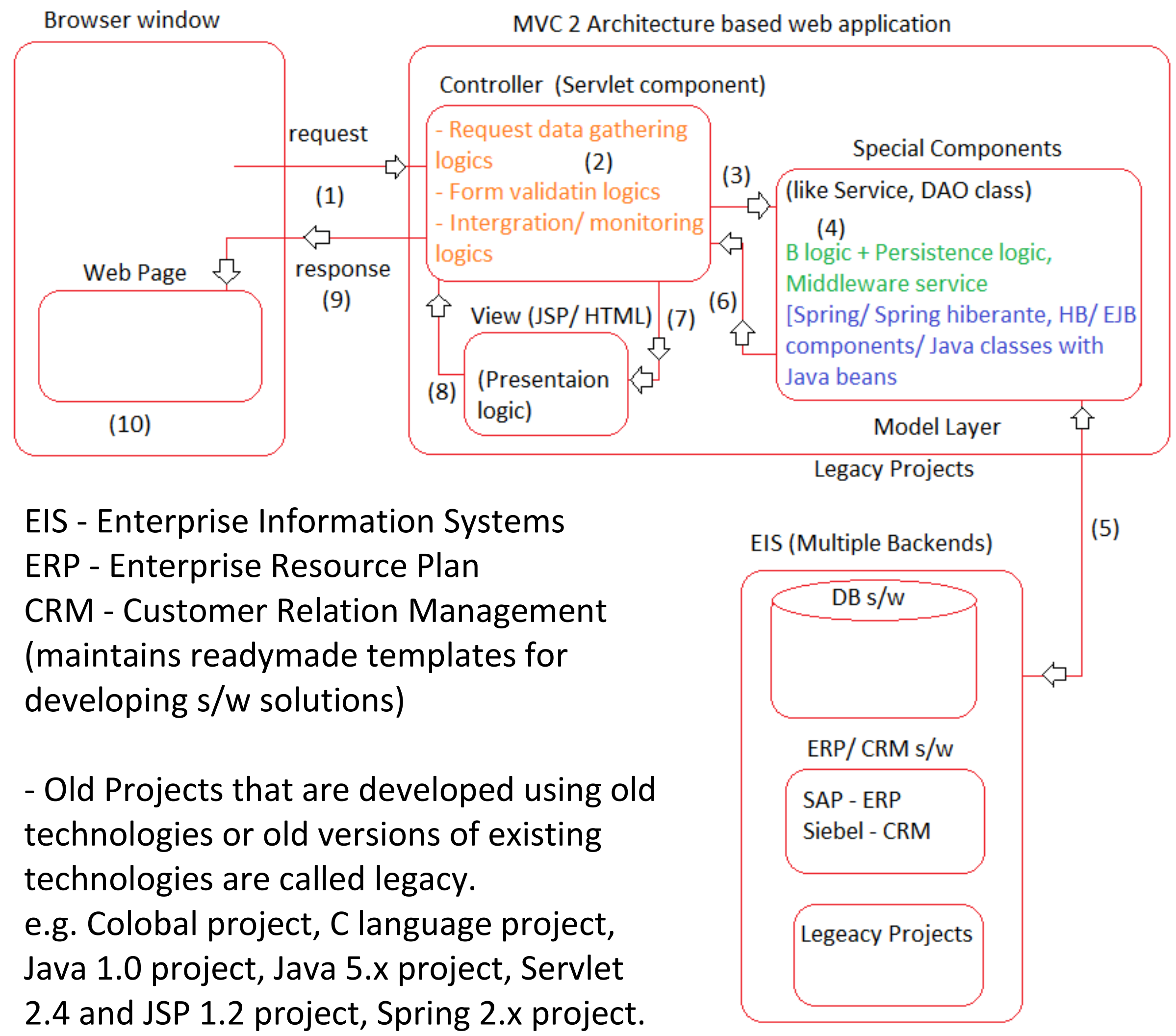
Parallel Development:

Project Team
|--> Part 1 (Presentation-tier developer)
Part 1 of team uses Servlet, JSP and other UI technologies support to develop view, controller layer logics.
|--> Part2 (Business tier developers)
Part2 of team uses Spring, hibernate, EJB, Java classes and etc. to develop model layer business logics, persistence logics.

Note: MVC 3 to 6 are no way related java, they are versions of asp.net MVC.

- Since these two parties (part1, part2) can work parallel. So, we can say parallel development possible. These two parties of team can be there in the same company or can be there in the different companies of two different locations.

## MVC 2 Architecture diagram:


Browser window
MVC 2 Architecture based web application
Controller (Servlet component)
- Request data gathering logics (2)
- Form validatin logics
- Intergration/ monitoring logics
request
(1)
response
(9)
Web Page
(10)
(3)
Special Components
(like Service, DAO class)
(4)
B logic + Persistence logic, Middleware service
[Spring/ Spring hiberante, HB/ EJB components/ Java classes with Java beans
(6)
View (JSP/ HTML)
(7)
(8)
(Presentaion logic)
Model Layer
Legacy Projects
(5)
EIS (Multiple Backends)
DB s/w
ERP/ CRM s/w
SAP - ERP
Siebel - CRM
Legeacy Projects

EIS - Enterprise Information Systems
ERP - Enterprise Resource Plan
CRM - Customer Relation Management (maintains readymade templates for developing s/w solutions)

- Old Projects that are developed using old technologies or old versions of existing technologies are called legacy.
e.g. Colobal project, C language project, Java 1.0 project, Java 5.x project, Servlet 2.4 and JSP 1.2 project, Spring 2.x project.

## MVC2 Rules/ Principles (MVC= MVC2):

a. All operations must happen under control or monitoring of controller Servlet.
b. Every component of every layer is given to have specific logics. So, place only those logics and do not place any additional logics.
c. There can be multiple view components, multiple model components (Service, DAO classes) but it is recommended to take single Controller Servlet component.
d. View camps should not interact with model components directly and vice-versa they should interact through Controller Servlet.

Q. What is difference between Architecture and Design pattern?

Ans.

- Design Pattern gives specific solution for specific problem of application development.
- Architecture gives plan and flow of execution while developing software apps as layered applications. In the creation architecture, we use different Design patterns in different layers MVC (MVC1, MVC2) is an architecture that gives plan to develop web application layered Application.
- In the development of each layer like M, V, C. So many design patterns will be involved.
- In Model layer, we use Service, DAO and etc. patterns.
- In View layer, we use Composite view, View helper patterns.
- In Controller, we use Front controller, Application controller patterns.

Q. Why Web framework are given, when we can develop MVC 2 architecture web application directly by using Servlet, JSP technologies?

Ans. If we develop web application by using Servlet, JSP technologies directly the limitations are

a. We need to develop all the logics of all the layers. No layer logics will be generated dynamically, So the burden on the programmer will be increased.
b. We should remember implement MVC2 rules or principles.
c. We should manage navigation (flow between the comps explicitly).
d. Takes bit time to develop MVC2 rules-based web application.
e. We cannot use other technologies in view layer other than JSP, HTML like Thymeleaf, velocity, free marker and etc.
   and etc.
   (It is constructing the house on your own).

Note: JS is not an independent technology because it cannot be executed on its own it must be embedded with HTML or JSP or etc. So, AngularJS, ReactJS, jQuery also falls under same category.

- To overcome these problems, we can web application frameworks, they are
  a. Struts --> from apache outdated (outdated)
  b. Spring MVC (part of Spring f/w) --> from interface21 (1)
  c. JSF --> from SunMs/ Oracle corpo (2)
  d. ADF --> from oracle corp (so costly)

e. We work --> from Open symphony

## Web MVC benefits:

Web MVC frameworks or web application frameworks provides abstraction layer on Servlet, JSP web technologies having following benefits

a. We need not develop main controller component because they are giving a pre- defined Servlet comp as built-in controller comp based on FrontController design pattern.
b. MVC f/w (Mainly FrontController) itself takes care of navigation among components layers.
c. Most of the MVC2 rules/principles will implemented automatically.
d. Productivity will be good.
e. Most of MVC frameworks allows use different technologies in the view layer including JSP, HTML.
f. Developers feels light weight while developing applications.
   and etc.
   (It is like purchasing readymade house from real estate company or builder).

Note:

✓ If the web application is small, medium scale web application and do not want maintain it long time then go for JSP, Servlet based MVC2 web application development.
  e.g. Movie promotional web sites, Car or Bike model promotional websites and etc.
✓ If the web application is large scale web application and we want to maintain it long time then go for MVC framework-based web application development.
  e.g. E-Commerce web sites, Banking websites, Insurance websites, University websites.

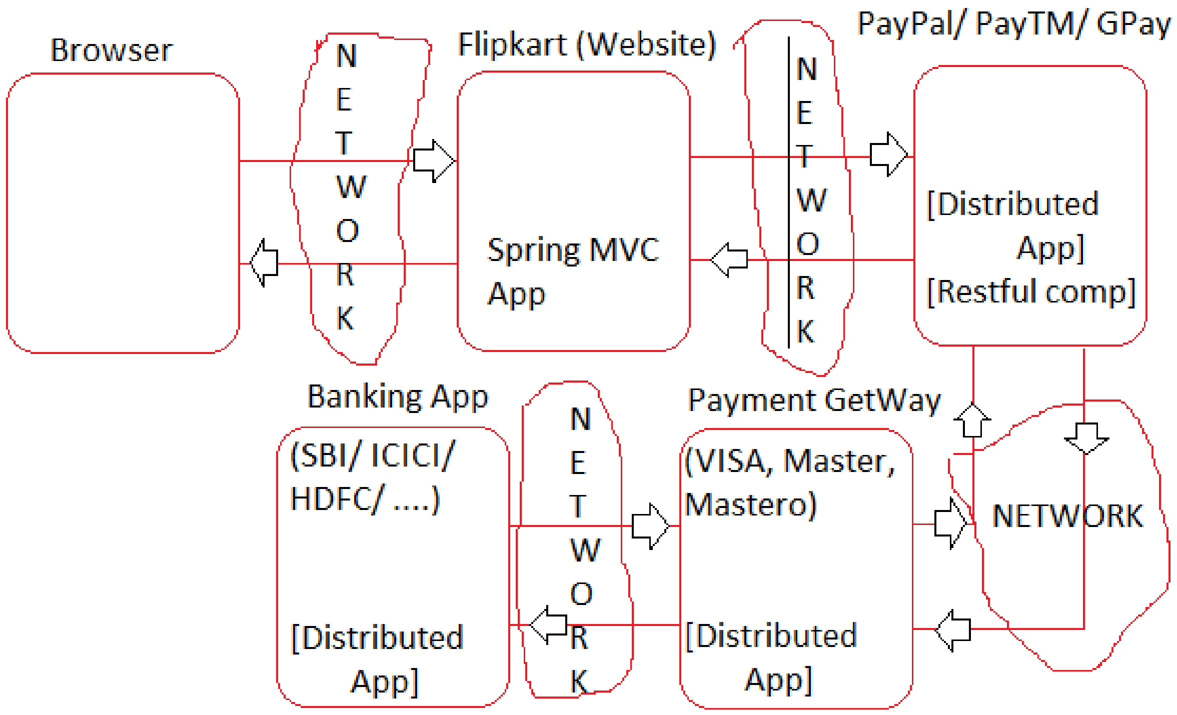
Note: With the integration of jQuery or AngularJS even small, medium scale web site develop is happening in java.

# Spring MVC

## Advantages:

a. MVC framework common advantages also available here.
b. Allows to use different UI Technologies like JSP, HTML, Thymeleaf, Java classes and etc. (We can even integrate all these UI Technologies based with JS and Js tools jQuery, Ajax, AngularJS, Angular, ReactJS and etc.).

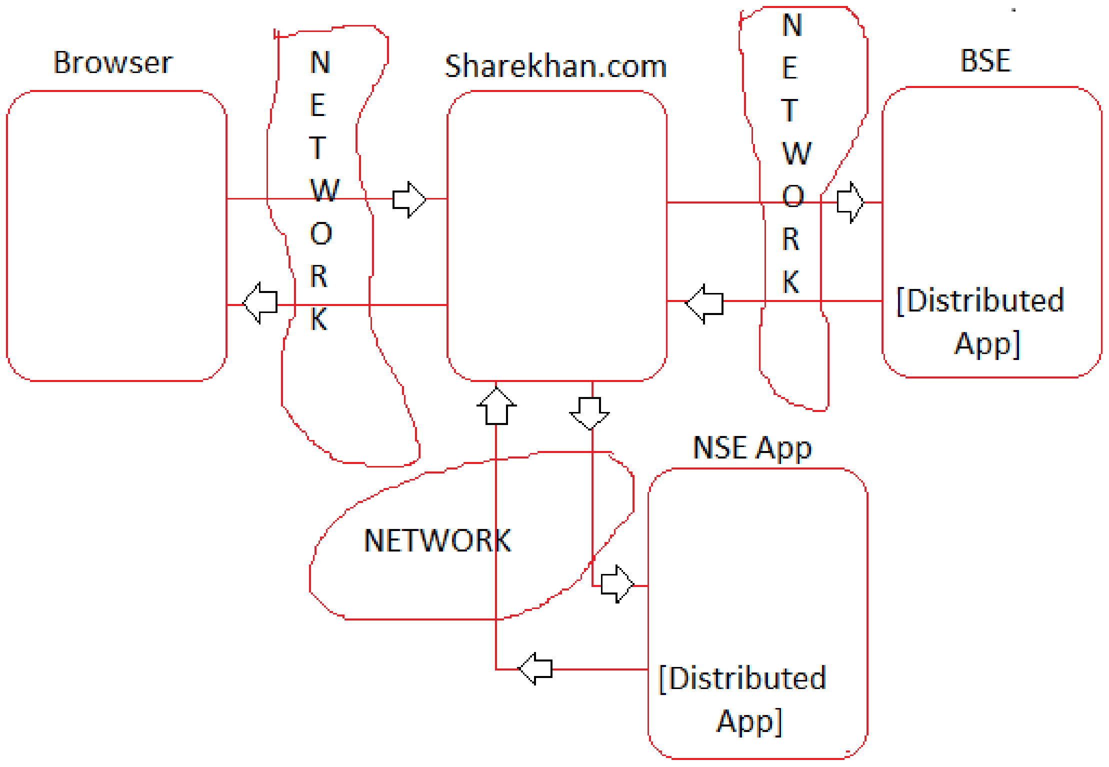
c. Allows to develop MVC2 architecture-based web application by having FrontController design pattern as implicit design pattern. FrontController built-in Servlet comp is Spring MVC:
org.sf.web.mvc.servlet.DispatcherServlet
d. Supports Internalization (118n) Displaying labels, numbers, dates, currency symbols according to different locales (language + country).
e. Easy integration with other modules of spring like AOP, Data, JDBC, ORM, TxMgmt, Security and etc. (we can take the support dependency injection).
f. Interaction with RESTful Distributed Apps is simplified in Spring MVC.
g. Integration with tiles framework is possible (makes every web page comes through collection of web components).
h. Built-in support of file uploading and downloading.
i. Implicit solution for Double posting problem.
and etc.



## Difference between Web Application and Distributed Application:

| Web Application | Distributed Application |
|---|---|
| a) Clients are web based (like browsers) | a) Allows diff types of clients (Web sites, Desktop, Mobile Apps, Smart devices, IVR System and etc.) |

| | |
|---|---|
| b) Generally, Clients software like browser s/w. (not programmable Apps). | b) All clients are programmable Apps. |
| c) Supports only http, https protocols. | c) Supports http, https, SOAP and other protocols. |
| d) End-user interacts with them directly. | d) End-user will not interact directly. |
| e) use Servlet, JSP directly or Spring MVC, JSF, webwork to develop web applications. | e) Use EJB (old), RMI (old), CORBA (old), Webservices (SOAP/Rest). |
| f) Runs based on request-response model. | f) Runs based on method call model. |



We can develop Spring MVC applications in four ways:

- Using XML driven configuration (Declarative Approach) (old - only in maintains projects)
- Using Annotation driven configuration (both in new and old projects)
- 100% Code driven/ Java Config driven configuration (latest but not popular)
- Spring boot driven configuration (very new only in new projects)

## XML driven Spring MVC

Note:

- ✓ Through Servlet component is Pre-defined its configuration in web.xml file mapping with URL/ URL pattern is mandatory, otherwise ServletContainer will not recognize that.
- ✓ So DispatcherServlet configuration in web.xml (XML, Annotation driven configuration of Spring MVC) in mandatory other ServletContainer will to recognize that Servlet component.

- Servlet, JSP components are managed by ServletContainer + JSP Container, where Spring beans like Service classes, DAO class, AOP classes, DataSource will be managed by Spring's IOC containers. In spring MVC we need work with both containers.
- If want to make Servlet comp taking different URLs based requests then should configuration either with directory match URL pattern or extension match URL pattern.

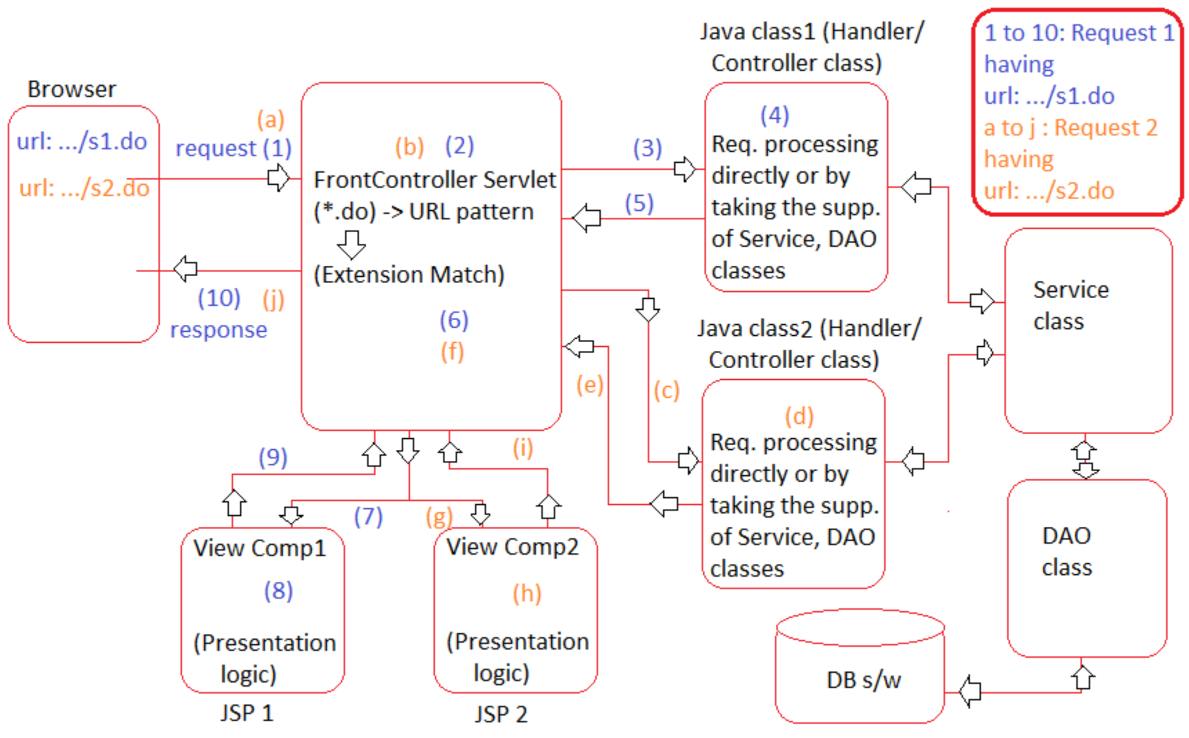
| | |
|---|---|
| /x/y/* | Directory match |
| / * | Directory match |
| *.do | Extension match |
| *.htm | Extension match |

- In Spring MVC "DispatcherServlet" is built-in front controller Servlet.
- In JSF "FacesServlet" is built-in front controller Servlet.
- In Struts "ActionServlet" is built-in front controller Servlet.
- In webwork "FilterDispatcher" is built-in front controller Servlet Filter.

## FrontController Design Pattern

- ➢ FrontController is a special web component of Java web application that traps and takes either all requests or multiple requests, applies common system services like logging, auditing, security and etc., delegates the requests handler classes (Java classes), takes the results (model) from them, passes them to view comps (like JSP components) and passes the formatted results to browser as response.
- ➢ In a single line, we can say Frontcontroller controls all the activities/ navigation of the Application.
- ➢ Keeping business logic or request processing logic in java classes is good practice because they make logics more reusable and invokable from different types of clients but they cannot take direct http requests from clients.

- To overcome this problem, we still write business logic/ request processing logics in Java classes (handler classes/ controller classes) but we develop one FrontController web comp either's Servlet (good)/ Servlet Filter to trap all requests and to delegate to handler classes and also for taking results from handler/ controller classes and to pass, them to view components for formatting results.
- FrontController design pattern (JEE pattern) always implemented in MVC2 architecture of web application development.



Browser
url: .../s1.do
url: .../s2.do
(a)
request (1)
(10) (j)
response
(b) (2)
FrontController Servlet
(*.do) -> URL pattern
(Extension Match)
(6)
(f)
(3)
(5)
(e)
(c)
Java class1 (Handler/ Controller class)
(4)
Req. processing directly or by taking the supp. of Service, DAO classes
Java class2 (Handler/ Controller class)
(d)
Req. processing directly or by taking the supp. of Service, DAO classes
1 to 10: Request 1 having url: .../s1.do
a to j : Request 2 having url: .../s2.do
Service class
DAO class
DB s/w
(9)
(7)
(g)
(i)
View Comp1
(8)
(Presentation logic)
JSP 1
View Comp2
(h)
(Presentation logic)
JSP 2

## Generally,

- FrontController is 1 per web application.
- Handler/ controller classes are 1 per each action or specific work.
- view components (JSP) are 1 per each handler class.
  for example 10 handler classes means 10 view components (JSP).

## Note:

- ✓ For every hyperlink, form submit any request submission one handler class will be taken.
- ✓ To allow FrontController Servlet taking multiple requests or all requests it should be configure either with Extension match or Directory match URL pattern.
- ✓ Since FrontController is doing so many operations it is better to keep it ready during deployment of the web application itself. (enable pre-

instantiation and pre- initialization of FrontController) by enabling of <load-on-startup>.

Q. What is the difference between FrontController and Controller/ Handler class?

Ans.

- FrontController is generally 1 per web application and it is a web component (servlet/ Servlet filter) having system service logics to apply by trapping all or multiple requests.
- Controller/ Handler classes are normal java classes that are taken on 1 per each action/ work basis (app contains more controller/ handler classes) having request processing logic directly or indirectly.

Q. What is the difference between MVC and FrontController?

Ans.

- MVC/ MVC 2 is architecture to develop Java web-based web application as layered Application.
- FrontController is design Pattern to develop "C-> controller" layer of MVC 2/ MVC architecture with better practices having total control on navigation.

  Note: Without MVC/ MVC 2 architecture, there is no FrontController. MVC 2/ MVC architecture is like any company FrontController is like CEO of the company.

Note: In spring MVC Applications we do not develop FrontController we use readymade FrontController called "DispatcherServlet" by configure it in web.xml file as shown below.

web.xml

```
<web-app>
          <!-- Servlet Configuration -->
          <servlet>
                    <servlet-name>dispatcher</servlet-name>
                    <servlet-class>org.sf.web.s.DispatcherServlet</servlet-class>
                    <load-on-startup>2</load-on-startup>
          </servlet>                (>=0 should be given)
          <!-- Servlet Mapping -->
          <servlet-mapping>
                    <servlet-name>dispatcher</servlet-name>
```

```
                <url-pattern> *.htm </url-pattern>
          </servlet-mapping> (Extension match URL pattern)
</web-app>
```

Jars in WEB-INF/lib folder:
spring-web-<ver>.jar
spring-webmvc-<ver>.jar (DispatcherServlet is available here).
org\springframework\web\servlet\DispatcherServlet.

DispatcherServlet Actions on the Deployment:

a. Deployment of Spring MVC web application.
b. Loading of web.xml checking well-formedness, validness and creating web.xml file InMemory Metadata.
c. Because of <load-on-startup> enabled on the DS, the pre-instantiation of DispatcherServlet takes place.
d. As part DispatcherServlet of initialization the init () of DispatcherServlet executes and this method creates IoC container XmlWebApplicationContext by taking WEB-INF/<DS logical name>-servlet.xml as of type spring configuration file by default (in our example WEB-INF/dispatcher-servlet.xml).
e. IoC container Loads Spring bean configuration file (WEB-INF/dispatcher-servlet.xml), checks its well-formedness, validness and creates Its InMemory Metadata.
f. IOC container performs Pre-instantiation of singleton scope beans and also completes injections on them like Service classes, DAO classes, DataSource, Handler classes and etc.
g. Keeps Singleton scope spring bean objects in the Internal cache of IOC container.
h. Keeps DispatcherServlet object in the Internal cache of ServletContainer.

Internal Cache of IoC container

| | |
|---|---|
| service | Service class object ref |
| dao | DAO class object ref |
| handler | Handler class object ref |

(Managed by DispatcherServlet)

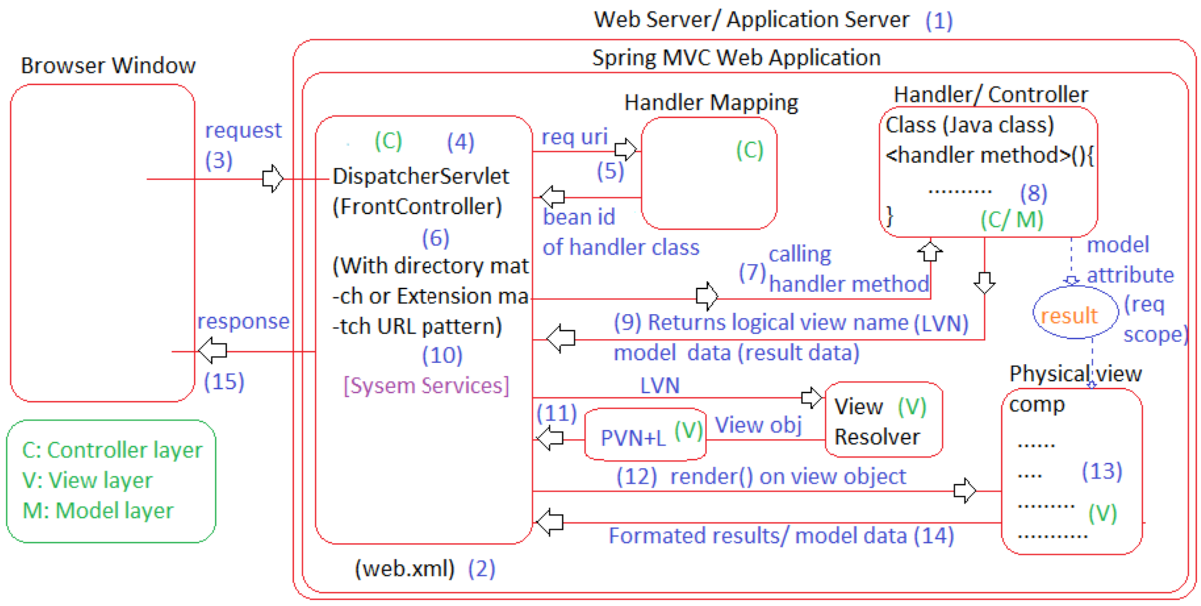
InternalCache of ServletContainer

| | |
|---|---|
| dispatcher | DispatcherServlet object ref |
| | |

Note: Spring MVC is designed around FrontController "DispatcherServlet" i.e.

entire navigation or flow will be taken care by DispatcherServlet. The FrontController DispatcherServlet acts as single entry-Single exit point for all the request and responses.

## Spring MVC Flow (Diagram based)



1. Programmer deploys the Spring MVC Web application in webserver or Application server.
2. All DispatcherServlet related Deployment Actions takes place (refer previous class) based in the configuration done in web.xml file.
3. End-user gives request to spring MVC web application using browser having request URL.
4. As FrontController "DS" (DispatcherServlet) traps and takes the request and applies common System services on the request like logging, auditing and etc.
5. DS handovers the request to HandlerMapping by giving incoming request URI/URL and HandlerMapping maps/ links request URL/ URI with Handler/ controller class and returns its bean id back to DS.
6. DS recieves the bean id of Handler class from HandlerMapping and submits it to DS Managed IOC container and gets Handler class object by called ctx.getBean(-,-) method.
7. DS calls handler method on Handler class object.
8. handler method either directly process the request (or) process the request by taking the support Service, DAO classes.
9. Handler class/ Controller class gives logical view name + model data (result) to DS.

(Note: this logical name can map [linked with different technologies-based view comps without disturbing the source code handler classes).

10. DS takes logical view name (LVN) and model data from handler/controller class and keeps model data in request scope as model attributes.
11. DS gives logical view name (LVN) to ViewResolver and ViewResolver maps LVN to Physical view name (PVN) and returns View Object having PVN and its Location back

    Note: ViewResolver does not render the view component. It will resolve/ identify physical name and Location and gives to DS in the for-View object.
12. DS calls render () on the received View object to transfer/render the control to physical View component.
13. Physical View comp format the results (model data) using presenting logics by getting model data (results) from model attributes (request scope).
14. Physical View Component (UI comp) gives formatted data results (model data) back Frontcontroller DS.
15. DS sends formatted results to browser as response.

Short flow:

Request --> DS HM beanId to DS --> DS get Handler object DS calls handler method --> DS gets LVN + model data keeps in model attribute --> DS gives LVN to ViewResolver --> ViewResolver gives View obj to DS --> DS calls render () on View object --> PVC --> DS --> response to browser

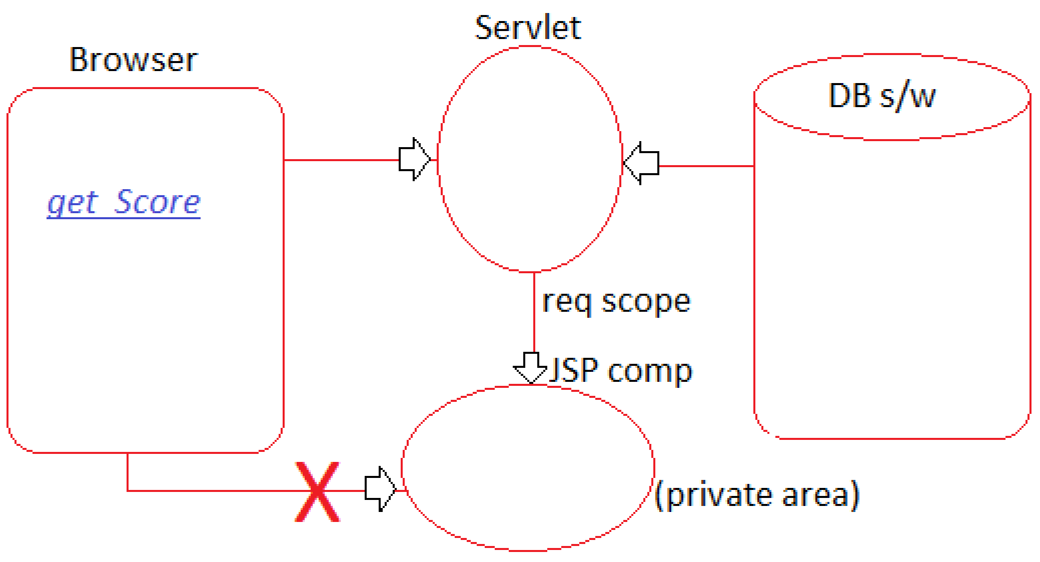
Note:

- ✓ Entire flow of Spring MVC is decided by the FrontController DS. So, we need arrange comps as required for the Spring MVC flow. Some components need developed manually (like handler classes/ physical view comps and etc.). Some other components are already available as readymade classes So we need to configure by injecting required inputs (like Handle mappings, view resolvers and etc.) So, all these should be configured as spring beans in spring bean configuration file (/WEB-INF/ dispatcher-servlet.xml).
- ✓ In real time is recommended to place JSP pages in private area of the web application (either in WEB-INF or its sub folders). Even spring MVC recommends to place JSP files in private area.

```
WEB-INF
   |--> pages
          |--> result.jsp
```

The advantages are,

a. We can hide the technology of website from end users (Secrecy or security will be improved).
b. If JSP page getting request scope data from servlet component then request to JSP page can be avoided and we can eliminate the possibility of displaying ugly values. (technical messages)



## Controller/ Handler class

- It is a java class that implements org.sf.web.servlet.mvc.Controller (I).
- Every controller must be configuring as spring bean in spring bean configuration file having bean id.
- It contains Business logic directly to process the request as Model layer component or it can talk service, DAO classes to get results from them as Controller layer component (recommended).
- In XML driven configuration we take this class on 1 per each action/ work basis.

Controller/ Handler class

```
public class MyController implements Controller(I) {       //invasive

          //Handler method
          public ModelAndView handleRequest(HttpServletRequest req,
                                             HttpServletResponse res) throws Exception{
                    return new ModelAndView("result","sysDate", new Date());
          }
}
```

result: LVN, must be String
sysDate: Model attribute name, must be String
new Date (): Model attribute value can be any object

dispatcher-servlet.xml
<bean id="mc" class="pkg.MyController"/>

Readymade Controller classes to simplify different uses cases:

a. AbstractController --> for hyperlinks.
b. AbtsrractCommandController --> for form submission without validations. (X)
c. SimpleFormController --> for form submission with validations. (X)
d. MultiActionController --> to handler multiple hyperlinks/ multiples submits button requests.
e. AbstractWizardFormContrller --> To display and process form that is coming in chain. (X)
   and etc.

Note:

- ✓ There are the various types of MVC controllers (Only in XML driven configuration).
- ✓ (X) Removed from Spring 4.x for supporting more annotation-based programming.
- ✓ Maintenance Projects of Spring MVC XML configuration or XML +annotation configuration.
- ✓ New Projects of Spring MVC, Spring boot MVC or XML + annotations configuration.

## HandlerMapping class

- ➢ This is a java class implementing org.springframework.web.servlet.HandlerMapping (I).
- ➢ This component is useful to map incoming URI/ URLs with Handler/ controller classes i.e. it each incoming URI with one handler class and returns that handler class bean id back to DS.
- ➢ We generally do not get the need of developing user-defined Handler mappings because we have lots of readymade handler mappings classes.
- ➢ Every HandlerMapping must be configure in Spring bean configuration file.

Readymade HandlerMapping classes:

a. SimpleUrlHandlerMapping (best in xml driven Spring MVC configuration).
b. BeanNameUrlHandlerMapping (default in XML driven configuration).
c. ControllerClassNameHandlerMapping (removed from spring 4.x onwards).
d. DefaultAnnotationHanlderMapping (removed from spring 5.x given for annotation driven configuration) (default for annotation driven configuration before spring 5.x).
e. RequestMappingHandlerMapping (best for annotation driven configuration default also from Spring 5.x).

dispatcher-servelt.xml

```
<bean id="suhm"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
                <props>
                        <prop key="/home.htm">mc </prop>
                </props>
        </property>
</bean>
```

home.htm: incoming request URI
mc: handler/ controller

## ViewResolver class

- It is java class that implements org.springframework.web.servlet.ViewResolver(I).
- This comp takes logical view name from DS and gives View object (View (I) implementation class object) having the physical view name location.
- This useful to achieve loose coupling towards changing UI technology of View layer without effecting other components.
- We have lots of readymade ViewResolver to fulfill our requirements. So, no need of developing user-defined ViewResolver.
- Every ViewResolver must be configured in Spring bean configuration file as spring bean.

Readymade ViewResolver classes:

a. InternalResourceViewResolver (useful to use JSP/ Servlets comps of private area as view components).

b. UrlBasedViewResolver (allows to take any technology view component).
c. ResourceBundleViewResolver (Allows to collect info about physical view components and location from properties file).
d. XmlViewResolver (allows to collect info about physical view components and location from xml file).
e. BeanNameViewResolver (allows to take java classes view components).
f. TilesViewResolver (allows to tiles f/w for views)
and etc.

Note: In different use cases we use diff ViewResolver.

dispatcher-servlet.xml
```
<bean id="irvr"
class="org.springframework.web.sevlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/"/>
        <property name="suffix" value=".jsp"/>
</bean>
```

Received logical view name (LVN) from DS and appends prefix (location) and suffix (extension) of physical view component and returns View object having that physical view component name and location.

View object

```
/WEB-INF/pages/result.jsps
prefix           LVN   suffix
```

## Code based Flow

Controller/ Handler class
```
public class MyController implements Controller(I) {
                                              //invasive
    //Handler method
    public ModelAndView         (j)
        handleRequest(HttpServletRequest req,
            HttpServletResponse res) throws Exception{
      (k) return new
        ModelAndView("result","sysDate", new Date());
    }
}
```

web.xml (b)

```
<web-app>
          <!-- Servlet Configuration -->
          <servlet>
                    <servlet-name>dispatcher</servlet-name>
                    <servlet-class>
org.sf.web.s.DispatcherServlet</servlet-class>
                    <load-on-startup>2</load-on-startup>
          </servlet>                    (>=0 should be given)
          <!-- Servlet Mapping -->
          <servlet-mapping>
                    <servlet-name>dispatcher</servlet-name>
                    <url-pattern> *.htm </url-pattern>
          </servlet-mapping> (Extension match URL pattern)
          <welcome-file-list>        (d)
                    <welcome-file>index.jsp</welcome-file>
          </welcome-file-list>
</web-app>
```

index.jsp

```
<jsp:forward page="home.htm"/>   (e)
```

(generates implicit request to Spring MVC web app based on the URL home.htm)

/WEB-INF/pages/result.jsp (Physical view component)

```
<h1> welcome to Spring MVC </h1>
model data : ${sysDate}
                 (p)
```

Our Web Application name is:
FirstMVCAPP

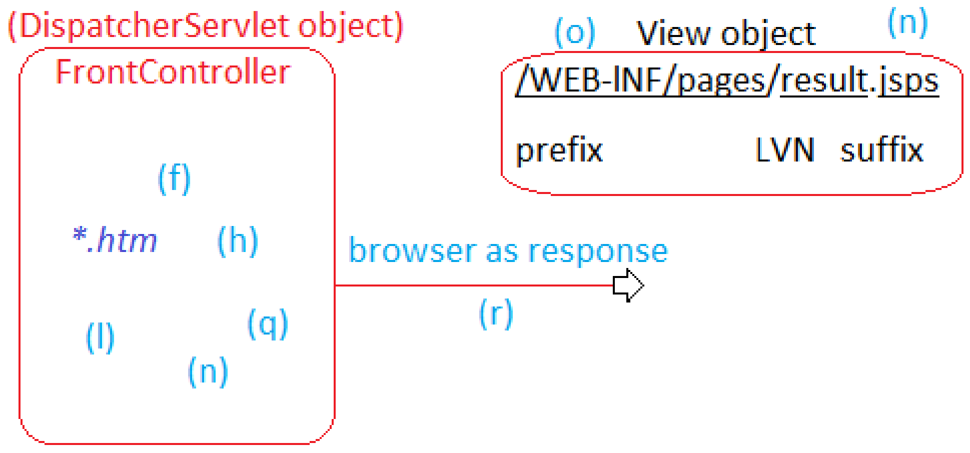
(a) Deployment

(b) DS initial operations
(all pre-instantations and initialization, injections on DS, Spring beans takes place)

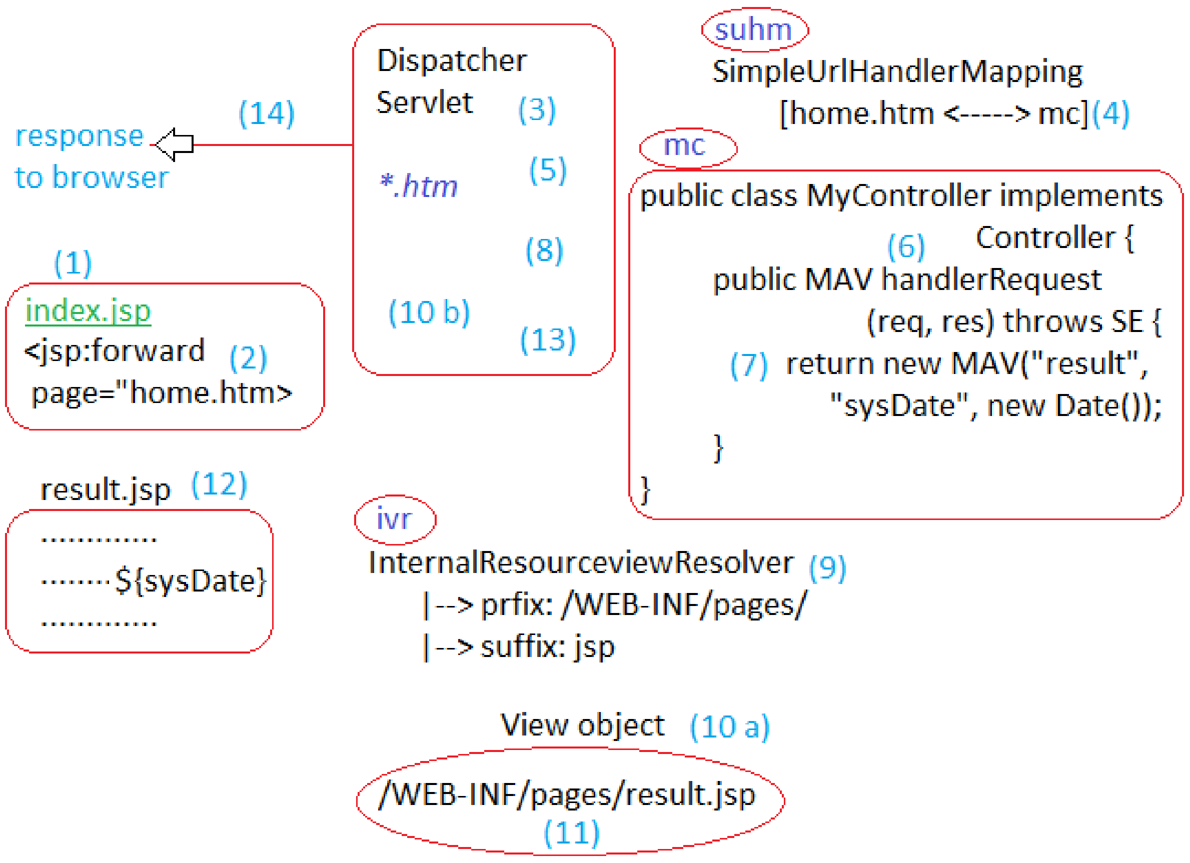
(c) request generation from browser
http://localhost:3030/FirstMVCApp

dispatcher-servlet.xml (b)

```
<!--Controller/ handller cfg-->
<bean id="mc" class="pkg.MyController"/>    (i)

<!--handler mapping cfg-->
<bean id="suhm"
class="org.springframework.web.servlet.handler.SimpleUrlH
                                                andlerMapping">
         <property name="mappings">
                  <props>          (g)
                           <prop key="/home.htm">mc </prop>
                  </props>
         </property>
</bean>

<!--View Resolver cfg-->
<bean id="irvr"                              (m)
class="org.springframework.web.sevlet.view.InternalResourceVie
wResolver">
         <property name="prefix" value="/WEB-INF/pages/"/>
         <property name="suffix" value=".jsp"/>
</bean>
```



## Spring MVC First App Story board


response to browser
(14)
Dispatcher Servlet
(3)
*.htm
(5)
(8)
(10 b)
(13)
suhm
SimpleUrlHandlerMapping
[home.htm <-----> mc](4)
mc
public class MyController implements Controller {
(6)
public MAV handlerRequest (req, res) throws SE {
(7) return new MAV("result", "sysDate", new Date());
}
}
(1)
index.jsp
<jsp:forward page="home.htm>
(2)
result.jsp
(12)
.............
........${sysDate}
.............
ivr
InternalResourceviewResolver
(9)
|--> prfix: /WEB-INF/pages/
|--> suffix: jsp
View object
(10 a)
/WEB-INF/pages/result.jsp
(11)

Note: Deployment, DS initial activities, request to web application are ignored here because there are regular in any Spring MVC application, if needed you can also involve them.

Steps to develop Spring MVC First Web application which displays the private area web component as home page:

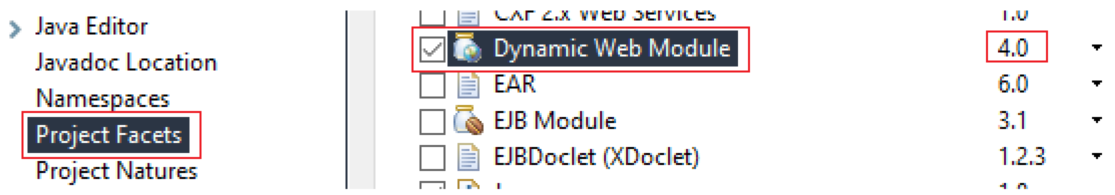
Step 1: Make sure that Tomcat server is configure with Eclipse IDE.

Step 2: Create Gradle Project.

Name: MVCProj1-FirstApp-ShowingHomePage

Step 3: Perform following operations on Gradle Project to make it as web application Project.

RTC on Project (Right click on Project) --> properties --> project facets --> select dynamic web module: 4.0



CXF 2.x Web Services 1.0
[✓] Dynamic Web Module 4.0
[ ] EAR 6.0
[ ] EJB Module 3.1
[ ] EJBDoclet (XDoclet) 1.2.3

Step 4: Add the following Jar files or dependencies to the build.gradle
spring-webmvc.<ver>.jar
servlet-api.<ver>.jar

Note: The Jars added using maven/ gradle build tool to build.gradle of the web application will be placed automatically in CLASSPATH/ BUILDPATH and also in WEB-INF/lib folder.

Step 5: Add web.xml file and configure DispatcherServlet and welcome file.
RTC on WEB-INF -->new others xml (web.xml)

Step 6: Add Spring bean configuration file as dispatcher-servlet.xml (<DS logical name>- servlet.xml) in WEB-INF folder.
RTC on WEB-INF new others Spring bean configuration file

Step 7: Create "pages" folder WEB-INF folder to make main JSP pages/ files in private area.

Note:
- ✓ “Pages” is not standard folder name. So, we can take any name.
- ✓ WEB-INF, classes, lib, web.xml names standard names and we cannot change them.

Step 8: Create packages in src/main/java folder and complete the development (with the source code).

```
src/main/java
          |--> com.nt.controller
```

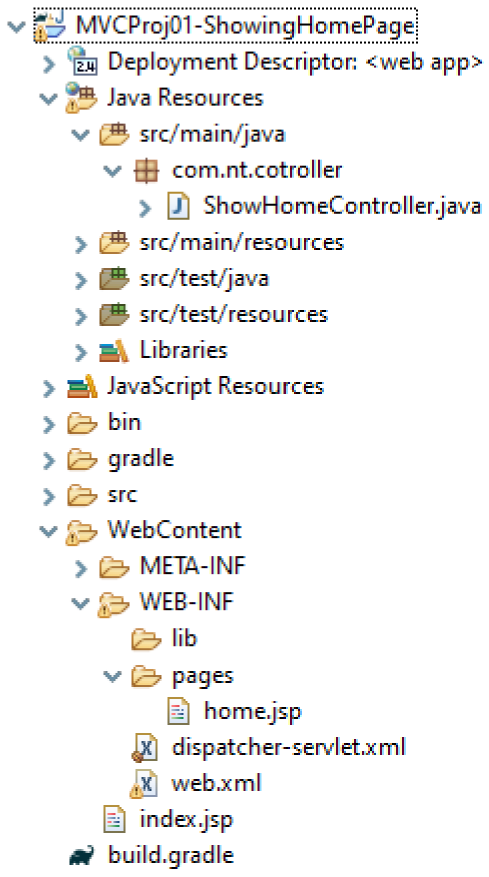
Note: Keep welcome file (index.jsp) always in public area (webcontent folder)

Step 9: Run the Application.
RTC on Project run as run on server select tomcat.

Note: In Gradle, by default webcontent folder does not participate in deployment. So, we perform deployment assembly exactly.
RTC on Project properties --> Deployment Assembly --> Add --> folder
-->webcontent

Note: Standard web application directory structure WEB-INF and its sub folders are called private area i.e., only ServletContainer can use them (not the outsiders), where the outside of Area of web WEB-INF folder (like webcontent folder data) comes under public area i.e., anyone can access that area.

## Directory Structure of MVCProj01-ShowingHomePage:


```
MVCProj01-ShowingHomePage
  Deployment Descriptor: <web app>
  Java Resources
    src/main/java
      com.nt.cotroller
        ShowHomeController.java
    src/main/resources
    src/test/java
    src/test/resources
    Libraries
  JavaScript Resources
  bin
  gradle
  src
  WebContent
    META-INF
    WEB-INF
      lib
      pages
        home.jsp
      dispatcher-servlet.xml
      web.xml
    index.jsp
  build.gradle
```

- Develop the above directory structure and folder, package, class, XML, JSP, file after convert to web application and add the jar dependencies in build.gradle file.
- Then use the following code with in their respective file.

index.jsp

```
<jsp:forward page="welcome.htm"/>
```

home.jsp

```
<%@ page isELIgnored="false"%>

<h1 style="color:red; text-align:center">WelCome to Spring MVC</h1>

<b>Model data : </b> ${sysDate}
```

build.gradle

```
plugins {
  id 'war'
}
repositories {
	jcenter()
}
dependencies {
	// https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api
	implementation group: 'javax.servlet', name: 'javax.servlet-api', 
version: '4.0.1'
	// https://mvnrepository.com/artifact/org.springframework/spring-
webmvc
	implementation group: 'org.springframework', name: 'spring-
webmvc', version: '5.2.8.RELEASE'
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
	<!-- The front controller of this Spring Web application, responsible 
for handling all application requests -->
	<servlet>
		<servlet-name>dispatcher</servlet-name>
		<servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
		<load-on-startup>2</load-on-startup>
	</servlet>

	<!-- Map all requests to the DispatcherServlet for handling -->
	<servlet-mapping>
		<servlet-name>dispatcher</servlet-name>
		<url-pattern>*.htm</url-pattern>
	</servlet-mapping>

	<welcome-file-list>
		<welcome-file>index.jsp</welcome-file>
	</welcome-file-list>
</web-app>
```

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

	<!-- Configure Handler mapping -->
	<bean id="suhm"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping
">
		<property name="mappings">
			<props>
				<prop key="welcome.htm">shm</prop>
			</props>
		</property>
	</bean>

	<!-- Configure controller -->
	<bean id="shm" class="com.nt.cotroller.ShowHomeController"/>

	<!-- Configure View Resolver  -->
	<bean id="irvr"
class="org.springframework.web.servlet.view.InternalResourceViewResolver
">
		<property name="prefix" value="/WEB-INF/pages/"/>
		<property name="suffix" value=".jsp"/>
	</bean>

</beans>
```

ShowHomeController.java

```
package com.nt.cotroller;

import java.util.Date;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class ShowHomeController implements Controller {

	@Override
	public ModelAndView handleRequest(HttpServletRequest request,
HttpServletResponse response) throws Exception {
		//return MAV object
		return new ModelAndView("home", "sysDate", new Date());
	}

}
```

We can pass inputs to Servlet comp in two ways:

a. request params (form data)
    - If the inputs are non-technical and expected from end-user through browser like name, age, gender and etc.
    - Will be stored automatically in request object.
    - Use request.getParamter(-) method to read them.
b. init params/ context params
    - If the inputs are technical inputs and expected from programmers through web.xml file like driver class name, JDBC URL, DB-user and etc.
    - Will be stored in ServletConfig object (init params), ServletContext object (context params)
    - Use cg.getInitParameter(-) or sc.getInitParameter(-) to read these values.
        i. cg --> ServletConfig object
        ii. sc --> ServletContext object.

Note:

- ✓ init params are specific to one Servlet/ JSP comp.
- ✓ Context params are common for all web components of a web application.

Note: We can take our choice name configuration file and location for that we have to done the following things.

- We have to create a folder in WEB-INF and there have IoC create a

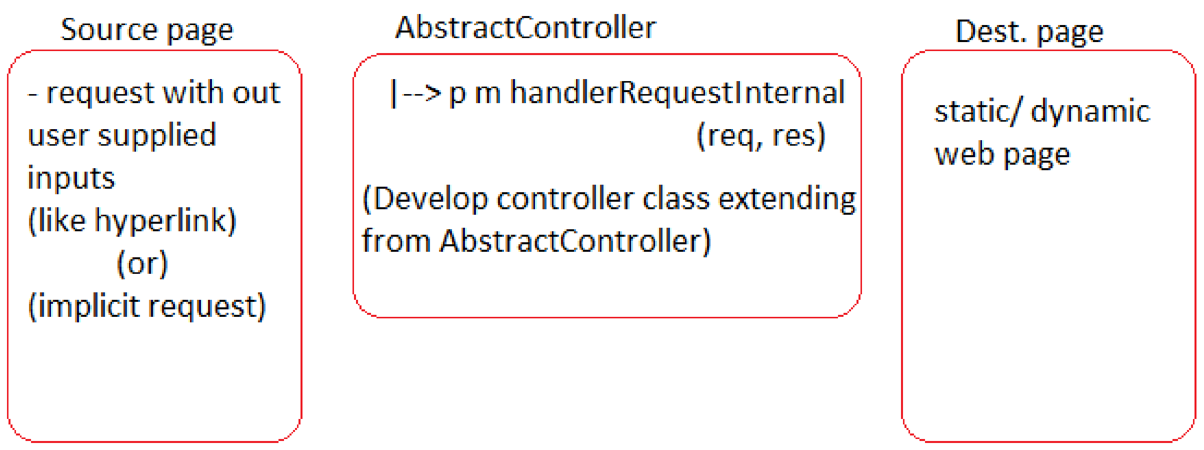
Spring bean configuration with our choice of name and we have the place the following code.

web.xml

```
        <servlet>
                <servlet-name>dispatcher</servlet-name>
                <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
                <init-param>
                        <param-name>contextConfigLocation</param-name>
                        <param-value>/WEB-INF/cfgs/mycfg.xml</param-value>
                </init-param>
                <load-on-startup>2</load-on-startup>
        </servlet>
```

Note: Fix inti param of DS to specify then name and location of Spring bean configuration file.

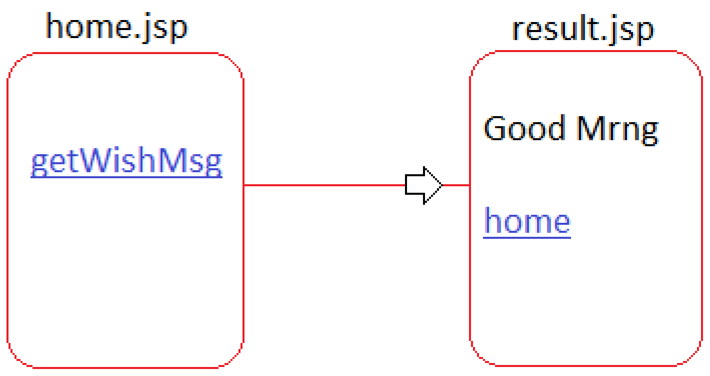
Note: In xml driven spring MVC, developing controller class directly by implementing pkg.Controller (I) is bad practice because we have different readymade controller classes for different use cases like AbstractController, MultiActionController and etc.


Source page
- request with out user supplied inputs (like hyperlink) (or) (implicit request)
AbstractController
|--> p m handlerRequestInternal (req, res)
(Develop controller class extending from AbstractController)
Dest. page
static/ dynamic web page

Use cases: listAllEmployees, getCurrentTredingjobs, getAllfriends, showCoranaReport, getWishMessage.

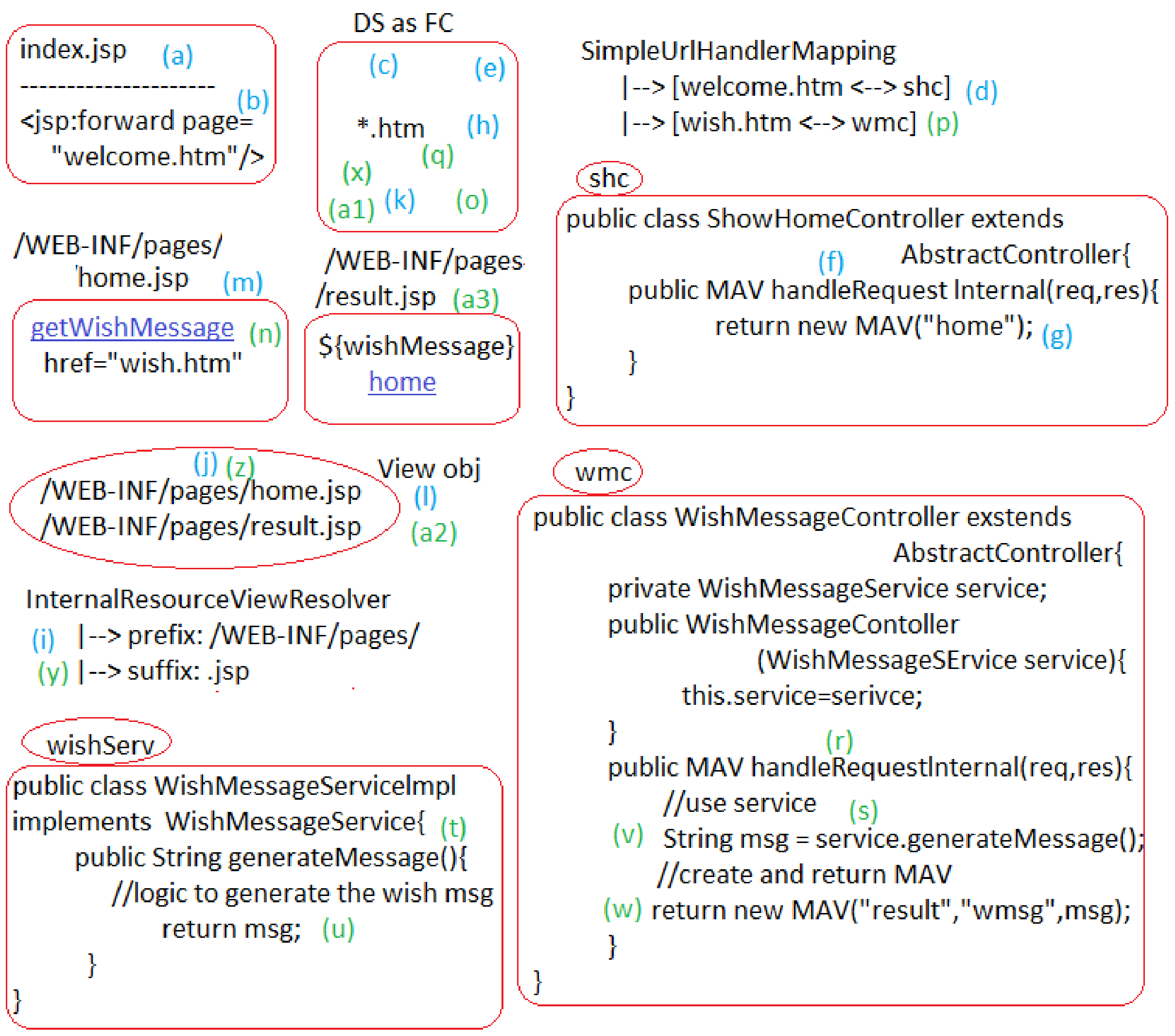
(These hyperlinks may or may not have query string supplied inputs but user-supplied inputs)

```
        <a href="wish.htrn"> get WishMessage </a> (or)
        <a href="account.htm?acno=1001">getBalance </a>
```



2 Controller (Abstract Controller)
|--> ShowHomeController
(To show home page without any request processing)
|--> WishMessageController
(To show result page with message after request processing)

## Storyboard of Wish Application using AbstractController



```
<jsp:forward page=
    "welcome.htm"/>
```

```
href="wish.htm"
```

```
public class ShowHomeController extends
                        AbstractController{
        public MAV handleRequest Internal(req,res){
                return new MAV("home");
        }
}
```

```
public class WishMessageController exstends
                              AbstractController{
        private WishMessageService service;
        public WishMessageContoller
                      (WishMessageSErvice service){
            this.service=serivce;
        }
        public MAV handleRequestInternal(req,res){
            //use service
            String msg = service.generateMessage();
            //create and return MAV
            return new MAV("result","wmsg",msg);
        }
}
```

```
public class WishMessageServiceImpl
implements  WishMessageService{
        public String generateMessage(){
            //logic to generate the wish msg
                return msg;
        }
}
```

Note:

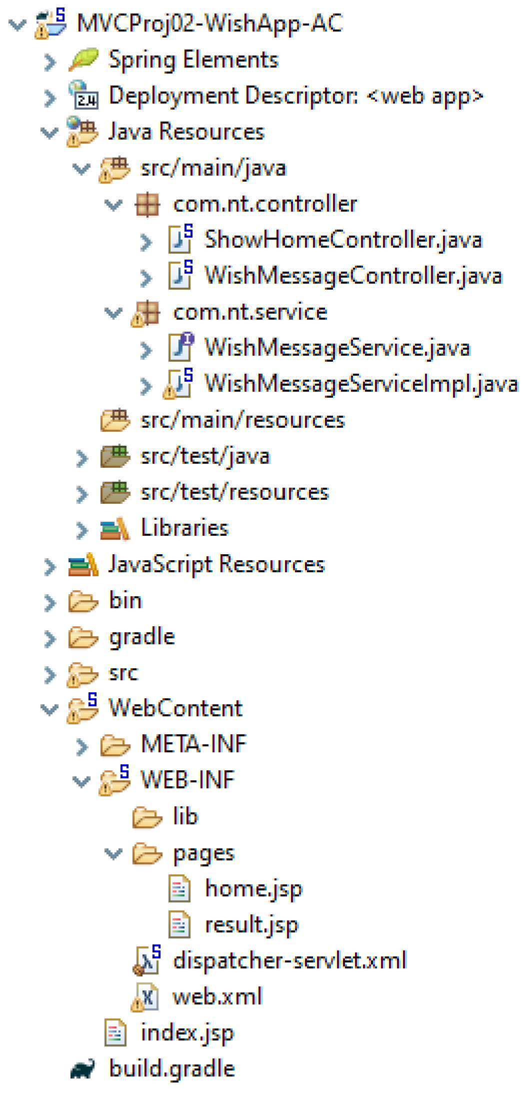
- ✓ Java 10 feature local variable type inference (compiler decides the data type based on the initial values).
- ✓ Using var we cannot go for compound variables declare, we cannot declare two variables in same line.
- ✓ var type variables cannot be initialized with null but can be initialized

simple values like 0, 0, 0 and etc.
✓ var type can be taken only for local variables.

Note: Different controller type Controller classes are having different handler methods but DS calls all those methods through handleRequest (req, res) method.

- AbstractController --> handleRqusetInternal (req, res)
- AbstractCommandController --> handle(req, res, …, …)
- SimpleFormContrller --> onSubmit(-, -, -, -)
  and etc.

## Directory Structure of MVCProj02-WishApp-AC:



```
MVCProj02-WishApp-AC
  > Spring Elements
  > Deployment Descriptor: <web app>
  v Java Resources
    v src/main/java
      v com.nt.controller
        > ShowHomeController.java
        > WishMessageController.java
      v com.nt.service
        > WishMessageService.java
        > WishMessageServiceImpl.java
      src/main/resources
    > src/test/java
    > src/test/resources
    > Libraries
  > JavaScript Resources
  > bin
  > gradle
  > src
  v WebContent
    > META-INF
    v WEB-INF
        lib
      v pages
          home.jsp
          result.jsp
        dispatcher-servlet.xml
        web.xml
      index.jsp
    build.gradle
```

- Develop the above directory structure and folder, package, class, XML, JSP, file after convert to web application and add the jar dependencies in build.gradle file.

- Copy the build.gradle from previous project we are using same.
- Then use the following code with in their respective file.

index.jsp

```
<jsp:forward page="welcome.htm"/>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
	<!-- The front controller of this Spring Web application, responsible 
for handling all application requests -->
	<servlet>
		<servlet-name>dispatcher</servlet-name>
		<servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
		<load-on-startup>2</load-on-startup>
	</servlet>
	<!-- Map all requests to the DispatcherServlet for handling -->
	<servlet-mapping>
		<servlet-name>dispatcher</servlet-name>
		<url-pattern>*.htm</url-pattern>
	</servlet-mapping>
	<welcome-file-list>
		<welcome-file>index.jsp</welcome-file>
	</welcome-file-list>
</web-app>
```

home.jsp

```
<h1 style="color: red; text-align: center">Welcome to Wish App</h1><br>
<h2 style="color: cyan; text-align: left">
	<a href="wish.htm">Click here to get te wish message</a>
</h2>
```

WishMessageService.java

```
package com.nt.service;
public interface WishMessageService {
	public String getWishMessage();
}
```

WishMessageServiceImpl.java

```
package com.nt.service;

import java.util.Calendar;

public class WishMessageServiceImpl implements WishMessageService {
	@Override
	public String getWishMessage() {
		Calendar calendar = null;
		int hours = 0;
		//get Calendar class obejct
		calendar = Calendar.getInstance();
		//get hour of the day
		hours = calendar.HOUR_OF_DAY;
		if (hours<12)
			return "Good Morning";
		else if (hours<16)
			return "Good Afternoon";
		else if (hours<20)
			return "Good Evening";
		else
			return "Good Night";
	}
}
```

ShowHomeController.java

```
package com.nt.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
public class ShowHomeController extends AbstractController {
	@Override
	protected ModelAndView handleRequestInternal(HttpServletRequest
request, HttpServletResponse response)
			throws Exception {
		return new ModelAndView("home");
	}
}
```

dispatcher-service.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"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd">

	<!-- Configure Handler mapping -->
	<bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping
">
		<property name="mappings">
			<props>
				<prop key="welcome.htm">shc</prop>
				<prop key="wish.htm">wmc</prop>
			</props>
		</property>
	</bean>

	<!-- Configure Controller -->
	<bean id="shc" class="com.nt.controller.ShowHomeController"/>

	<bean id="wmc" class="com.nt.controller.WishMessageController">
		<constructor-arg ref="wishService"/>
	</bean>

	<!-- Configure Service -->
	<bean id="wishService"
class="com.nt.service.WishMessageServiceImpl"/>

	<!-- Configure View Resolver -->
	<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver
">
		<property name="prefix" value="/WEB-INF/pages/"/>
		<property name="suffix" value=".jsp"/>
	</bean>

</beans>
```

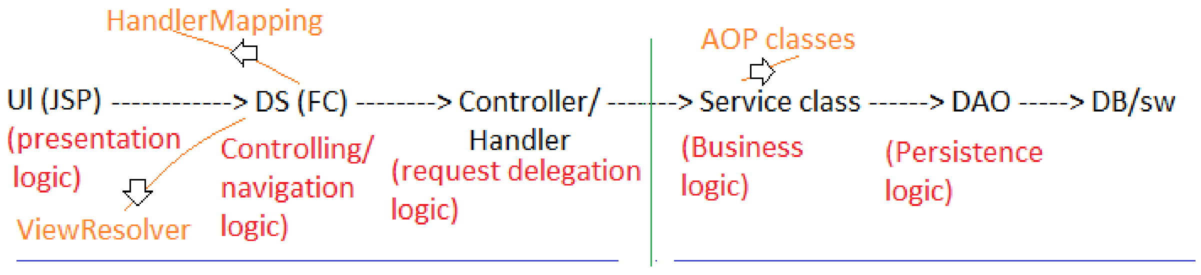
WishMessageController.java

```
package com.nt.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;

import com.nt.service.WishMessageService;

public class WishMessageController extends AbstractController {

	private WishMessageService service = null;


	public WishMessageController(WishMessageService service) {
			this.service = service;
	}

	@Override
	protected ModelAndView handleRequestInternal(HttpServletRequest 
request, HttpServletResponse response)
			throws Exception {
		String msg = null;
		//use serive
		msg = service.getWishMessage();
		return new ModelAndView("result", "wishMessage", msg);
	}
}
```

result.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1" isELIgnored="false"%>

<h1 style="color: red; text-align: center">Welcome to Wish App</h1><br>
<h2 style="color: cyan; text-align: center">
	${wishMessage}
</h2><br>

<a href="welcome.htm">Home</a>
```

## Complete Layered Application Using Spring MVC

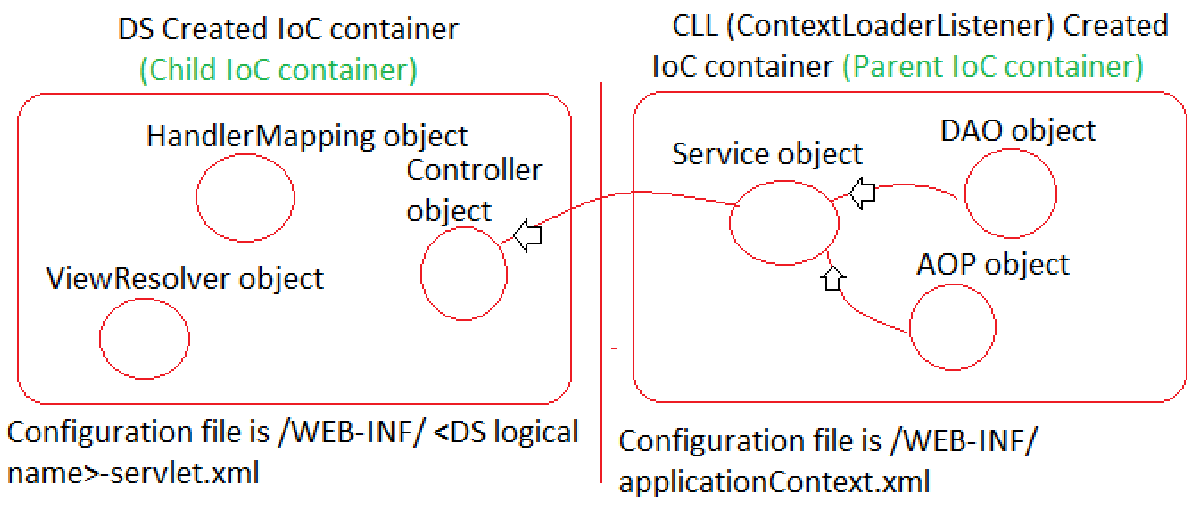

HandlerMapping
UI (JSP) ------------> DS (FC) --------> Controller/ Handler --------> Service class ------> DAO -----> DB/sw
(presentation logic)
ViewResolver
Controlling/ navigation logic)
(request delegation logic)
AOP classes
(Business logic)
(Persistence logic)

(Presentation tier components)
(All these components are there to take requests and to deliver response. So, they are called Presentation tier components)

(Business-tier components)
(All these components are there to process requests by using business logic and persistence logics like calculations, analyzations, Tx mgmt, DB operations and etc. So, these called Business-tier Components)

- We need to develop the presentation-tier, business tier comps having loose coupling i.e. the degree of Dependency should less. [This indicates we should be in a position to use Spring in each tier (presentation tier or business tier irrespective spring is used or not in another another)].
- To implement the above concept, we need to take two IOC container 1 for presentation tier components and another one for business-tier components.
- DS created IOC container for presentation tier components.
- ContextLoaderListener created IOC container for business-tier components.
- It is built-in ServletContext Listener
    - It creates IOC container when ServletContext object is created (during the deployment of web application).
      (by calling contextInitialized (-) method)
    - It stops/ closes same IoC container when ServletContext object is destroyed (When web application is stopped or reloaded/undeployed)
      [by calling contextDestoyed (-) method]

Note: The ContextLoaderListener created IOC container takes /WEB-INF/ applicationContext.xml as the default spring bean configuration file.



DS Created IoC container
(Child IoC container)
HandlerMapping object
Controller object
ViewResolver object
Configuration file is /WEB-INF/ <DS logical name>-servlet.xml
CLL (ContextLoaderListener) Created IoC container (Parent IoC container)
Service object
DAO object
AOP object
Configuration file is /WEB-INF/ applicationContext.xml

- Since we need to inject Service class object to Controller object. So, we need to take CLL created IoC container that manages business tier comps as parent IoC container and DS created IoC container that manages presentation tier comps as the child container.

Note: Parent IOC container managed Spring Bean can be injected to child IOC container managed Spring Bean, but reverse is not possible.

Objects creation order during the deployment of web application:

- Listener classes objects
- ServletContext objects
- FilterConfig objects (if Filters are configured)
- Filter objects (if filters are configured)
- ServletConfig objects (if Servlets are configured)
- <load-on-startup> enabled Servlet class objects

- XmlWebApplicationContext IoC container is self-Intelligent IoC container, because when it is created and if it notices another IoC container of same type is already available or created then it makes already available/ created IoC container as parent IoC container and current created IoC container as child container.
- Since CLL creates first IoC container of XmlWebApplicationContext and DS creates Second IoC container of same type. So, the DS created IoC container automatically becomes child IoC container by taking CLL created IoC container as parent IoC container.

Note: Some old versions of existing servers like WebLogic 8/ 9 if DS comp <load-on-startup> value 0/ 1 there is possibility of creating DS object first, CLL object due to this DS IoC container becomes parent CLL IoC container becomes Child IOC container and the whole Service object injection to Controller will be failed. So, it recommended to take <load-on-startup> value as 2, so that CLL object will always be created before DS object.

## Two Container creation flow

```
web.xml for two IOC containers
-----------------------------------------------
<web-app>
        <listener>
                <listener-class>            (b) CLL object
                        org.springframework.web.context.ContextLoaderListener
                </listener-class>
        <listener>

     |  <servlet>
     |          <servlet-name> dispatcher</servlet-name>
 (g) |          <servlet-class>
     |                  org.sf.web.mvc.servlet.DispatcherServlet
     |          </servlet-class>
     |          <load-on-startup>2 </load-on-startup>
     |  </servlet>
     |  <servlet-mapping>
                <servlet-name> dispatcher</servlet-name>
                <url-pattern>*.htm </url-pattern>
        </servlet-mapping>
</web-app>
```

(a) Deployement

(c) ServletContext Object

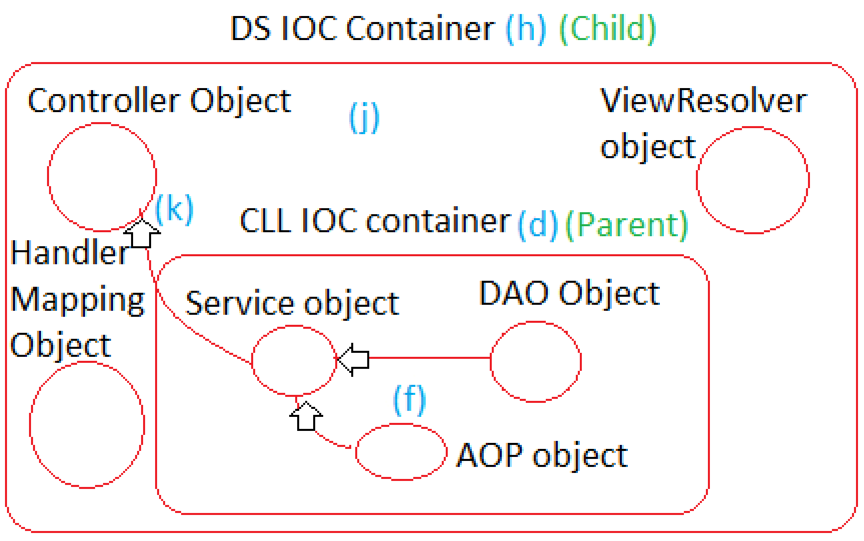
(d) CLL Create IoC container

(e) (InMemory MetaData)
/WEB-INF/applicationContext.xm
(f) Pre-instantiation of
singleton scop beans and injection

(h) DS Created IoC container

(i) (InMemory MetaData)
/WEB-INF/dispatcher-servlet.xml
(j) Pre-instantiation of
singleton scope bean and injection

(l) Keeping singleton scope Spring objects inInternal cache of both IoC Containers



Q. Here both DS and ContextLoaderListener are predefined and given by Spring framework only, when if we change presentation tier technology again, we need to develop that ContextLoaderListener class, so where is loose coupling here?

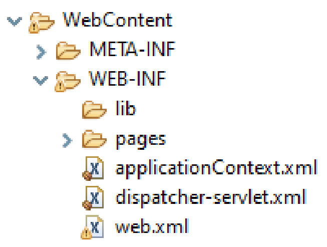
Ans. We can work with ContextLoaderListener though we are not configuring DS in web.xml because still Spring is used in business tier. So, we can still add Spring jar files.

Q. If we use non contained (non-spring) presentation technologies, how we can inject business tier objects?

Ans. Use the plugins supplied by those presentation technologies. (like JSF or only Servlet, JSP or JSP/ HTML angular and etc.

## Directory Structure of MVCProj03-WishApp-AC-TwoContainer:

- Copy paste the MVCProj02-WishApp-AC application and change rootProject.name to MVCProj03-WishApp-AC-TwoContainer in settings.gradle file
- Change the Web Project Settings to MVCProj03-WishApp-AC-TwoContainer.
- Add applicationContext.xml in Webcontent/WEB-INF/.
- Add the following code in their respective files.



```
WebContent
  META-INF
  WEB-INF
    lib
    pages
    applicationContext.xml
    dispatcher-servlet.xml
    web.xml
```

Remove the Service class configuration from dispatcher-servlet.xml file.

applicationContext.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"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd">

	<!-- Configure Service -->
	<bean id="wishService"
class="com.nt.service.WishMessageServiceImpl"/>

</beans>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>

	<!-- Bootstraps the root web application context before servlet
initialization -->
	<listener>
		<listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
	</listener>

	<!-- The front controller of this Spring Web application, responsible
for handling all application requests -->
	<servlet>
		<servlet-name>dispatcher</servlet-name>
		<servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
		<load-on-startup>2</load-on-startup>
	</servlet>

	<!-- Map all requests to the DispatcherServlet for handling -->
	<servlet-mapping>
		<servlet-name>dispatcher</servlet-name>
		<url-pattern>*.htm</url-pattern>
	</servlet-mapping>
```

```
        <welcome-file-list>
                <welcome-file>index.jsp</welcome-file>
        </welcome-file-list>

</web-app>
```

If You want to take your choice of location for ContextLoaderListener and DispatcherServlet:

```
<web-app>
        <!-- needed for ContextLoaderListener -->
        <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/business-beans.xml</param-value>
        </context-param>  [Gives to change the name and location
                        of spring bean configuration file for parent container but
                recommended and default name is /WEB-INF/applicationContext.xml]
        <!-- Bootstraps the root web application context before servlet initialization -->
        <listener>
                <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
        </listener>

        <!-- The front controller of this Spring Web application, responsible for handling
all application requests -->
        <servlet>
                <servlet-name>dispatcher</servlet-name>
                <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
                <init-param>
                        <param-name>contextConfigLocation</param-name>
                        <param-value>/WEB-INF/presentation-beans.xml</param-value>
                </init-param> [Gives to change the name and location
                <load-on-startup>2</load-on-startup> of spring bean configuration file
        </servlet> for child container but recommended and default name is
                                /WEB-INF/applicationContext.xml]
        <!-- Map all requests to the DispatcherServlet for handling -->
        <servlet-mapping>
                <servlet-name>dispatcher</servlet-name>
                <url-pattern>*.htm</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
                <welcome-file>index.jsp</welcome-file>
        </welcome-file-list>
</web-app>
```

Q. Can we take same name for Init params and context params?
Ans. Yes, Possible because init params allocate memory in ServletConfig object and they are specific to each Servlet.
Context params allocate Memory in ServletContext object and they are visible in all web components of web application, like Servlet components, JSP components, listeners, filter and etc.

## HandlerMapping

- All handler mappings are the classes implementing pkg.HandlerMapping (I).
- These are useful to map incoming request URLs to controller/ handler classes.

### Different Handler Mappings:

- BeanNameUrlHandlerMapping (default in XML driven configuration)
- SimpleUrlHandlerMapping (mostly used in XML driven configuration)
- ControllerClassNameHandlerMapping (removed from spring 5.x)
- DefaultAnnotationHandlerMapping (default in annotation driven configuration up to spring 4.x and removed in spring 5.x)
- RequestMappingHandlerMapping (default in annotation driven configuration from spring 5.x)
  and etc.

### BeanNameUrlHandlerMapping:

- It is Default, if no handler mapping is configured in XML driven configuration.
- It will map Incoming request URI with that handler/ controller who is having incoming request URI as the bean id.
- Incoming request URL is: /welcome.htm

  dispatcher-servlet.xml
  ```
  <bean class="pkg.BeanNameUrlHandlerMapping"/>
  <bean id-"/welcome.htm" class="pkg.ShowHomeController"/>
  ```

### Directory Structure of MVCProj04-WishApp-AC-HandlerMapping:

- Copy paste the MVCProj03-WishApp-AC-TwoContainer application and change rootProject.name to MVCProj04-WishApp-AC-HandlerMapping in settings.gradle file
- Change the Web Project Settings to MVCProj04-WishApp-AC-HandlerMapping.

- Need Not to add anything same as MVCProj03-WishApp-AC-TwoContainer.
- Add the following code in their respective files.

dispatcher-servlet.xml

```
<!-- Configure Handler mapping -->
        <bean
class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMap
ping"/>


        <!-- Configure Controller -->
        <bean id="/welcome.htm"
class="com.nt.controller.ShowHomeController"/>

        <bean id="/wish.htm"
class="com.nt.controller.WishMessageController">
                <constructor-arg ref="wishService"/>
        </bean>
```

ControllerClassNameHandlerMapping:

- ➢ Removed from spring 4.x and 5.x because its confusing functionality.
- ➢ Takes incoming URL removes the extension, makes first letter as upper-case letter and appends Controller then search Controller class name having that name to map the request.

  welcome.htm --- mapped to ---> WelcomeController
  home.htm --- mapped to ---> HomeController
  (Test in Spring MVC 3.x setup)

SimpleUrlHandlerMapping:

- ➢ Allows to map different incoming URIS with different controller by taking the support of "mappings" property of type "java.util.Properties"

  dispatcher-servlet.xml

```
          <bean class=
"org.springframework.web.servLet.handler.SimpLeUrLHandLerMapping”>
                    <property name= "mapping”>
                              <props>
```

```
                                <prop key="welcome.htm">shc</prop>
                        </props>
                </property>
        </bean>
        <bean id="shc" class="com.nt.controller.ShowHomeController"/>
```

dispatcher-servlet.xml

```
        <!-- Configure Handler mapping -->
        <bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping
">
                <property name="mappings">
                        <props>
                                <prop key="welcome.htm">shc</prop>
                                <prop key="wish.htm">wmc</prop>
                        </props>
                </property>
        </bean>

        <!-- Configure Controller -->
        <bean id="shc" class="com.nt.controller.ShowHomeController"/>

        <bean id="wmc" class="com.nt.controller.WishMessageController">
                <constructor-arg ref="wishService"/>
        </bean>
```

## HandlerMapping Chaining:

- If we configured multiple Handler Mappings, they will work having chaining among them i.e. if first HandlerMapping fails to locate controller class then it passes to next handler mappings in the order of configuration.

dispatcher-servlet.xml

```
        <!-- Configure Handler mapping -->
        <bean
class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMap
ping"/>
        <bean
```

```
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping
">
            <property name="mappings">
                  <props>
                        <prop key="welcome.htm">shc</prop>
                        <prop key="wish.htm">wmc</prop>
                  </props>
            </property>
      </bean>
```

- While working multiple handler mappings if any ambiguity problem comes for any incoming request URI i.e. one incoming URI is mapped with two different controllers' classes by using two different handler mappings then the 1st configured handler mapping linked controller will be picked up.
- Incoming request URI is: welcome.htm (a)

dispatcher-servlet.xml

```
      <!-- Configure Handler mapping -->
      <bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping
">                                                              |
            <property name="mappings">                          |
                  <props>                     (b)               |
                        <prop key="welcome.htm">shc</prop>      |   1st HM
                  </props>                                      |
            </property>                                         |
      </bean>                                                   |
      <bean
class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMap
ping"/>     | 2nd HM

      <!-- Configure Controller -->          (c)
      <bean id="shc" class="com.nt.controller.ShowHomeController"/>

      <bean id="/welcome.htm"
class="com.nt.controller.ShowHomeController1"/>
```

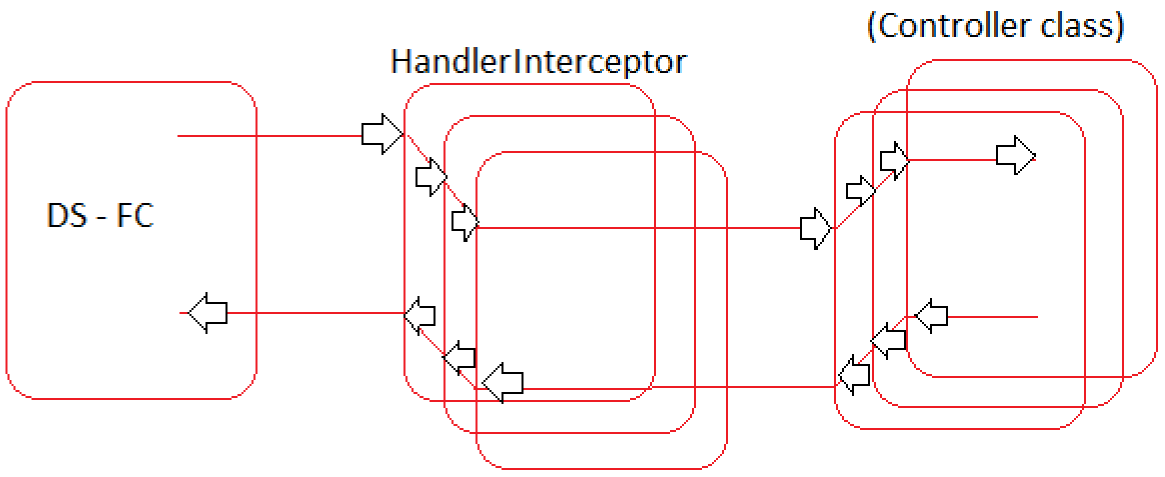
- If want give priority to another HandlerMapping, not based on order of configuration then we specify that priority order by injecting numeric value to "order" property of HandlerMapping which indicates high value indicates low priority and low value indicates high priority.
- Default order value is 65535 (Big value)
- incoming request URI: welcome.htm (a)

dispatcher-servlet.xml

```
        <!-- Configure Handler mapping -->
        <bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping
">
                <property name="mappings">
                        <props>
                                <prop key="welcome.htm">shc</prop>
                        </props>
                </property>
                <property name="order" value="2"/>
        </bean>

        <bean
class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMap
ping">
                <property name="order" value="1"/> (b) because of low
        </bean>                                     priority value

        <!-- Configure Controller -->
        <bean id="shc" class="com.nt.controller.ShowHomeController"/>

        <bean id="/welcome.htm"
class="com.nt.controller.ShowHomeController1"/> (c)
```

## HandlerInterceptor

- These are like servlet filters to controller classes.
- These are extension hooks to Controller/ Handler classes to add pre/ post logics having ability to enable or disable without the touching the source code of Controller/ Handler classes.



- If multiple interceptors are configured on Controller class/ classes. They trap request going to controller classes in the order they configured and they trap response coming from controller classes in reverse order.
- To develop Interceptor class our classes must implement spring supplied pkg.HandlerInterceptor (I) (In spring 5 it is given java 8 interface having default methods).

HandlerInterceptor methods are:

1. default void afterCompletion (req, res, Object handler, Exception ex) throws Exception
   - Executes after rendering the view component useful for clean up operations like removing data from session scopes.
2. default void postHandle (req, res, Object handler, MAV mav) throws Exception
   - Executes after executing the controller useful for post processing logics like converting model attribute values to different formats like changing number formats, currency symbols date formats and etc.
3. default boolean preHandle(req, res, Object handler) throws Exception
   - Executes before executing the controller useful for pre processing logics like checking browser type, checking timings, Authentication and etc.
   - If this method returns "true" then Controller of flow of execution will execute otherwise the controller will not execute.

Note:

✓ Up to Spring 4 Developer preferred developing handler/ controller class by extending from HandlerInterceptorAdapter class which is a predefined class implementing HandlerInterceptor (I) providing null method definitions for all the methods. Show we can override only those methods in which we are interested in.

✓ From spring 5, we can develop Interceptor class directly by implementing and HandlerInterceptor (I) because it is given Java 8 interface with default methods. So, we can override only those methods in which we are interested in.

Problem:

```
interface I1 {
        void m1 ();
        void m2 ();
        void m3 ();
}
```

```
public class App1 implements I1
        ……………..
        ...we must implement all the 3 methods.
        ……….
}
```

Solution 1: (Legacy)

```
(Adapter class)
public class I1Adapter implements I1 {
        public void m1 () {….} //null method
        public void m2 () {….} //null method
        public void m3 () {….} //null method
}
------------------------------------------------------------------------------
public class App1 extends I1Adapter {
        //no need of overriding all the 3 methods
        // we can implement only one method in which
        //are interested in
}
```

Solution 2: (Java 8 interface with default methods) (Best)

```
interface I1 {
        default void m1 () {….}
        default void m2 () {….}
        default void m3 () {….}
}
-------------------------------------------------------------------------------------
public class Appl implements I1 {
        // override that method in which you are interested in.
}
```

Note: javax.servlet.Servlet (I) is still normal Java8 interface by keeping older versions compatibility in mind.

Procedure to add Interceptor that allows request between 9 am to 5pm to Application:
Step 1: Keep any old App ready (like WishApp)
Step 2: Develop handler interceptor.
Step 3: Configure Handler class in Spring bean configuration file (dispatcher-servlet.xml)
Step 4: Link interceptors with controller classes using HandlerMapping support
Step 5: Develop that timeout.jsp (in public area, in webcontent folder outside of WEB-INF folder)

Directory Structure of MVCProj05-WishApp-AC-HandlerInterceptor:

- Copy paste the MVCProj03-WishApp-AC-TwoContainer application and change rootProject.name to MVCProj045-WishApp-AC-HandlerInterceptor in settings.gradle file.
- Change the Web Project Settings to MVCProj05-WishApp-AC-HandlerInterceptor.
- Create package com.nt.interceptor with TimingCheckingInterceptor.java file and timeout.jsp file in WebContent folder.
- Add the following code in their respective files.

dispatcher-servlet.xml

```
        <!-- Configure Handler mapping -->
        <bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping
">
                <property name="mappings">
                        <props>
                                <prop key="welcome.htm">shc</prop>
                                <prop key="wish.htm">wmc</prop>
                        </props>
                </property>
                <property name="interceptors">
                        <list>
                                <ref bean="tci"/>
                        </list>
                </property>
        </bean>
```

```
	<!-- Configure Handle interceptor -->
	<bean id="tci"
class="com.nt.interceptor.TimingCheckingInterceptor"/>
```

TimingCheckingInterceptor.java

```
package com.nt.interceptor;

import java.util.Calendar;

import javax.servlet.RequestDispatcher;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.HandlerInterceptor;

public class TimingCheckingInterceptor implements HandlerInterceptor {

	@Override
	public boolean preHandle(HttpServletRequest req,
HttpServletResponse res, Object handler)
			throws Exception {

		int hours = 0;
		RequestDispatcher rd = null;
		//get Current day hour
		hours = Calendar.getInstance().get(Calendar.HOUR_OF_DAY);

		//Checks the timing
		if (hours<9||hours>15) {
			rd = req.getRequestDispatcher("/timeout.jsp");
			rd.forward(req, res);
			return false;
		}
		else
		return true;
	}

}
```

Timeout.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<h1 style="color: red; text-align: center">Request Must be given between 9
am to 5 pm</h1>
<br>

<a href="welcome.htm">Home</a>
```

Q. Why we are configuring interceptors with Handler Mappings, why not with Controller classes directly?
Ans. We generally apply interceptors on multiple controllers. HandlerMapping component is the place where we map multiple incoming request URIS with multiple controllers. So, we prefer Handler Mappings to apply interceptors on multiple controllers.

Q. How to apply specific interceptors on specific controller classes?

Ans. Map related controller classes with their incoming request URI using separate HandlerMapping and link interceptor on controller classes using that HandlerMapping.

```
<bean class="SimpleUrlHandlerMapping">
        <property name="mappings">
                <props>
                        <prop key="home.htm">shc </prop>
                        <prop key="emps.htm">ec</prop>
                </props>
        </property>
        <property name="interceptors">
                <list>
                        <bean class="pkg.TimingChekingInterceptor"/>
                </list>                         (inner bean)
        </property>
</bean>

<bean class="SimpleUrlHandlerMapping">
        <property name="mappings">
                <props>
```

```
                    <prop key="wish.htm">shc </prop>
                    <prop key="exam.htm">ec</prop>
                </props>

            </property>
            <property name="interceptors">
                <list>
                    <bean class="BrowserChekingInterceptor"/>
                </list>                             (inner bean)
            </property>
</bean>
```

## ViewResolver

- These are given to resolve/ identify the physical view component name and location based on the given logical view name. It returns View object having the name and location of physical view component.
- All ViewResolver are implementation classes of org.sf.web.servlet.ViewResolver (I).
  e.g. InternalResourceviewResolver
  UrlBasedViewResolver
  ResourceBundleViewResolver
  XmlViewResolver
  TilesViewResolver
  and etc.

- View objects are the objects of classes implementing org.springframework.web.servlet.View (I). On this View object render (-) method will be called in order to pass the control to physical UI component.
  e.g. InternalResourceView
  JstlView
  XlstView
  TilesView
  VolocityView
  and etc.

### UrlBasedViewResolver:

➢ Capable of resolving/ identifying physical view name and location of any technology but we must configure View class explicitly for locating JSP/ Servlet component of private area configures "1nternalResourceView" or "JstlView" as the View class.

- For working Tiles environment configure "TilesView" as the view class.
- For working with freemarker environment configured "FreeMarkerView" as the view class.

```
<bean class="org.springframework.web.servlet.view.UrlBasedViewResolver">
        <property name="viewClass"
                        value="org.sf.web.servlet.InternalResource"/> (or)
        <property name="viewClass"
                value="org.springframework.web.servlet.view.JstlView"/>
        <property name="prefix" value="/WEB-INF/pages/"/>
        <property name="suffix" value=" .jsp"/>
</bean>
```

Q. What is the difference between JstlView and InternalResourceView classes?

Ans.

- InternalResourceView makes the programmer to added JSTL jar files to the WEB-INF/lib folder only when JSTL tags are used in the JSP pages otherwise not required.
- JstlView makes the programmer to added JSTL jar files (2) to the WEB-INF/lib folder irrespective of JSTL tags are used or not in JSP page.
- JstlView class extends from InternalResourceView class.

Directory Structure of MVCProj06-WishApp-AC-ViewResolver:

- Copy paste the MVCProj03-WishApp-AC-TwoContainer application and change rootProject.name to MVCProj06-WishApp-AC-ViewResolver settings.gradle file.
- Change the Web Project Settings to MVCProj06-WishApp-AC-ViewResolver.
- Add the following code in their respective files.

dispatcher-servlet.xml

```
        <!-- Configure View Resolver -->
        <bean
class="org.springframework.web.servlet.view.UrlBasedViewResolver">
                <property name="viewClass"
value="org.springframework.web.servlet.view.InternalResourceView"/>
                <property name="prefix" value="/WEB-INF/pages/"/>
                <property name="suffix" value=".jsp"/>
        </bean>
```

## Code in build.gradle to Move the WebContent folder to Deployment assembly automatically:

Problem: There is a bug in gradle web project when we go for gradle refresh the WebContent folder is removed from deployment assembly automatically to fix the bug and add the WebContent folder dynamically in deployment assembly we have to write the following lines of code in build.gradle.

Solution 1

```
eclipse {
        wtp {
                component {
                        resource sourcePath: "/WebContent", deployPath: "/"
                }
        }
}
```

Solution 2

```
apply plugin : 'eclipse-wtp'
webAppDirName='WebContent'
```

Solution 3

```
plugins {
   id 'eclipse-wtp'
}

webAppDirName='WebContent'
```

Note: If the view class not configured while working with UrlBasedViewResolver then we get org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'org.springframework.web.servlet.view.UrlBasedViewResolver#0' defined in ServletContext resource [/WEB-INF/dispatcher-servlet.xml]: Initialization of bean failed; nested exception is java.lang.IllegalArgumentException: Property 'viewClass' is required.

## JSTL:

- JSTL stands for JSP Standard tag library. It is a set of Jsp tab libraries having bunch tags in each JSP tag library. These are useful to make JSP components/ page Java code less JSP pages or script less JSP pages.

JSTL Tag libraries are,

a) core b) sql c) formatting d) xml e) functions

Note: Every JSTL JPS tag library is identity with its taglib URI which is fixed taglib URI.

| | |
|---|---|
| core (c) | http://java.sun.com/jsp/jstl/core |
| sql (sql) | http://java.sun.com/jsp/jstl/sql |
| formatting (fmt) | http://java.sun.com/jsp/jstl/fmt |
| xml (x) | http://java.sun.com/jsp/jstl/xml |
| functions (fn) | http://java.sun.com/jsp/jstl/fn |

- Add the following Jar dependencies in build.gradle and code to following files.

dispatcher-servlet.xml

```
        <!-- Configure View Resolver -->
        <bean
class="org.springframework.web.servlet.view.UrlBasedViewResolver">
                <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"/>
                <property name="prefix" value="/WEB-INF/pages/"/>
                <property name="suffix" value=".jsp"/>
        </bean>
```

build.gradle

```
        // https://mvnrepository.com/artifact/javax.servlet/jstl
        implementation group: 'javax.servlet', name: 'jstl', version: '1.2'
        // https://mvnrepository.com/artifact/taglibs/standard
        implementation group: 'taglibs', name: 'standard', version: '1.1.2'
```

result.jsp

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page isELIgnored="false"%>
<h2 style="color: cyan; text-align: center">
        <c:out value="${wishMessage}"/>
</h2>
```

Note: To JSTL tags we need import tag library URI to the jsp page using <%@taglib ... %> by specifying user-defined prefix.
prefix="c": Not fixed, so useful to differentiate the jsp tags of two different JSP tag libraries when both have same names with different functionalities
<c:out values="${wishMessage}"/>: JSTL + EL to display scoped data on to the browser.

InternalResourceviewResolver:

- Sub class of UrlBasedViewResolver but given to only to use the private are Servlet/ JSP component as view components by taking "InternalResourceView" or "JstlView" as the default view class.
- This cannot used to work with different view technologies, like Tiles, Velocity and etc. This can use only Servlet, JSP, HTML files of private area as the view components.
- Here View class configuration is optional, if JSTL jars are added then it will "JstlView" as default view class otherwise, it will " InternalResourceView" as the view class.

Note:

- ✓ Both UrlBasedViewResolver and InternalResourceviewResolver does not support ViewResolver Chaining. Even ResourceBundleViewResolver, XmlViewResolver also do not support view resolver chaining.
- ✓ BeanNameViewResolver (to take Java classes as view classes), TilesViewResolver supports ViewResolver Chaining.

- Remove the JSTL related jar dependencies from build.gradle, and JSTL related code from result.jsp and change the dispature-servlet.xml.

dispatcher-servlet.xml

```
        <!-- Configure View Resolver -->
        <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver
">
                <property name="prefix" value="/WEB-INF/pages/"/>
                <property name="suffix" value=".jsp"/>
        </bean>
```

While working UrlBasedViewResolver or InternalResourceviewResolver the limitations are,

- All view components must be there in same location having same prefix and must have same extension i.e. same technology.
- There is tight coupling LVN and physical view file name.
- All view comps should be rendered using same view class.

Note: To overcome these problems use ResourceBundleViewResolver or XmlViewResolver

## ResourceBundleViewResolver:

- Allow to configure view class name, URL separately for each logical view name (LVN) by taking the support of properties file (ResourceBundle).
- It allows to configure different technology view components for different logical view names and also to place different view components in different locations with different extensions.
- Default properties file name is views.properties (src/main/java folder). If any other name or location is taken, we must explicitly configure it.

views.properties (src/main/java)

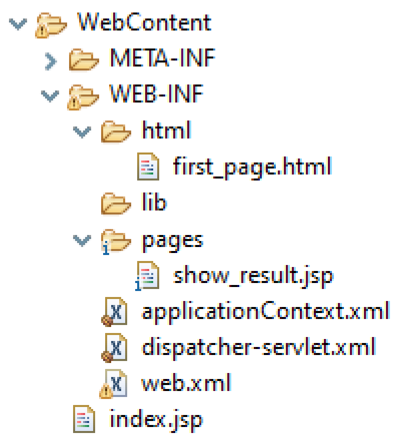
```
# <lvn>.(class) = <fully qualified view class name>
home .(class) = org.sf. web.servlet.InteranlResourceView
#<lvn>.url = name and location ,extension of physical view comp
home .url=/WEB-INF/html/l first_page.html
# add for more lvns
result. (class) =org.sf.web.servlet.JstlView
result.url=/WEB-INF/pages/show_result.jsp
```

dispatcher-servlet.xml

```
<bean class="pkg.ResourceBundleViewResolver"
            <property name="basename" value="views"/>
</bean>
```

## Directory Structure of MVCProj07-WishApp-AC-ViewResolver:

- Copy paste the MVCProj03-WishApp-AC-TwoContainer application and change rootProject.name to MVCProj07-WishApp-AC-ViewResolver settings.gradle file.
- Change the Web Project Settings to MVCProj07-WishApp-AC-ViewResolver.
- Add the following folder and file in WebContent. And add a views.properties file in src/main/java.
- Add the following code in their respective files.


```
WebContent
  META-INF
  WEB-INF
    html
      first_page.html
    lib
    pages
      show_result.jsp
    applicationContext.xml
    dispatcher-servlet.xml
    web.xml
  index.jsp
```

views.properties

```
#<lvn>.(class)=<fully qualified view class name>
home.(class)=org.springframework.web.servlet.view.InternalResourceView
#<lvn>.url=<Physical view name and location>
home.url=/WEB-INF/html/first_page.html

#<lvn>.(class)=<fully qualified view class name>
result.(class)=org.springframework.web.servlet.view.JstlView
#<lvn>.url=<Physical view name and location>
result.url=/WEB-INF/pages/show_result.jsp
```

first_page.html

```
<h1 style="color: red; text-align: center">Welcome to Wish App</h1><br>
<h2 style="color: cyan; text-align: left">
        <a href="wish.htm">Click here to get te wish message</a>
</h2>
```

show_result.jsp

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page isELIgnored="false"%>

<h1 style="color: red; text-align: center">Welcome to Wish App</h1><br>
<h2 style="color: cyan; text-align: center">
        <c:out value="${wishMessage}"/>
</h2>

<a href="welcome.htm">Home</a>
```

dispatcher-servlet.xml

```
        <!-- Configure View Resolver -->
        <bean
class="org.springframework.web.servlet.view.ResourceBundleViewResolver"
>
                <property name="basename" value="views"/>
        </bean>
```

Note: ResourceBundleViewResolver takes src/main/java/views.properties as the default properties file name and location, if not mention also it will recognize if file is there. In Normal Eclipse Dynamic web application, it is going to be src/views.properties.

XmlViewResolver:

- Same as ResourceBundleViewResolver but allows to take inputs from another xml file.
- The default name is [WEB-INF/views.xml file, if any other file name or location is taken you must specify explicitly using "location" property.

WEB-INF/views.xml (separate xml file)

```
<beans ...>       lvn
    <bean id="home" class="         View class name
        org.springframework.web.servlet.view.InternalResourceView">

        <property name="url" value="/WEB-INF/html/first_page.html"/>
                                 fixed        physical view comp name and Location
    </bean>
    <bean id="result" class="org.springframework.web.servlet.view.JstlView">
        <property name="url" value="/WEB-INF/pages/show_result.jsp"/>
    </beans>
```

dispatcher-servlet.xml

```
<bean class="org.springframework.web.servlet.view.XmlViewResolver">
        <property name="location" value="/WEB-INF/views.xml"/>
</bean>
```

- To work with XML file, we have to create the views.xml file (Spring bean configuration file) in WEB-INF location.
- Rest directory structure is same.

- And add the following code in their respective file.

views.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"
        xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd">

        <bean id="home"
class="org.springframework.web.servlet.view.InternalResourceView">
                <property name="url" value="/WEB-
INF/html/first_page.html"></property>
        </bean>

        <bean id="result"
class="org.springframework.web.servlet.view.JstlView">
                <property name="url" value="/WEB-
INF/pages/show_result.jsp"></property>
        </bean>

</beans>
```

dispatcher-servlet.xml

```
        <!-- Configure View Resolver -->
        <bean
class="org.springframework.web.servlet.view.XmlViewResolver">
                <property name="location" value="/WEB-INF/views.xml"/>
        </bean>
```

Note: We can avoid giving extension words in the incoming URIs if we configured DS with '/" URL pattern. This make DS and FrontController cum Default Servlet i.e. if not one is taking this will trap and take the request.

web.xml

```
<web-app>
        <!-- The front controller of this Spring Web application, responsible
for handling all application requests -->
```

```
        <servlet>
                <servlet-name>dispatcher</servlet-name>
                <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
                <load-on-startup>2</load-on-startup>
        </servlet>
        <!-- Map all requests to the DispatcherServlet for handling -->
        <servlet-mapping>
                <servlet-name>dispatcher</servlet-name>
                <url-pattern>/</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
                <welcome-file>index.jsp</welcome-file>
        </welcome-file-list>
</web-app>
```

Incoming request URIs in dispatcher-servlet.xml

dispatcher-servlet.xml

```
        <!-- Configure Handler mapping -->
        <bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping
">
                <property name="mappings">
                        <props>
                                <prop key="welcome">shc</prop>
                                <prop key="wish">wmc</prop>
                        </props>
                </property>
        </bean>
```

## Limitations of XML driven Spring MVC Application development:

- Controller/ Handler classes are invasive.
- Handler methods signatures are fixed though we are not using most of the params like req, res we need to take them.
- Handler method return type is ModelAndView object (fixed). Though we want give only " LVN" still we need to create complete MAV object.
- We cannot develop single controller class for multiple uses cases by placing multiple handler methods.

## Annotation driven Spring Web MVC

### Web application development:

Step 1: Take controller classes as normal java classes having @Controller annotations and link them with Spring bean configuration file using <context:compent-scan ..../>.
Step 2: Add handler methods with flexible signature (return type, method name, params are flexible) and add @RequestMapping annotation on the top of handler method specifying incoming request URI/ URL.
Step 3: Configure DefaultAnnotationHandlerMapping (up to 4.x) or RequestMappingHandlerMapping (from 5.x) as HandlerMapping in Spring bean configuration file (If not configured also they become default)
Step 4: Configure your choice ViewResolver.

### Sample Controller class:

ShowHomeController.java

```
package com.nt.controller;

@Controller
public class ShowHomeController {

	@RequestMapping("/welcome")
	public String showHomePage () {
		return "home";
	}

}
```

dispatcher-servlet.xml

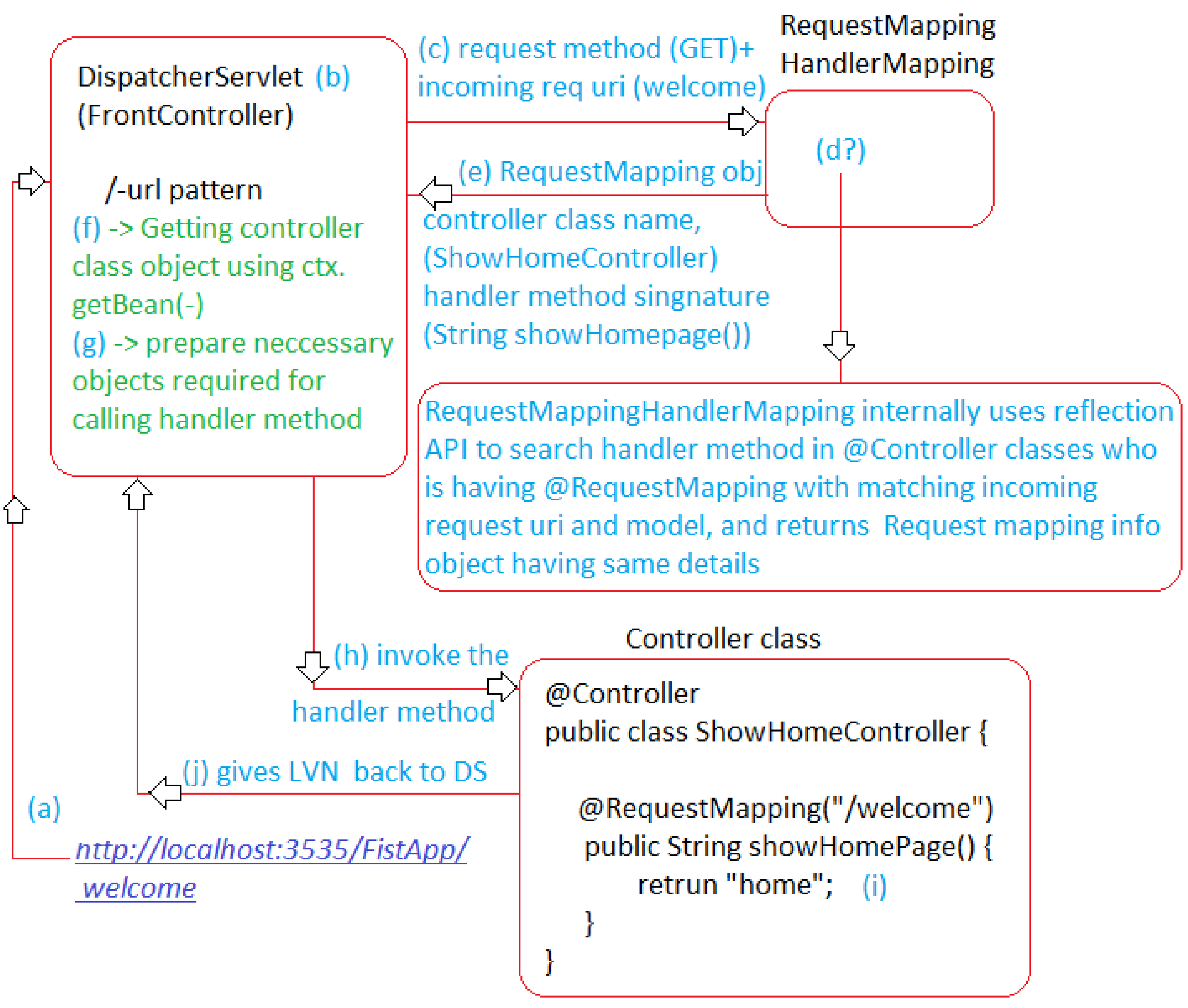
```
<beans .... >
	<! -- Mapping -->
	<bean class=" RequestMappingHandlerMapping" />

	<! -- View resolver -->
	<bean class="pkg.InternalResourceViewResolver">
		<property name="prefix" value="/WEB-INF/pages/"/>
		<property name="suffix" value=" .jsp"/>
	</bean>

	<context:component-scan base-package="com.nt.controller"/>
</beans>
```

Thumb rule in Annotation driven programming:

a. User-defined class using Stereo type annotations and link them with Spring bean configuration file using <context:componet-scan … />.
b. Pre-defined classes using <bean> tags of Spring bean configuration file.

Annotations:
@Controller: To mark java class as Spring bean cum web controller/ handler class
@RequestMapping: To mark java method of controller as handler method and also to map incoming request URL/ URI with handler method.

## Annotation code-based flow



DispatcherServlet (b)
(FrontController)
/-url pattern
(f) -> Getting controller class object using ctx. getBean(-)
(g) -> prepare neccessary objects required for calling handler method
(c) request method (GET)+ incoming req uri (welcome)
RequestMapping HandlerMapping
(d?)
(e) RequestMapping obj
controller class name, (ShowHomeController)
handler method singnature (String showHomepage())
RequestMappingHandlerMapping internally uses reflection API to search handler method in @Controller classes who is having @RequestMapping with matching incoming request uri and model, and returns Request mapping info object having same details
(h) invoke the handler method
Controller class
(j) gives LVN back to DS
(a)
http://localhost:3535/FistApp/welcome
(i)

```
@Controller
public class ShowHomeController {

    @RequestMapping("/welcome")
    public String showHomePage() {
        retrun "home";
    }
}
```

Note: The handler methods of controller classes in Annotation driven Spring MVC can have flexible signatures however there are list of allowed parameter types and allowed return types (i.e. we cannot take other than them)

## Possible Parameter types or Method types:

- WebRequest, NativeWebRequest
- javax.servlet.ServletRequest, javax.servlet.ServletResponse
- javax.servlet.http.HttpSession

  and etc.

  Note:
  - Most of them are given by keeping restful web services in mind.
  - For more parameter type refer this [reference]

## Possible Return types:

- @ResponseBody
- HttpEntitY<B>, ResponseEntitY<B>
- String
- View
- Void
- @ModelAttribute
- java.util.Map, org.springframework.ui.Model

  and etc.

  Note:
  - Most of them are given by keeping restful web services in mind.
  - For more return types refer this [reference]

## Annotation driven Spring MVC App development advantages:

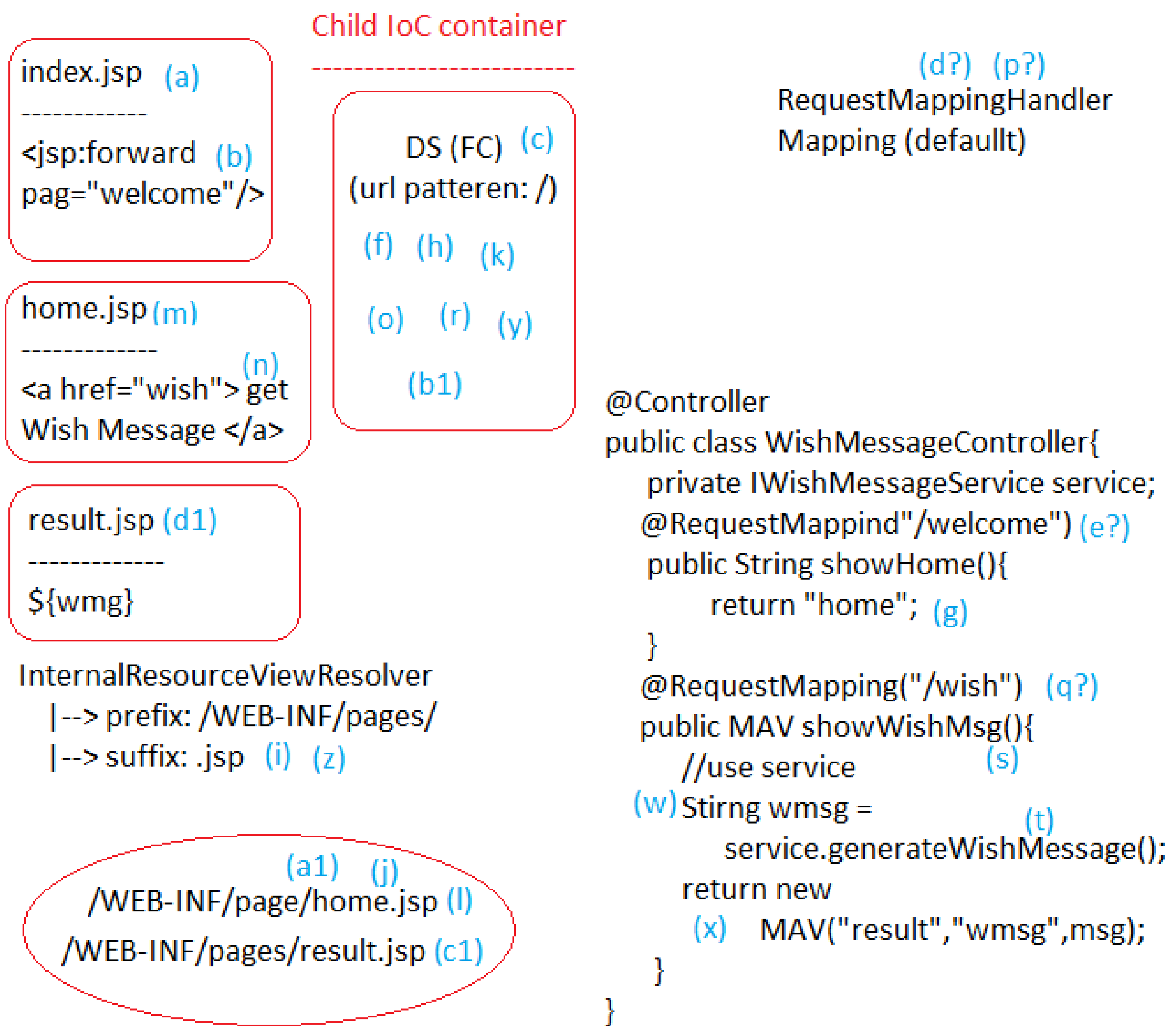
a. Controller classes are non-invasive.
b. One Controller class can have multiple handler methods. Generally, we can take 1 controller class with multiple handler methods representing multiple related uses-cases incoming request URIs.
c. Handler methods are having flexible signatures.

   and etc.

## In Spring MVC Annotation driven configuration:

- JSPs (as needed)
- DS (1 per project)
- Controller class (1 per related use-cases)
- Service class (1 per module)
- DAO (1 per Db table or 1 per related DB tables)

## Story board of WishApp Web Application using Annotation driven configuration

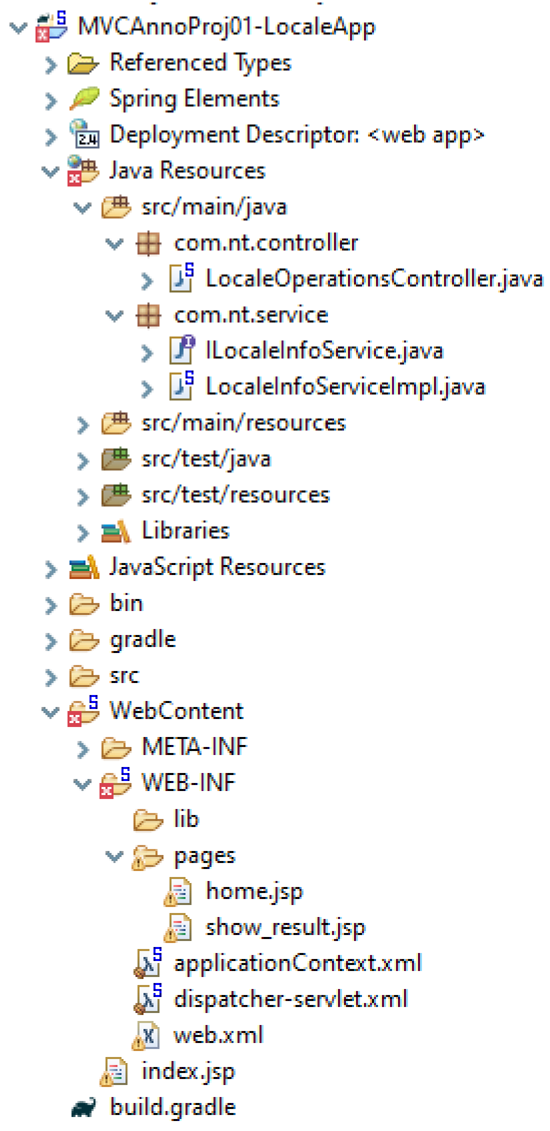

Child IoC container
index.jsp (a)
<jsp:forward (b) pag="welcome"/>
DS (FC) (c)
(url patteren: /)
(f) (h) (k)
(o) (r) (y)
(b1)
(d?) (p?)
RequestMappingHandler Mapping (defaullt)
home.jsp (m)
<a href="wish"> (n) get Wish Message </a>
result.jsp (d1)
${wmg}
InternalResourceViewResolver
|--> prefix: /WEB-INF/pages/
|--> suffix: .jsp (i) (z)
(a1) (j)
/WEB-INF/page/home.jsp (l)
/WEB-INF/pages/result.jsp (c1)

```
@Controller
public class WishMessageController{
     private IWishMessageService service;
    @RequestMappind"/welcome") (e?)
     public String showHome(){
          return "home";  (g)
     }
    @RequestMapping("/wish")  (q?)
    public MAV showWishMsg(){
        //use service              (s)
 (w) Stirng wmsg =                   (t)
             service.generateWishMessage();
        return new
     (x)    MAV("result","wmsg",msg);
     }
}
```

Parent IoC container

```
@Service("wishService")
public class WishMessageSerivceImpl implements
                                   (u)      IWishMessageService{
        public String genateWishMessage(){
              ..............
              ......................
              retun msg;   (v)
        }
}
```

## Directory Structure of MVCAnnoProj01-LocaleApp:



```
MVCAnnoProj01-LocaleApp
    Referenced Types
    Spring Elements
    Deployment Descriptor: <web app>
    Java Resources
        src/main/java
            com.nt.controller
                LocaleOperationsController.java
            com.nt.service
                ILocaleInfoService.java
                LocaleInfoServiceImpl.java
        src/main/resources
        src/test/java
        src/test/resources
        Libraries
    JavaScript Resources
    bin
    gradle
    src
    WebContent
        META-INF
        WEB-INF
            lib
            pages
                home.jsp
                show_result.jsp
            applicationContext.xml
            dispatcher-servlet.xml
            web.xml
        index.jsp
    build.gradle
```

- Develop the above directory structure and folder, package, class, XML, JSP, file after convert to web application and add the jar dependencies in build.gradle file then use the following code with in their respective file.
- Copy the build.gradle file form previous project because we are using same jars with JSTL and Standard jar.

Note: If red mark in folder or directory level need not to worry go ahead, if you get red mark at file level then solve first then run the project.

### index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<jsp:forward page="welcome"/>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
	<!-- Bootstraps the root web application context before servlet
initialization -->
	<listener>
		<listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
	</listener>
	<!-- The front controller of this Spring Web application, responsible
for handling all application requests -->
	<servlet>
		<servlet-name>dispatcher</servlet-name>
		<servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
		<load-on-startup>2</load-on-startup>
	</servlet>
	<!-- Map all requests to the DispatcherServlet for handling -->
	<servlet-mapping>
		<servlet-name>dispatcher</servlet-name>
		<url-pattern>/</url-pattern>
	</servlet-mapping>
	<welcome-file-list>
		<welcome-file>index.jsp</welcome-file>
	</welcome-file-list>
</web-app>
```

applicationContext.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"
	xsi:schemaLocation="http://www.springframework.org/schema/bea
ns https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
		http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-
4.3.xsd">
	<context:component-scan base-package="com.nt.service"/>
</beans>
```

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"
        xsi:schemaLocation="http://www.springframework.org/schema/bea
ns https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
                http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-
4.3.xsd">

        <!-- Configure Handler mapping -->
        <bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestM
appingHandlerMapping"/>

        <!-- Configure View Resolver -->
        <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolve
r">
                <property name="prefix" value="/WEB-INF/pages/"/>
                <property name="suffix" value=".jsp"/>
        </bean>

        <context:component-scan base-package="com.nt.controller"/>

</beans>
```

home.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<h1 style="color: red; text-align: center">Welcome to Locale App</h1>
<h1 style="color: red; text-align: center">Home</h1>

<h2 style="color: cyan; text-align: left">
        <a href="countries">All Countries</a>
</h2>
```

ILocaleInfoService.java

```
package com.nt.service;

import java.util.Set;

public interface ILocaleInfoService {

	public Set<String> getAllCountries();

}
```

LocaleInfoServiceImpl.java

```
package com.nt.service;

import java.util.Locale;
import java.util.Set;
import java.util.TreeSet;

import org.springframework.stereotype.Service;

@Service("localeService")
public class LocaleInfoServiceImpl implements ILocaleInfoService {

	@Override
	public Set<String> getAllCountries() {
		Locale locales[] = null;
		Set<String> countriesSet = null;
		//get all locales
		locales = Locale.getAvailableLocales();
		countriesSet = new TreeSet<>();
		//copy all contries to List collection
		for (Locale l : locales) {
			if (!l.getDisplayCountry().equals("")) {
				countriesSet.add(l.getDisplayCountry());
			}
		}
		return countriesSet;
	}

}
```

show_result.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1" isELIgnored="false"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>

<h1 style="color: red; text-align: center">Welcome to Locale App</h1>
<h1 style="color: red; text-align: center">Result</h1>

<c:choose>
        <c:when test="${!empty listInfo && listInfo ne null}">
                <c:forEach var="country" items="${listInfo}">
                        <table bgcolor="pink" align="center">
                                <tr>
                                        <td style="color: blue;">${country}</td>
                                </tr>
                        </table>
                </c:forEach>
        </c:when>
        <c:otherwise>
                <h1 style="color:red; text-align: left;">No countries
found</h1>
        </c:otherwise>
</c:choose>
<h3>Countries count : </h><c:out value="${fn:length(listInfo)}"></c:out>
<br>
<a href="welcome">Home</a>
```

LocaleOperationsController.java

```
package com.nt.controller;

import java.util.Set;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import com.nt.service.ILocaleInfoService;
```

```
@Controller
public class LocaleOperationsController {

        @Autowired
        private ILocaleInfoService service;

        @RequestMapping("/welcome")
        public String showHomePage() {
                return "home";
        }

        @RequestMapping("/countries")
        public ModelAndView fetchCountries() {
                Set<String> countriesSet = null;
                ModelAndView mav = null;
                //use Service
                countriesSet = service.getAllCountries();
                //add  object to mav object
                mav = new ModelAndView();
                mav.addObject("listInfo", countriesSet);
                mav.addObject("operation", "contries");
                mav.setViewName("show_result");
                //retrun MAV
                return mav;
        }

}
```

Different type of Signature of RequestMapping methods:

```
        @RequestMapping("/countries")
        public String fetchCountries(Map<String, Object> map) {
                Set<String> countriesSet = null;
                //use Service
                countriesSet = service.getAllCountries();
                //add  object to mav object
                map.put("listInfo", countriesSet);
                map.put("operation", "contries");
                return "show_result";
        }
```

```
    @RequestMapping("/countries")
    public String fetchCountries(ModelMap map) {
            Set<String> countriesSet = null;
            //use Service
            countriesSet = service.getAllCountries();
            map.put("listInfo", countriesSet);
            map.put("operation", "contries");
            return "show_result";
    }
```

- If the handler method is not giving logical view name (like method return type is void) then the incoming request URI/ request path of that handler method will be taken as lVN automatically. For this internally use RequestToViewNameTranslator.

```
    @RequestMapping("/countries")
    public void fetchCountries(Model model) {
            Set<String> countriesSet = null;
            //use Service
            countriesSet = service.getAllCountries();
            //add  attribute to model
            model.addAttribute("listInfo", countriesSet);
            model.addAttribute("operation", "countries");
    }
```

- Here “countries” will be taken a s LVN, so looks for /WEB-INF/pages/countries.jsp as Physical view name and location.

Note: To pass model data + LVN to Ds, don’t take MAV as the parameter of handler method take it as the return type of handler method.

```
    @RequestMapping("/countries")
    public void fetchCountries(ModelAndView mav) {
            Set<String> countriesSet = null;
            //use Service
            countriesSet = service.getAllCountries();
            //add  object to mav object
            mav.addObject("listInfo", countriesSet);
            mav.addObject("operation", "countries");
            mav.setViewName("show_result");
    }
```

- MAV is not part of possible parameters types for handler method.

```
	@RequestMapping("/countries/add")
	public void fetchCountries(Model model) {
		Set<String> countriesSet = null;
		//use Service
		countriesSet = service.getAllCountries();
		//add  object to mav object
		model.addAttribute("listInfo", countriesSet);
		model.addAttribute("operation", "countries");
	}
```

Problem in the below signature:

- We need to create Map object manually to add model data.
- No controller on LVN, since multiple words are in the request path (countries/add) then it takes “add” LVN and “countries” as the folder name overall it looks for “/WEB-INF/pages/countries/add.jsp”.

```
	@RequestMapping("/countries/add")
	public Map<String, Object> fetchCountries() {
		Set<String> countriesSet = null;
		Map<String, Object> map = new HashMap<>();
		//use Service
		countriesSet = service.getAllCountries();
		//add  object to mav object
		map.put("listInfo", countriesSet);
		map.put("operation", "countries");
		return map;
	}
```

Note: Taking Model/ ModelMap/ Map as the return type of the handler method is having following limitations. So not recommended to take as them as return type.

a. We should create their objects manually in order to add model data.
b. Sometime getting their implementation class name or sub class names to create object is bit complex.
c. We don’t get control on LVN more ever request path of handler method becomes LVN.

Note: Its recommended to take return type same as the parameter type of the handler method.

a. Ds will create those objects. So, our handler method just receives them as parameter to keep model data.
b. Need not to know Implementation/ sub class names of the above return type to create the object.
c. By taking return type as String, we can get control on LVN

Best Signature for Handler method:

```
@RequestMapping("/countries/add")
public String fetchCountries(Model model) { //(BEST)
        Set<String> countriesSet = null;
        //use Service
        countriesSet = service.getAllCountries();
        //add  object to mav object
        model.addAttribute("listInfo", countriesSet);
        model.addAttribute("operation", "countries");
        return "show_result";
}
```

## Rules to develop Controller/ Handler class

a. Class must be public class (To make IOC container loading and instantiating the class).
b. We must add @Controller annotation to make it as Spring bean cum web controller.
c. We can take multiple public methods as handler methods by adding @RequestMapping annotations by specifying request PATH and request mod/ method (GET, POST Spring MVC).
d. Multiple handler methods cannot Same PATH same controller class.

```
@Controller
public class MyController {
        @ResquestMapping("/welcome")
        public void showHome1() {
                ........                         //invalid
        }
        @ResquestMapping("/welcome")
        public void showHome2() {
                ........
        }
} //exception is java.lang.IllegalStateException: Ambiguous mapping
```

e. Request PATHs as case-sensitive.

```
@Controller
public class MyController {
        @ResquestMapping("/WELCOME”)
        public void showHome() {
                ……….
        }                                        //valid
        @ResquestMapping("/welcome”)
        public void showHome() {
                ………
        }
}
```

f. Different handler methods of a controller can have same request PATH with different methods/ modes (1 for GET, 1 for POST).

```
@Controller
public class MyController {
        @ResquestMapping(value="/welcome",
                                        method=RequestMethod.GET)
        public void showHome() {
                ………
        }                                        //vaild
        @ResquestMapping(value="/welcome",
                                method=RequestMethod.POST)
        public void showHome() {
                ………..
        }
}
```

g. One handler method can have multiple request PATHs

```
@Controller
public class MyController {
        @ResquestMapping(value={"/welcome", "/login", "/home"})
        public void showHome() {
                ……….
        }                               //valid
}
```

h. If no request path is specified in @RequestMapping the default will be "/" and default request mode/ method is "GET".

```
@Controller
public class MyController {
            @ResquestMapping //her path is "/" request mode is "GET"
            public void showHome() {
                        ………..
            }
}
```

equals to

```
@Controller
public class MyController {
            @ResquestMapping(value="/")
            public void showHome() {
                        ……….
            }
}
```

- "/" indicates that the handler method takes initial request that comes to home page/ welcome page without any request PATH (i.e. handler method becomes default handler method)

Note:

- ✓ Spring MVC is given to develop web application having capability to take requests from browser. Browser can give only two modes/ methods of request they are GET, POST mode.
- ✓ HTTP supports actually 8 modes requests from different types of clients.
- ✓ Browser as client, can send only "GET", "POST" mode requests.
- ✓ All the 8 methods/ modes are request are GET, POST, HEAD, PUT, DELETE, TRACE, PATCH, OPTIONS, CONNECT (reserved).
- ✓ While working with Restful webservices we will use all the 8 modes or max mode requests, because as web service component it should allow requests from different types of clients (not just browser).

ERROR message:

- 404 error: If the request resource is not found (incoming URL not matching with handler method request path).
- 405 error: Incoming request URL is matching with handler method

request path but request mode/ method (GET/ POST) not matching.
- 500 error: Any Exception in the execution.

i. Q. At max how many handler methods of a controller class can have same request paths with different request modes/ methods? (Spring MVC setup)
   Ans. As of now (2) 1 with GET mode, 1 with POST mode

j. Q. Can we take two handler methods of a controller class with default request path ("/")?
   Ans. Yes, one with GET mode and another one with POST mode.

```
@RequestMapping
public String showHomePage1() {
        System.out.println("TestControler.showHomePage1()");
        return "home_page";
}
@RequestMapping(method = RequestMethod.POST)
public String showHomePage2() {
        System.out.println("TestControler.showHomePage2()");
        return "home_page";
}
```

k. Q. Can we configure one handler method with different modes/ methods request?
   Ans. Yes,

```
@RequestMapping(value = "/first", method = {RequestMethod.GET,
                                            RequestMethod.POST})
public String showHomePage1() {
        System.out.println("TestControler.showHomePage1()");
        return "home_page";
}
```

l. Q. If two handler methods of two Controller classes are having same request PATH and mode with different functionalities then how can we request them? (Very important)

Problem code or Invalid code

```
@Controller
pubic class StudentController {
        @RequestMapping("/register")
```

```
          publicString saveData(){
                    ................
          }
}

Controller
public class EmployeeController{
          @RequestMapping("/register")
          public String saveEmp(){
                    ..........
          }
}
```

Valid code or Solution Code

```
@Controller
@RequestMapping("/student") //Global path, Controller path
pubic class StudentController {
          @RequestMapping("/register")
          publicString saveData(){
                    ................
          }
}

@Controller
@RequestMapping("/employee")
public class EmployeeController{
          @RequestMapping("/register")
          public String saveEmp(){
                    ..........
          }
}
```

JSP/ HTML/ UI code

```
<a href="student/reqister">registr student</a>
<a href="emplovee/reqister">registr employee</a>
```

Note:

- ✓ [student/register goes to saveData() of StudentController].
- ✓ [employee/register goes to saveEmp() of EmployeeController]

m. Q. How can we chain one handler method with another method? (or) How can we redirect one handler method request another handler method?

```
@Controller
public class CRUDOperationController {
        @RequestMpping("/remove")
        public String deleteStudent() {
                .... //some code to delete record
                return "redirect:display";
        }
        @RequestMpping("/display")
        public String displayStudent() {
                .... //some code to all record
                return "report_page";
        }
        @RequestMpping("/modify")
        public String updateStudent() {
                .... //some code to update reccord
                return "redirect:display";
        }
        @RequestMpping("/register")
        public String insertStudent() {
                .... //some code to update reccord
                return "redirect:display";
        }
}
```

Note:

- ✓ Here Source and destination handler methods can be there either in the same controller or in different controller class.
- ✓ redirect:<uri> internally uses response.sendRedirect(-) method i.e. request goes to next handler method of same or different controller class after having one network round trip with browser.

n. New Annotations are given from spring 4.x as alternate @RequestMapping annotation.

- @GetMapping(value="/first" ) -> /first+GET mode equals to @RequestMapping(value="/first" method=RequestMethod.GET)
- @PostMapping(value="/first" ) -> /first+POST mode equals to @RequestMapping(value="/first" method=RequestMethod.POST)

- Other @PutMapping, @DeleteMapping and etc. are there but they related to Restful webservices.

```
@GetMapping(value="/welcome")
public String showHome() {
		return "home";
}
```

Note:

- ✓ @XxxMapping annotation given spring 4.x are method level annotations i.e. they cannot be applied at controller class level for giving global path.
- ✓ We cannot apply multiple spring 4.x @XxxMapping annotations on same handler method then only first one will be taken.

```
@PostMapping("/first")
@GetMapping("/first")
public String showHomePage1() {
		System.out.println("TestController.showHomePage1()");
		return "home_page";
}
```

takes /first with POST mode request and gives error for /first with GET mode request.

```
<form action="first" method="POST"> <!-- (success)
		<input type="submit" value="submit">
</form>
<a href="first">go</a> <!-- GET mode request (error 405) -->
```

Note: If you want to inject ServletContainer created objects to Controller class like ServletConfig, ServletContext, request, response and etc. We can go @Autowired/ @Inject/ @Resource or we can take as parameters of handler method.

```
@Controller
public class TestController {
		@Autowired
		private ServletContext sc; // 1 per web application
		@Autowired
		private ServletConfig cg; // 1 per Servlet DS
```

```
        //@RequestMapping(value="/first", method= RequestMethod.GET)
        @GetMapping("/first")
        public String showHomePage1(HttpServletRequest req
                                        HttpServletResponse res) {
                System.out.println("TestController.showHomePage1()");
                System.out.println(sc.getClass());
                System.out.println(cg.getClass());
                System.out.println(req.getClass());
                System.out.println(res.getClass());
                return "home_page";
        }
}
```

## Data Rendering to View

- Passing model Data from Controller Class 'to View comps through DS
  a. Using ModelAndView object (request scope)
  b. Using Map object (request scope)
  c. Using ModelMap object (request scope)
  d. Using Model object (request Scope) (Best)
  e. Using request object (request scope)
  f. Using Session object (Session scope)
  g. Using ServletContext object (application scope)

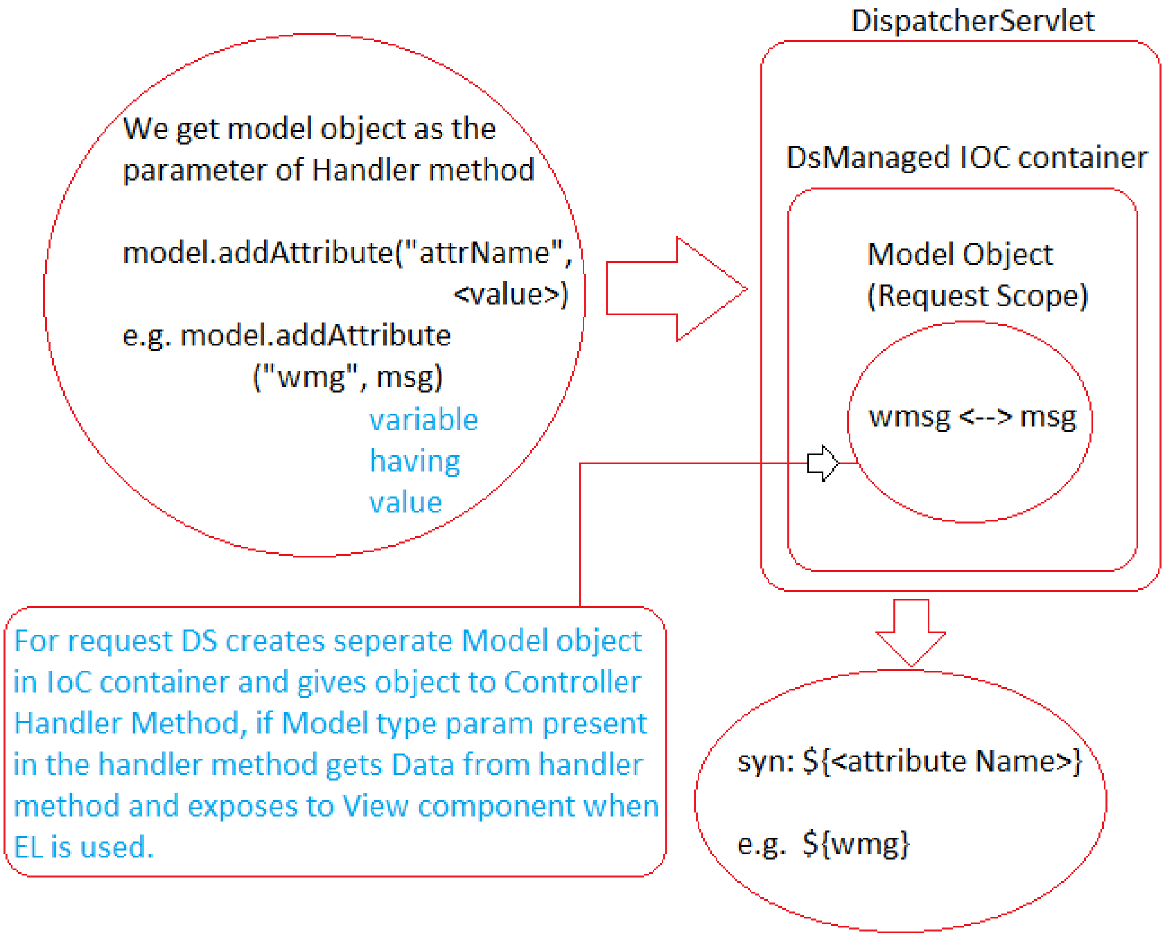
Note:

- ✓ Since request to request inputs are different so generated outputs will also be different. So, Data rendering is always good through request scope.
- ✓ f & g are bad because of their scope usage of Servlet API.
- ✓ e is because ad makes to use Servlet API.
- ✓ b, c &d from these three we can use anything in our code because all pointing same implementation BindingAwareModelMap.
- ✓ Map<k, v> bit best because of non-invasive nature.
- ✓ a is old style and not good.
- ✓ Jsp pages can read and use that model data either using EL or JSTL +EL.

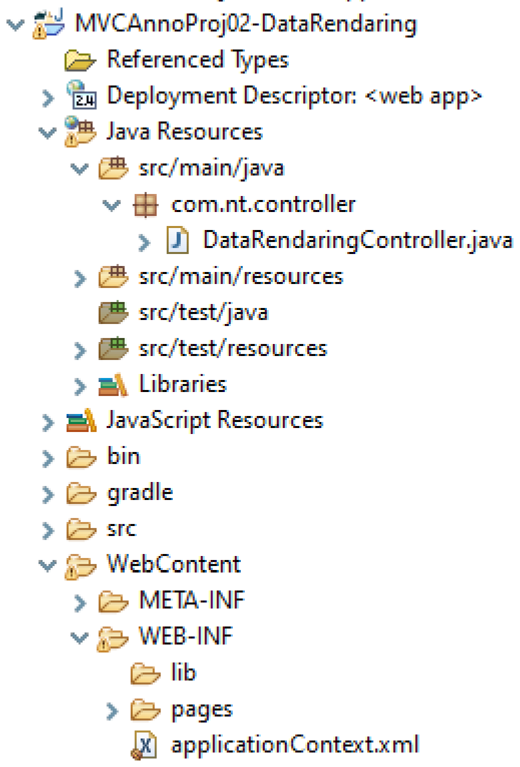
Misc.:

HashMap object initial capacity: 16 (elements)
Load factor: 0.75 i.e. 16 * 0.7*12 (Threshold number)
Once 12 elements are added the HashMap capacity become double 16+16=32.

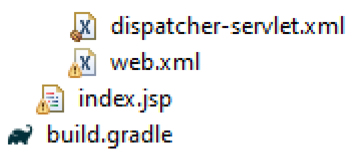
## Using Model Object:


DispatcherServlet
We get model object as the parameter of Handler method
model.addAttribute("attrName", <value>)
e.g. model.addAttribute ("wmg", msg)
variable having value
DsManaged IOC container
Model Object (Request Scope)
wmsg <--> msg
For request DS creates seperate Model object in IoC container and gives object to Controller Handler Method, if Model type param present in the handler method gets Data from handler method and exposes to View component when EL is used.
syn: ${<attribute Name>}
e.g. ${wmg}

## Directory Structure of MVCAnnoProj02-DataRendering:



```
MVCAnnoProj02-DataRendaring
    Referenced Types
    Deployment Descriptor: <web app>
    Java Resources
        src/main/java
            com.nt.controller
                DataRendaringController.java
        src/main/resources
        src/test/java
        src/test/resources
        Libraries
    JavaScript Resources
    bin
    gradle
    src
    WebContent
        META-INF
        WEB-INF
            lib
            pages
            applicationContext.xml
```



```
    dispatcher-servlet.xml
    web.xml
  index.jsp
build.gradle
```

- Develop the above directory structure and folder, package, class, XML, JSP, file after convert to web application and add the jar dependencies in build.gradle file then use the following code with in their respective file.
- Copy the build.gradle, web.xml, dispatcher-servlet.xml, index.jsp, applicationContext.xml from previous project.

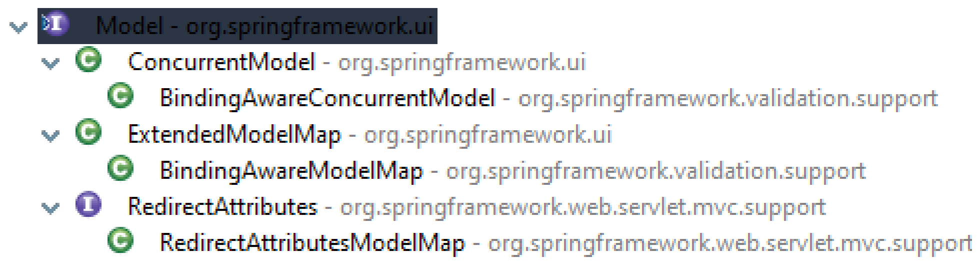
DataRenderingController.java

```java
package com.nt.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class DataRendaringController {

	@GetMapping("/welcome")
	public String processData(Model model) {
		System.out.println("DataRendaringController.processData()");
		System.out.println(model.getClass());
		model.addAttribute("msg", "Welcome Here");
		return "display";
	}

}
```

display.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false"%>
  <h1 style="color: red; text-align: center;">${msg}</h1>
```

Note: The Spring MVC takes BindingAwareModelMap as the default implementation class of Model (I) and this having capability of allocation Memory in only for required elements. Without bringing load factor into picture.


```
Model - org.springframework.ui
    ConcurrentModel - org.springframework.ui
        BindingAwareConcurrentModel - org.springframework.validation.support
    ExtendedModelMap - org.springframework.ui
        BindingAwareModelMap - org.springframework.validation.support
    RedirectAttributes - org.springframework.web.servlet.mvc.support
        RedirectAttributesModelMap - org.springframework.web.servlet.mvc.support
```

## Working with Arrays, Collections as the model data:

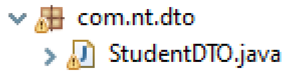
DataRenderingController.java

```
@GetMapping("/welcome")
public String processData(Model model) {
        String name[] = new String[] {"raja", "rani", "suresh"};
        List<String> fruitList = new ArrayList<>();
        fruitList.add("apple");
        fruitList.add("banana");
        fruitList.add("graps");
        Set<Long> phoneSet = new HashSet<>();
        phoneSet.add(45678234L);
        phoneSet.add(45678235L);
        phoneSet.add(45678236L);
        phoneSet.add(45678236L);
        Map<String, Integer> ageMap = new HashMap<>();
        ageMap.put("raja", 30);
        ageMap.put("rani", 25);
        ageMap.put("suresh", 33);
        //add them i model attribute
        model.addAttribute("namesInfo", name);
        model.addAttribute("fruitsInfo", fruitList);
        model.addAttribute("phonesInfo", phoneSet);
        return "display";
}
```

display.jsp

```
@GetMapping("/welcome")
public String processData(Map<String, Object> map) {
        System.out.println("DataRendaringController.processData()");
        map.put("msg", "Welcome Here");
        return "display";
}
```

## Working with collection having Java bean like below:



- Develop the above page and class in src/main/java folder on the above project and add Lombok API the build.gradle then using the following code in their respective files.

StudentDTO.java

```
@AllArgsConstructor @Data
public class StudentDTO implements Serializable {
        private int sno;
        private String sname;
        private String sadd;
}
```

DataRenderingController.java

```
        @GetMapping("/welcome")
        public String processData(Model model) {
                List<StudentDTO> listDTO = new ArrayList<>();
                listDTO.add(new StudentDTO(101, "raja", "hyd"));
                listDTO.add(new StudentDTO(102, "rani", "delhi"));
                listDTO.add(new StudentDTO(103, "suresh", "vizg"));
                //add them i model attribute
                model.addAttribute("studentDetails", listDTO);
                return "display";
        }
```

display.jsp

```
<b>Student Details : </b>
<c:choose>
        <c:when test="${studentDetails ne null && !empty studentDetails}">
                <table border="1" align="center">
                        <tr> <th>sno</th><th>sname</th><th>sadd</th></tr>
                        <c:forEach var="student" items="${studentDetails}">
                                <tr>
                                        <td>${student.sno}</td>
                                        <td>${student.sname}</td>
                                        <td>${student.sadd}</td>
                                </tr>
                        </c:forEach>
                </table>
        </c:when>
</c:choose>
```

Using Map object:

DataRenderingController.java

```
    @GetMapping("/welcome")
    public String processData(Map<String, Object> map) {
        System.out.println("DataRendaringController.processData()");
        System.out.println(map.getClass());
        map.put("msg", "Welcome Here");
        return "display";
    }
```

display.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1" isELIgnored="false"%>
  <h1 style="color: red; text-align: center;">${msg}</h1>
```

Note: This also internally use BindingAwareModelMap class as the implementation class of Map<String, Object> (I). So, we can also use this instead of model object

Using ModelMap object:

DataRenderingController.java

```
    @GetMapping("/welcome")
    public String processData(ModelMap map) {
        System.out.println("DataRendaringController.processData()");
        System.out.println(map.getClass());
        map.put("msg", "Welcome Here");
        return "display";
    }
```

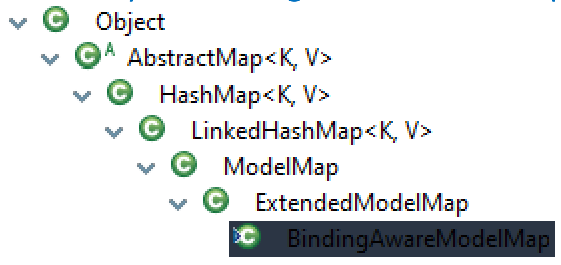
Note: By default, ModelMap reference variable also points to BindingAwareModelMap.

Q. What is difference b/w Model(I), ModelMap(C) and Map<String, Object> in Data Rendering (passing data to JSP page from Controller)?
Ans.

| Model (I) | ModelMap (c) | Map<String, Object> (Bit good) |
|---|---|---|
| a. It is spring specific interface, so invasive. | a. It is spring specific class, so invasive | (a) Map is java.util pkg interface, so it is Non-invasive. |
| b. No need of working with Generics, so wrong key types can be avoided. | b. No need of working with Generics, so wrong key types can be avoided. | b. Generally, need of working with Generics, so wrong key types can be taken. |
| c. We can call only addAtttribute(-,-) method to add model attributes not the put(-,-). | c. we can call both addAtttribute(-,-), put(-,-) method to add model Attributes. | c. we can call only put(-,-) method to add model attributes. |

Note: In Spring 5.x all are using BindingAwareModelMap (c) as the implementation class or sub class object to maintain model data.

Hierarchy of BindingAwareModelMap class:


```
Object
  AbstractMap<K, V>
    HashMap<K, V>
      LinkedHashMap<K, V>
        ModelMap
          ExtendedModelMap
            BindingAwareModelMap
```

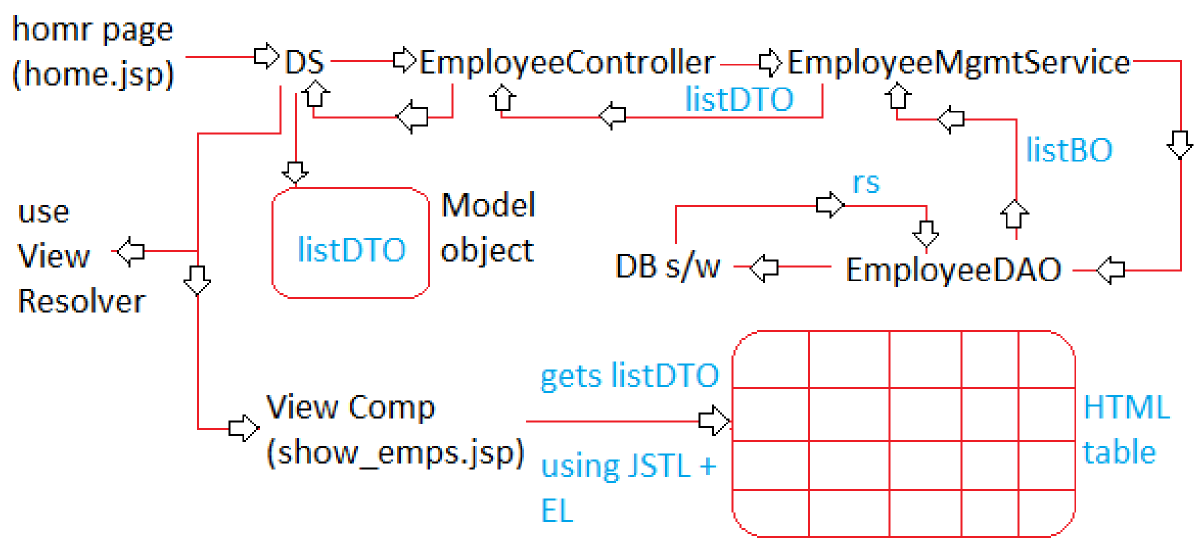
(This class implementing both Model (I), Map<k, v> (I) indirectly)

Q. What the difference b/w calling put(-,-) or addAtribute(-,-) method to add model data/attributes?
Ans.

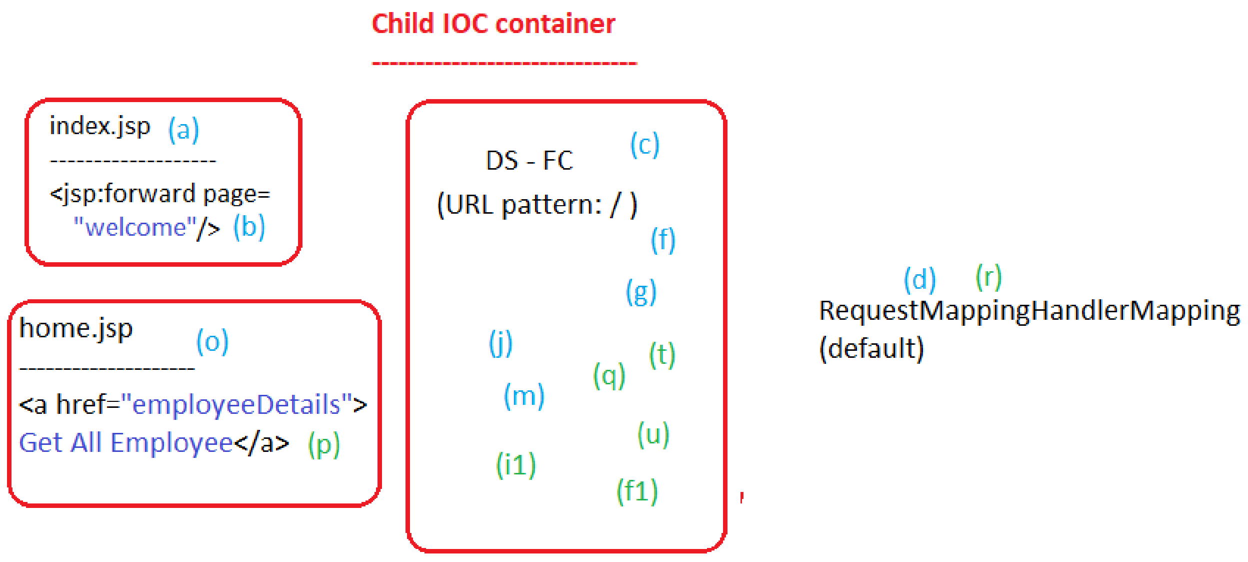
| put(-,-) | addAttribute(-,-) |
|---|---|
| a. BindingAwareModelMap class put(-,-) method calls Directly HashMap object put(-,-) method. | a. Calls HashMap put(-,-) through the put(-,-) of BindingAwareModelMap class indirectly. |
| b. Does not check for "null" value by using Assert.NotNull(-,-) method on | b. Checks for null on model attribute name by calling Assert.NotNull(-,-) |

| Model Attribute name. | method. |
|---|---|
| c. returns Object, so method chaining is not possible towards adding more model attributes.<br>e.g. map.put("msg", "welcome") .put("date", new Date()); (not possible) X | c. returns ModelMap or its sub class objects, so method chaining possible toward adding more model attributes.<br>e.g. model.addAttribute("msg", "welcome").addAttribute("sysDate", new Date()); (possible). |
| d. Invokable only on Map<k, v> object, ModelMap object. | d. Invokable only on Model object, ModelMap object. |

## Layered application using Annotation Driven Spring MVC



## Storyboard



```
@Controller
public class EmployeeController {

     @Autowired
     private IEMployeeMgmtService service;

    @GetMapping("/welcome")  (e)
    public String showHome() {  (h)
          return "home";  (i)
    }

    @GetMapping("/employeeDetails")  (s)                    (v)
    public String getAllEmployeeDetails(Model model) {
          //use service
    (d1)  List<EmployeeDTO> listDTO =          (w)
                         service.fetchAllEmployee();
          //add to model object
          model.addAttribute("employeeDTO", listDTO);
          return "display_employee";  //LVN  (e1)
    }

}
```

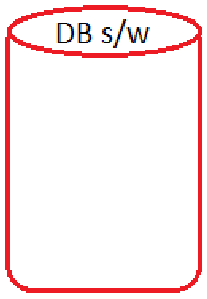
```
                                                              View object   (l)   (h1)
InternalResourceViewResolver    (k)   (g1)      /WEB-INF/pages/home.jsp  (n)
     |--> prefix: /WEB-INF/pages/               /WEB-INF/pages/display_employee.jsp
     |--> suffix: .jsp                                                   (j1)
```

```
display_employee.jsp
-----------------------------------
   .............
   ............. Code for display the
   ${employeeDTO} using JSTL and EL

  <a href="welcome">Home</a>
```

## Parent IOC container

```
public interface IEmployeeMgmtService {
        public List<EmployeeDTO> fetchAllEmployee();
}

@Service("empService")
public class EmployeeMgmtServiceImpl implements
IEmployeeMgmtService {

        @Autowired
        private IEmployeeDAO dao;
                                                (x)
        @Override
        public List<EmployeeDTO> fetchAllEmployee() {
              //use DAO                                (y)
         (b1) List<EmployeeBO> listBO = dao.retrieveAllEmployee();
              //Convert listBO to listDTO
                   ...........
              return listDTO; (c1)
        }
}
```

```
public interface IEmployeeDAO {
        public List<EmployeeBO> fetchAllEmployee();
}
@Repository("empDAO")
public class EmployeeDAOmpl implements IEmployeeDAO {

         private static final String GET_EMPlOYEE_DETAILS = "SELECT
EMPNO, ENAME, JOB, SALARY FROM EMP";

        @Autowired
         private JdbcTemplate jt;

        @Override                          (z)
        public List<EmployeeBO> fetchAllEmployee() {
              //execute the query
               .......................
               // Convert ResultSet to  listBO
              return listBO;   (a1)
         }
}
```



Q. Why cannot we take single java bean class as Model class for layer to carry Data? Why we need separate DTO, BO classes?
Ans.

1. BO/ Entity class/ Domain class objects should have only persistable or persistent data. Sometime we need extra properties to hold like serialNo, grossSalary, netSalary and etc. extra data, so take separate DTO class.
2. Sometime we need to roundup or round down values or we need correct values before giving to DB or after collecting from DB this needs separate DTO.
3. Spring Data, Spring ORM, HB and etc. ORM environment have got ability to generate DB table dynamically based on DB class/Entity class properties, keeping extra properties in BO class itself may create additional columns in DB table.

```
<!-- Get ServerManaged JDBC DataSoure object from JNDI registry of
Underlying Server -->
        <bean id="jofb"
class="org.springframework.jndi.JndiObjectFactoryBean">
                <property name="jndiName" value="java:/comp/env/DsJndi"/>
          </bean>
```

JndiObjectFactoryBean is pre-defined ServiceLocator given by Spring API having capability to get DataSource object from underlying Server Managed Jndi registry based on the given Jndi name (java:/comp/env/DsJndi) and makes that received DataSource object (Resultant object) as the Spring Bean in IoC container.

Note: To create Server Managed JDBC connection pool for oracle in Tomcat server of Eclipse IDE add the following <Resource > tag under <context> of context.xml file

context.xml

```
<Resource auth="Container"
driverClassName="oracle.jdbc.driver.OracleDriver" maxIdle="10"
maxTotal="100" maxWaitMillis="90000" name="DsJndi"
password="manager" type="javax.sql.DataSource"
url="jdbc:oracle:thin:@127.0.0.1:1521:xe" username="system"/>
```

.

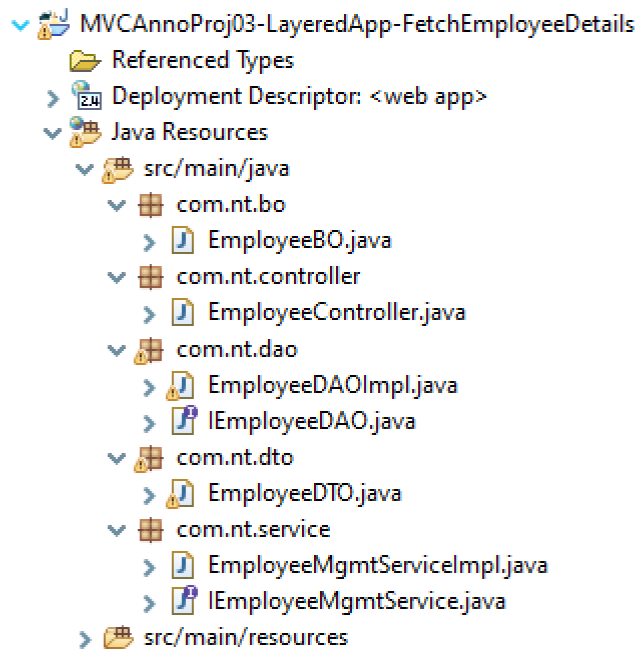
Note:

- ✓ Spring JDBC automates entire persistence logic but we can customize that persistence logic in the place we want by taking the support of Callback interfaces Callback interfaces related callback methods exposes the JDBC objects as params. we take those params/ JDBC objects and we customize the logics as we want.
  e.g. RowMapper to customize single record manipulation.
  ResultsetExtractor to customize multiple records manipulations and etc.

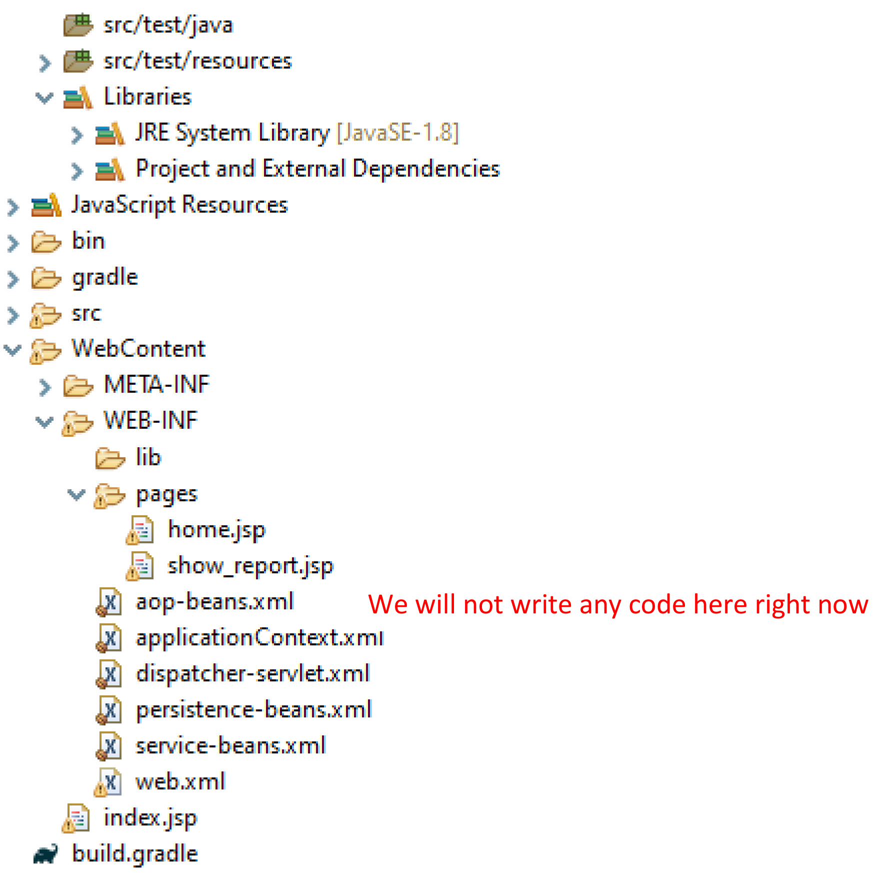
- ✓ The callback methods of callback interfaces will be called automatically by JdbcTemplate methods by passing the internally JDBC objects as arguments. So, that we can use those objects for customizing persistence logic by writing JDBC code. like convert RS single record BO object and converting RS multiple records List<BO> objects.

  List<EmployeeBO> listBO = jt.query(***GET_ALL_EMPLOYEES***, **new** EmployeeRowMapper());
  collects DS --> gets Connection object from pool creates --> PS having given Query --> executes Query gets RS --> takes given EmployeeRowMapper object --> calls extractData(RS) method on that -> gets ListBO having EmployeeBO object --> returns same listBO back as return value.

## Directory Structure of MVCAnnoProj03-LayeredApplication-FetchEmployeeDetails:



```
MVCAnnoProj03-LayeredApp-FetchEmployeeDetails
    Referenced Types
  > Deployment Descriptor: <web app>
  Java Resources
    src/main/java
      com.nt.bo
        > EmployeeBO.java
      com.nt.controller
        > EmployeeController.java
      com.nt.dao
        > EmployeeDAOImpl.java
        > IEmployeeDAO.java
      com.nt.dto
        > EmployeeDTO.java
      com.nt.service
        > EmployeeMgmtServiceImpl.java
        > IEmployeeMgmtService.java
    > src/main/resources
```



```
    src/test/java
  > src/test/resources
  v Libraries
    > JRE System Library [JavaSE-1.8]
    > Project and External Dependencies
> JavaScript Resources
> bin
> gradle
> src
v WebContent
  > META-INF
  v WEB-INF
      lib
    v pages
        home.jsp
        show_report.jsp
      aop-beans.xml          We will not write any code here right now
      applicationContext.xml
      dispatcher-servlet.xml
      persistence-beans.xml
      service-beans.xml
      web.xml
    index.jsp
  build.gradle
```

Develop the above directory structure and folder, package, class, XML, JSP, file after convert to web application and add the jar dependencies in build.gradle file then use the following code with in their respective file.

build.gradle

```
plugins {
   id 'war'
   id 'eclipse-wtp'
}

webAppDirName="WebContent"
repositories {
   jcenter()
}

dependencies {
         // https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api
         implementation group: 'javax.servlet', name: 'javax.servlet-api',
version: '4.0.1'
         // https://mvnrepository.com/artifact/org.springframework/spring-
webmvc
```

```
	implementation group: 'org.springframework', name: 'spring-webmvc', version: '5.2.8.RELEASE'
	implementation group: 'org.projectlombok', name: 'lombok', version: '1.18.12'
	implementation group: 'org.springframework', name: 'spring-jdbc', version: '5.2.8.RELEASE'
	implementation group: 'javax.servlet', name: 'jstl', version: '1.2'
	implementation group: 'com.oracle.database.jdbc', name: 'ojdbc6', version: '11.2.0.4'
}
```

index.jsp

```
<jsp:forward page="welcome"/>
```

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

	<import resource="persistence-beans.xml"/>
	<import resource="service-beans.xml"/>
	<import resource="aop-beans.xml"/>
</beans>
```

service-beans.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"
	xsi:schemaLocation="http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
		http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context-4.3.xsd">

	<context:component-scan base-package="com.nt.service"/>

</beans>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
	<listener>
		<listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
	</listener>

	<servlet>
		<servlet-name>dispatcher</servlet-name>
		<servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
		<load-on-startup>2</load-on-startup>
	</servlet>
	<servlet-mapping>
		<servlet-name>dispatcher</servlet-name>
		<url-pattern>/</url-pattern>
	</servlet-mapping>
	<welcome-file-list>
		<welcome-file>index.jsp</welcome-file>
	</welcome-file-list>
</web-app>
```

persistence-beans.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"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
		http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-
4.3.xsd">

	<context:component-scan base-package="com.nt.dao"/>

	<!-- Get ServerManaged JDBC DataSoure object from JNDI registry of
Underlying Server -->
	<bean id="jofb"
```

```
class="org.springframework.jndi.JndiObjectFactoryBean">
                <property name="jndiName" value="java:/comp/env/DsJndi"/>
        </bean>

        <!-- Configure JdbcTemplate injecting DataSource -->
        <bean id="template"
class="org.springframework.jdbc.core.JdbcTemplate">
                <constructor-arg name="dataSource" ref="jofb"/>
        </bean>

</beans>
```

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"
        xsi:schemaLocation="http://www.springframework.org/schema/bean
s https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
                http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-
4.3.xsd">

        <!-- Configure Handler mapping -->
        <bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestM
appingHandlerMapping"/>

        <!-- Configure View Resolver -->
        <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver
">
                <property name="prefix" value="/WEB-INF/pages/"/>
                <property name="suffix" value=".jsp"/>
        </bean>

        <context:component-scan base-package="com.nt.controller"/>

</beans>
```

EmployeeDTO.java

```java
package com.nt.dto;

import java.io.Serializable;

import lombok.Data;

@Data
public class EmployeeDTO implements Serializable {
	private int serialNo;
	private int empNo;
	private String ename;
	private String job;
	private float sal;
	private int deptNo;
	private float grossSalary;
	private float netSalary;
}
```

EmployeeBO.java

```java
package com.nt.bo;

import lombok.Data;

@Data
public class EmployeeBO {
	private int empNo;
	private String ename;
	private String job;
	private float sal;
	private int deptNo;
}
```

IEmployeeDAO.java

```java
package com.nt.dao;

import java.util.List;

import com.nt.bo.EmployeeBO;

public interface IEmployeeDAO {
	public List<EmployeeBO> getAllEmployees();
}
```

EmployeeDAOIml.java

```
package com.nt.dao;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.ResultSetExtractor;
import org.springframework.stereotype.Repository;

import com.nt.bo.EmployeeBO;

@Repository("empDAO")
public class EmployeeDAOImpl implements IEmployeeDAO {

	private static final String GET_ALL_EMPLOYEES="SELECT EMPNO, 
ENAME, JOB, SAL, DEPTNO FROM EMP";

	@Autowired
	private JdbcTemplate jt;

	@Override
	public List<EmployeeBO> getAllEmployees() {
		List<EmployeeBO> listBO = null;
		listBO = jt.query(GET_ALL_EMPLOYEES, new 
EmployeeRowMapper());
		return listBO;
	}

	private class EmployeeRowMapper implements 
ResultSetExtractor<List<EmployeeBO>> {
		@Override
		public List<EmployeeBO> extractData(ResultSet rs) throws 
SQLException, DataAccessException {
			List<EmployeeBO> listBO = new ArrayList();
			//copy RS records to one BO class object
			while(rs.next()) {
				EmployeeBO bo = new EmployeeBO();
				bo.setEmpNo(rs.getInt(1));
```

```
                                        bo.setEname(rs.getString(2));
                                        bo.setJob(rs.getString(3));
                                        bo.setSal(rs.getFloat(4));
                                        bo.setDeptNo(rs.getInt(5));
                                        listBO.add(bo);
                              }
                              return listBO;
                    }
          } //inner class

}
```

IEmployeeMgmtService.java

```
package com.nt.service;

import java.util.List;

import com.nt.dto.EmployeeDTO;

public interface IEmployeeMgmtService {

          public List<EmployeeDTO> fetchAllEmployees();
}
```

EmployeeMgmtServiceImpl.java

```
package com.nt.service;

import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.nt.bo.EmployeeBO;
import com.nt.dao.IEmployeeDAO;
import com.nt.dto.EmployeeDTO;

@Service("empService")
public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService
{
          @Autowired
          private IEmployeeDAO dao;
```

```
	@Override
	public List<EmployeeDTO> fetchAllEmployees() {
		List<EmployeeBO> listBO = null;
		List<EmployeeDTO> listDTO = new ArrayList<>();
		//use DAO
		listBO = dao.getAllEmployees();
		//convert listBO to listDTO
		listBO.forEach(bo->{
			EmployeeDTO dto = new EmployeeDTO();
			BeanUtils.copyProperties(bo, dto);
			dto.setSerialNo(listDTO.size()+1);
			dto.setGrossSalary(dto.getSal()+dto.getSal()*0.3f);

	dto.setNetSalary(dto.getGrossSalary()+dto.getGrossSalary()*0.1f);
			dto.setSal(Math.round(dto.getSal()));
			listDTO.add(dto);
		});
		return listDTO;
	}

}
```

EmployeeController.java

```
package com.nt.controller;

import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

import com.nt.dto.EmployeeDTO;
import com.nt.service.IEmployeeMgmtService;

@Controller
public class EmployeeController {

	@Autowired
	private IEmployeeMgmtService service;

	@GetMapping("/welcome")
	public String showHomePage() {
```

```
                return "home";
        }

        @GetMapping("/list_emps")
        public String showEmployee(Map<String, Object> map) {
                List<EmployeeDTO> listDTO = null;
                //use service
                listDTO = service.fetchAllEmployees();
                //keep result in model attribute
                map.put("empsInfo", listDTO);
                return "show_report";
        }
}
```

home.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>

<html>
<head>
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min
.css">
</head>

<body>
        <div class="container p-3 my-3 bg-primary text-white">
                <h1 style="text-align: center">Employee Details</h1>
        </div>

        <div class="container p-3 my-3 border">
                <h3 style="color: cyan; text-align: left">
                        <a href="list_emps">Get all Employees</a>
                </h3>
        </div>
</body>

</html>
```

show_report.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1" isELIgnored="false"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min
.css">
</head>
<body>
        <div class="container p-3 my-3 bg-primary text-white">
                <h1 style="text-align: center">Employee
InformationReport</h1>
        </div>
        <div class="container">
                <c:choose>
                        <c:when test="${empsInfo ne null && !empty
empsInfo}">
                                <table class="table table-dark table-striped table-
sm">
                                        <tr>
                                                <th>Serial No.</th>
                                                <th>Employee ID</th>
                                                <th>Employee Name</th>
                                                <th>Designation</th>
                                                <th>Salary</th>
                                                <th>Gross Salary</th>
                                                <th>Net Salary</th>
                                                <th>Department No.</th>
                                        </tr>
                                        <c:forEach var="dto" items="${empsInfo}">
                                                <tr>
                                                        <td>${dto.serialNo}</td>
                                                        <td>${dto.empNo}</td>
                                                        <td>${dto.ename}</td>
                                                        <td>${dto.job}</td>
                                                        <td>${dto.sal}</td>
                                                        <td>${dto.grossSalary}</td>
                                                        <td>${dto.netSalary}</td>
```

```
                                    <td>${dto.deptNo}</td>
                               </tr>
                          </c:forEach>
                     </table>
                </c:when>
           </c:choose>
           <h3><a href="welcome">Home</a></h3>
           <h2><a href="JavaScript:doPrint()">Print</h2>
      </div>
      <script lang="JavaScript"">
                function doPrint() {
                     frames.focus();
                     frames.print();
                }
      </script>
</body>
</html>
```

To apply bootstrapping (Bunch of CSS, JS libraries):

Step 1: Write following tag by collecting from www.getBootstrap.com

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
```

Step 2: Know the style classes and apply them using "class" attribute different tags.

Code for taking Printout of the Webpage:

```
     <a href="JavaScript:doPrint()">Print</h2>
     <script lang="JavaScript"">
               function doPrint() {
                    frames.focus();
                    frames.print();
               }
     </script>
```

Note: We can use the JS and CSS also from the Bootstrap. It is a set of predefine library helpful to apply style and validation logic for web page

## Data Binding

### Passing Data from view comps to Controller:

a. Using forms
   - Traditional HTML forms
   - Spring MVC JSP forms (Spring MVC supplied JSP tags)
   - UI Forms (jQuery/ Angular JS/ Angular forms/ React Js forms)
     [This is useful to carry to end-user supplied dynamic data]

b. Using Request params appended to the Query String
   (hyperlink having href URL with query string)
   href="<path>?<param1>=<val1>& <param2>=<va12>"
   [ This useful carry application supplied static/ dynamic data]
   e.g. edit & delete, these hyperlinks will carry application supplied PK value as additional Data.

c. Using JavaScript (JS based Ajax/ jQuery/ Angular/ ....)
   To make any comp/ any event sending request and data to web application.
   e.g.
   - When page is loaded onLoad event should the send the request to get countries to select box
   - When country selected in select box request go with data and should bring states into another select box

## Using Forms

- In spring MVC working with forms will have two phases,

a. Initial Phase (talks about displaying from page through handler method)
   - Generally, its "GET" mode request.
   - We can design form either using traditional HTML tags or Spring MVC tags(best).
   - In the handler we create and keep model (new)/ command (old) class object with empty data/ initial Data.
   - This model/ command class is java bean having property names matching with form component names (text box, radio buttons and other names).
   - If form page form is designed using Spring MVC supplied JSP pages them Model class object initial value can be displayed as the initial values of form comps (like text boxes) with traditional HTML tags this is not possible.

Method in Handler/ Controller class

```
@Controller
public class StudentController {
        @GetMapping("/register")
        public String showForm(Map<String,Object> map){
                Student st=new Student(); //model/Command class obj
                map.put("stfrm", st);
                return "student_form"; //LVN of the form page.
        }
}
```

Command class/ Model class

```
public class Student {
        private int sno;
        private String sname;
        private String sadd
        //getters && setters
        ............
}
```

Traditional HTML tags-based form page X

```
<form action="register" method="POST">
        student number: <input type="text" name="sno"/>
        student name: <input type="text" name="sname"/>
        student address: <input type="text" name="sadd"/>
        <input type="submit" value="register student"/>
</form> X
```

Note: Not recommended to use these tags because we cannot put model object in text boxes of form page as initial value. [supports 1-way binding i.e. only form submission to Model object].

Spring MVC supplied JSP form tags-based form page

```
<%@taglib uri=" ..............................." prefix="frm"%>
<frm:form action="register" method="POST" modelAttribute="stfrm">
        student number: <frm:input type="text" path="sno"/>
        student name: <frm: input type="text" path="sname"/>
        student address: <frm: input type="text" path="sadd"/>
        <frm:input type="submit" vlaue="register student">
</frm:form>
```

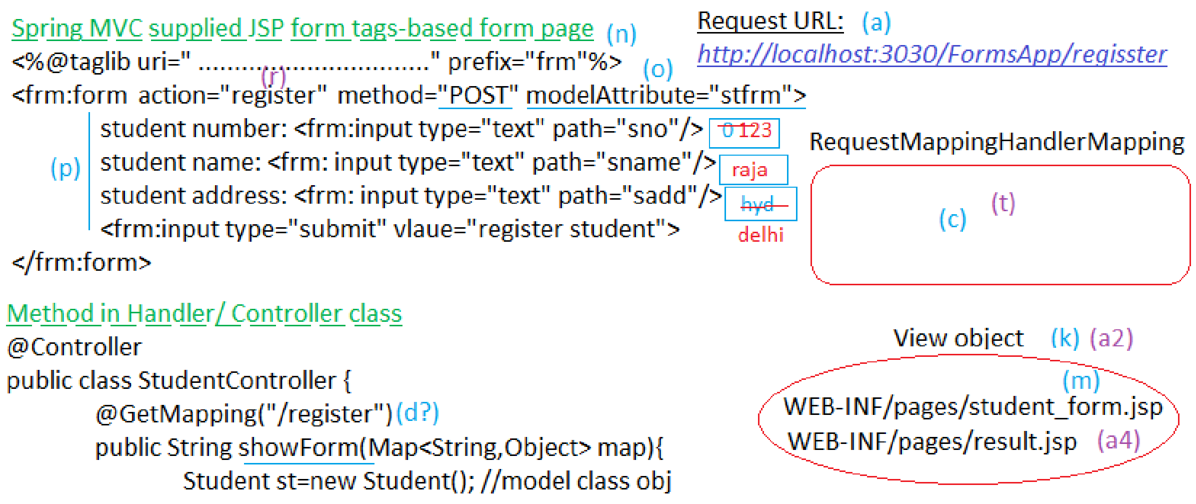
Note: These Spring supplied tags supports two-way binding i.e. while displaying form page model object initial values will be bound to text boxes as initial values and when form submitted them form data will be stored into model object.

b. Post back request (submitting form page and processing request)
   - Here form data will be read and will be stored in Model object by doing necessary conversions.
   - Can create or locate form Model object to perform form binding / request wrapping (nothing but writing form data to Model object).
   - We need handler method controller as shown below.

```
@Controller
public class StudentController{
          @PostMapping("/register")
          public String processFotm(@ModelAttribute("stfrm")Student stud,
                                                        Map<String, Object> map) {
                    ..... // process request directly or using service
                    ..... // add results as model attributes
                    return "result";
          }
}
```

Note: @ModelAttribute given java class as model class and also performs form binding activity (Writing form data to Model class object).

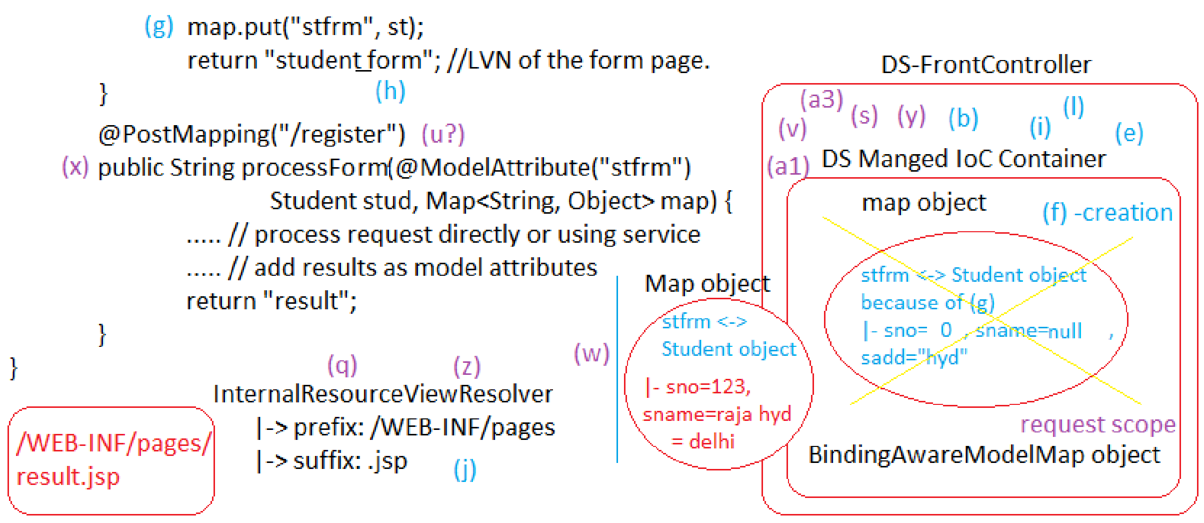
## Flow:



Spring MVC supplied JSP form tags-based form page (n)
(o)
(r)
(p)
~~0~~ 123
raja
~~hyd~~
delhi
Request URL: (a)
http://localhost:3030/FormsApp/regisster
RequestMappingHandlerMapping
(c) (t)
Method in Handler/ Controller class
(d?)
View object (k) (a2)
(m)
WEB-INF/pages/student_form.jsp
WEB-INF/pages/result.jsp (a4)

```
<%@taglib uri=" ..................................." prefix="frm"%>
<frm:form  action="register"  method="POST" modelAttribute="stfrm">
          student number: <frm:input type="text" path="sno"/>
          student name: <frm: input type="text" path="sname"/>
          student address: <frm: input type="text" path="sadd"/>
          <frm:input type="submit" vlaue="register student">
</frm:form>
```

```
@Controller
public class StudentController {
          @GetMapping("/register")
          public String showForm(Map<String,Object> map){
                    Student st=new Student(); //model class obj
```



```
            (g) map.put("stfrm", st);
                return "student_form"; //LVN of the form page.
        }                       (h)
        @PostMapping("/register") (u?)
    (x) public String processForm(@ModelAttribute("stfrm")
                    Student stud, Map<String, Object> map) {
                ..... // process request directly or using service
                ..... // add results as model attributes
                return "result";
        }
}
```

DS-FrontController
(a3) (v) (s) (y) (b) (i) (l) (e)
(a1) DS Manged IoC Container
map object
(f) -creation
stfrm <-> Student object
because of (g)
|- sno= 0 , sname=null ,
sadd="hyd"
request scope
BindingAwareModelMap object
Map object
stfrm <-> Student object
|- sno=123, sname=raja hyd = delhi
(w)
(q) (z)
InternalResourceViewResolver
|-> prefix: /WEB-INF/pages
|-> suffix: .jsp
(j)
/WEB-INF/pages/ result.jsp

<a>go </a> it is not error request goes to previous URL (using which this page launched)

```
<frm:form method="POST" modelAttribute="stfrm">
        ........
        ..........
        ...........
</fm:form>
```

When form is submitted the request goes to previous URL (the URL using which the form page is launched like/ register).

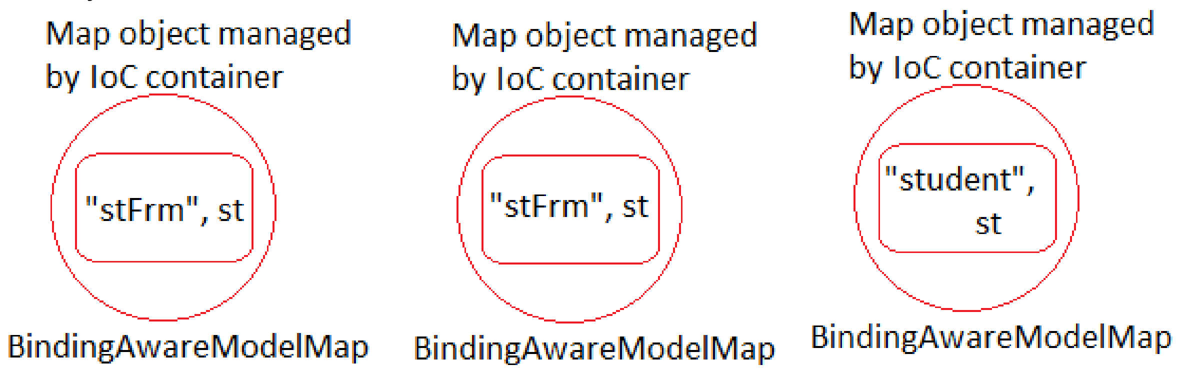
Q. What is the difference Traditional HTML form and Spring MVC JSP form tags?

Ans.

| Traditional HTML forms | Spring MVC JSP forms |
|---|---|
| a. Given by w3c. | a Given by Spring framework. |
| b. Supports one-way binding (form to java object/ request object). | b. Support two-way binding (java object to form and form to java object). |
| c. Initial values of form components cannot be collected from Java object. | c. Can be collected. |
| d. Comes to browser as it is (as HTML tag. | d. internally jsp tags will be converted into html tags with initial collected from java object (model object). |

## Different ways of writing handler method for initial phase request:

a.
```
@GetMapping("/register")
public String showForm(Map<String,Object> map){
            Student st=new Student(); //model class object
            st.setSadd("hyd");
            map.put("stFrm",st);
            return "student_form";
}
```
b.
```
@GetMapping("/register")
public String showForm(@ModelAttribute("stFrm") Student st){
            st.setAdd("hyd");
            return "studen_form";
}
```
c.
```
@GetMapping("/register")
public String showForm(@ModelAttribute Student st){
            st.setAdd("hyd");
            return "student_form";
}
```


Map object managed by IoC container
"stFrm", st
BindingAwareModelMap
Map object managed by IoC container
"stFrm", st
BindingAwareModelMap
Map object managed by IoC container
"student", st
BindingAwareModelMap

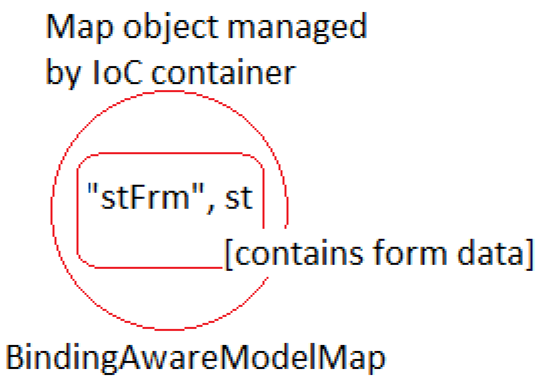
Note: If we don't provide model attribute name in @ModelAttribute(-) then the model/ Command class name will be taken as default model attribute name (but first letter become lowercase latter).

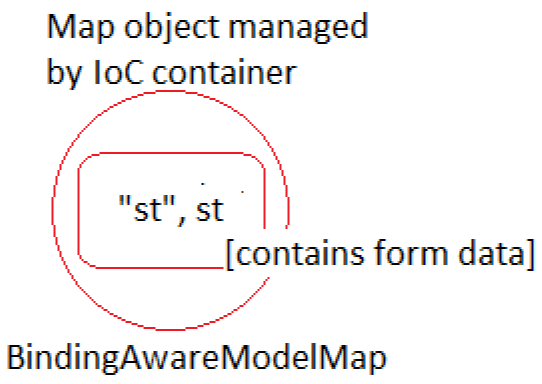
## Different ways of creating Handler Method for POST back request:

a. Use this for "a", "b" versions of initial phase request
```
@PostMapping("/register")
public Stirng processForm(Map<String,Object> map, Student st){
            ..............
            ......//logic to process request and to results
            to map as model attribute
            .....................
            return "result";
}
```

b. Use this for "c" version of initial phase request

```
@PostMapping("/register")
public Stirng processForm(Map<String,Object> map, @ModelAttribute
                                                        Student st){
          ................
          ......//logic to process request and to results to
          map as model attribute
          ..............................
          return "result";
}
```





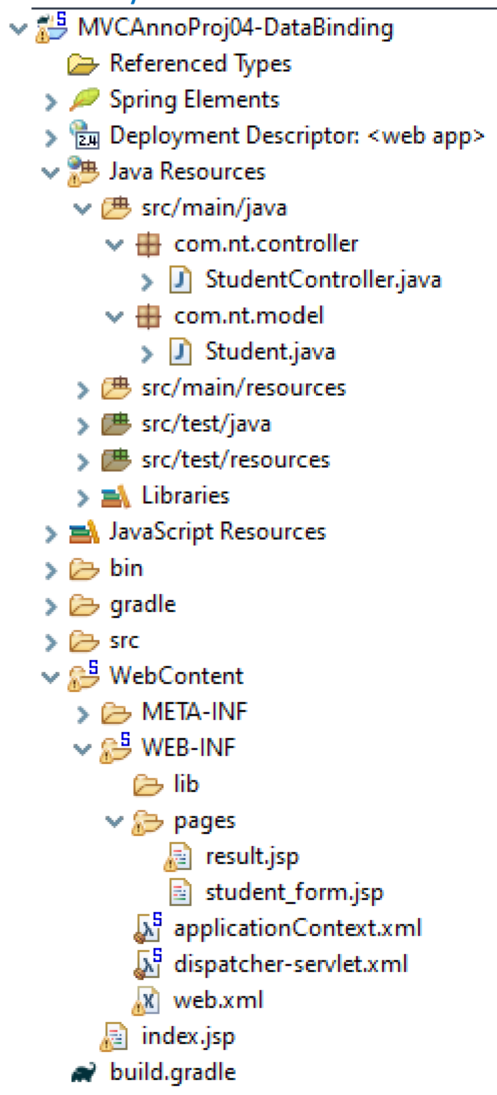
Note: The default Scope of Map object and Model class object/ Form class object/ Command class object is request scope.

Form Data Binding (writing form data to Model class object) performs operations in the following order:

a. Searches handler method that ready to handle Post back request based on request Mode (POST) and request Path (like "/register") [Basically searches for handler method with @PostMapping("/register")].
b. Based @ModelAttribute annotation-based parameter it creates Model class object either default name (method parameter name (st)) or specified name (like stFrm) as model attribute name also keep it in Map object.
c. Reads form data using req.getParameter(-) methods performs necessary conversions using convertor and PropertyEditor and writes from data to Model class object (that is created above).
d. Calls Post back request handler method having the Map object, model class object as the arguments.

Note: In tradition HTML tags based from page the default request mode is "GET" where in Spring MVC supplied JSP tags-based forms the default request mode is "POST".

## Directory Structure of MVCAnnoProj02-DataRendering:


```
MVCAnnoProj04-DataBinding
    Referenced Types
  > Spring Elements
  > Deployment Descriptor: <web app>
  v Java Resources
    v src/main/java
      v com.nt.controller
        > StudentController.java
      v com.nt.model
        > Student.java
    > src/main/resources
    > src/test/java
    > src/test/resources
    > Libraries
  > JavaScript Resources
  > bin
  > gradle
  > src
  v WebContent
    > META-INF
    v WEB-INF
        lib
      v pages
          result.jsp
          student_form.jsp
        applicationContext.xml
        dispatcher-servlet.xml
        web.xml
      index.jsp
    build.gradle
```

- Develop the above directory structure and folder, package, class, XML, JSP, file after convert to web application and add the jar dependencies in build.gradle file then use the following code with in their respective file.
- Copy the build.gradle, web.xml, dispatcher-servlet.xml, index.jsp, applicationContext.xml from previous project.

### index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<jsp:forward page="register"/>
```

### Student.java

```
package com.nt.model;
import lombok.Data;
```

```
@Data
public class Student {
	private int sno;
	private String sname;
	private String sadd;
	private float avg;
}
```

StudentController.java

```
package com.nt.controller;

import java.util.Map;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import com.nt.model.Student;

@Controller
public class StudentController {

	@GetMapping("/register")
	public String showFrom(@ModelAttribute("stFrm") Student st) {
		st.setSadd("hyd");
		return "student_form";
	}

	@PostMapping("/register")
	public String processFrom(Map<String, Object> map,
@ModelAttribute("stFrm") Student st) {
		System.out.println("StudentController.processFrom() : "+st);
		return "result";
	}

}
```

student_form.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://www.springframework.org/tags/form"
```

```
prefix="frm"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min
.css">
</head>
<body>
        <div class="container p-3 my-3 bg-primary text-white">
                <h1>Student Registration Form page</h1>
        </div>
        <frm:form action="register" method="POST"
modelAttribute="stFrm">
        <div class="container">
                <table class="table table-dark table-striped table-sm">
                        <tr><td>Student Number: </td>
                                <td><frm:input path="sno"/></td>
                        </tr>
                        <tr><td>Student Name: </td>
                                <td><frm:input path="sname"/></td>
                        </tr>
                        <tr><td>Student Address: </td>
                                <td><frm:input path="sadd"/></td>
                        </tr>
                        <tr><td>Student Mark: </td>
                                <td><frm:input path="avg"/></td>
                        </tr>
                        <tr>
                                <td colspan="2"><input type="submit"
                        value="Register student"/></td>
                        </tr>
                </table>
        </div>
        </frm:form>

</body>
</html>
```

result.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1" isELIgnored="false"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min
.css">
</head>
<body>
	<div class="container p-3 my-3 bg-primary text-white">
		<h1>Result page</h1>
	</div>

	<div class="container">
		<h3>Model class object data: </h3> ${stFrm}
		<br><br><a href="register">Home</a>
	</div>
</body>
</html>
```

Note: Spring MVC is supplying two JSP Tag libraries,

a. Generic Tag library taglib URI: *http://www.springframework.org/tags*
   recommended prefix: spring
b. Form tag library taglib URI: *http://www.springframework.org/tags/form*
   recommended prefix: form

Q. What is form backing object?
Ans. Model/ command class object is also called as form baking object, because it is having data that is required to populate to form components.

Q. What if model class properties not matching with form component names?
Ans. only matched comp values will be bound to Model class object.

## Using Request Parameters

- Sending data as request params in query string in the form key=value pairs. ?key1=value1&key2=value2=key3=va1ue3

- keys are request param names and values are request param values, both are case-sensitive.

e.g. http://localhost:3030/FirstMVCApp/register?sno=101&sname=raja

To read these values in Servlet programming:

```
int no=Integer.parseInt(req.getParameter("sno")); //gives 101
String name=req.getParameter("sname"); //gives raja
```

Note:

- ✓ GET form data also comes to Server as request params and stored into request object automatically.
- ✓ GET explicitly query string having request params will be stored into request object automatically.
- ✓ POST form data comes to server as request body/ payload but will be stored into request object automatically.

To read these values Spring MVC controller Use @RequestParam:

@RequestParam reads the given request param value and puts handler method parameter

syntax 1: @RequestParam("key") <Data type> <var>
syntax 2: @RequestParam <Data type> <var> (key and var name must match)

e.g.

1. @GetMapping("/links")
   public String processForm(Map<String,Object> map, @RequestParam ("sno") int no, @RequestParam("sname") String name)
2. @GetMapping("/links")
   public String processForm(Map<String,Object> map, @RequestParam int no, @RequestParam String name)

StudentController.java

```
@GetMapping("/link")
public String getLinkData(Map<String, Object> map,
            @RequestParam int sno,
            @RequestParam String sname){
      System.out.println("Request Param: "+sno+" "+sname);
      return "show_params";
}
```

show_params.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1" isELIgnored="false"%>              <h1>Show
Params Page</h1>
         <h3>Request param values: </h3> ${param.sno}, ${param.sname}
<br><br><a href="register">Home</a>
```

URL: *http://localhost:2525/MVCAnnoProj04-DataBinding/link?sno=102&sname=karan*

Note: If we give request params with wrong case then we get 400 error (bad request)
e.g.

```
@GetMapping("/links")
public String processForm(Map<String,Object> map, @RequestParam int Sno,
                                                        @RequestParam String Sname)
```

URL: *http://localhost:2525/MVCAnnoProj04-DataBinding/link?sno=102&sname=karan*

- Additional request params in the query String will be ignored without any error/ exception.

e.g.

```
@GetMapping("/links")
public String processForm(Map<String,Object> map, @RequestParam int Sno,
                                                        @RequestParam String Sname)
```

URL: *http://localhost:2525/MVCAnnoProj04-DataBinding/link?sno=102&sname=karan&sadd=hyd* *(The additional request param)*

- If one request param is having multiple values key1=value1 & key1=va1ue2 & key1=value3 then we read by using simple/ wrapper type method param or array/ List/ Set type method param.

e.g.

```
@GetMapping("/link")
public String getLinkData(Map<String, Object> map, @RequestParam int sno,
         @RequestParam String sname, @RequestParam List<String> sadd){
         System.out.println("Request Param: "+sno+" "+sname+" "+sadd.get(0)+"
"+sadd.get(1));
         return "show_params";
}
```

(OR)

```
@GetMapping("/link")
public String getLinkData(Map<String, Object> map, @RequestParam int sno,
                    @RequestParam String sname, @RequestParam String sadd){
          System.out.println("Request Param: "+sno+" "+sname+" "+sadd);
          return "show_params";
}            //Here sadd holds multiple values as the comma separate list of String
values like vizag, hyd
```

URL: *http://localhost:2525/MVCAnnoProj04-DataBinding/link?sno=102&sname=karan&sadd=hyd&sadd=delhi*

- While creating request prams in the query String of request URL and while reading them through handler methods the order of params do not matter.

e.g.

```
@GetMapping("/link")
public String getLinkData(Map<String, Object> map, @RequestParam int sno,
          @RequestParam List<String[]> sadd, @RequestParam String sname){
          System.out.println("Request Param: "+sno+" "+sname+" "+sadd.get(0)+"
"+sadd.get(1));
          return "show_params";
}
```

(OR)

```
@GetMapping("/link")
public String getLinkData(Map<String, Object> map, @RequestParam int sno,
          @RequestParam String sname, @RequestParam List<String[]> sadd){
          System.out.println("Request Param: "+sno+" "+sname+" "+sadd.get(0)+"
"+sadd.get(1));
          return "show_params";
}
```

URL: *http://localhost:2525/MVCAnnoProj04-DataBinding/link?sno=102&sname=karan&sadd=hyd&sadd=delhi*

(OR)

URL: *http://localhost:2525/MVCAnnoProj04-DataBinding/link?sno=102&sadd=delhi&sadd=hyd&sname=karan*

- If @RequestParam is specified in the handler method and that request param is not passed through query String of request URL then we get 400 (bad request error) because the "required" param of @RequestParam is having "true" as the default value indicating request param must be supplied, otherwise exception will come.

- To disable that compulsion of supplying request param we can take required=false in @RequestParam. This time it will take null as default value when request param is not given in the URL. To assign default value any request param then we take "defaultValue" param of @RequestParam.

e.g.

```
@GetMapping("/link")
public String getLinkData(Map<String, Object> map,@RequestParam int sno,
                    @RequestParam(required = false, defaultValue = "anonymous")
                                        String sname, @RequestParam List<String> sadd){
        System.out.println("Request Param: "+sno+" "+sname+" "+sadd.get(0)+"
"+sadd.get(1));
        return "show_params";
}
```

URL: *http://localhost:2525/MVCAnnoProj04-DataBinding/link?sno=102&sadd=hyd&sadd=delhi*

Note: Since sname is not given and required-false, defaultValue is taken, so "anonymous" will be stored into sname method param

(OR)

URL: *http://localhost:2525/MVCAnnoProj04-DataBinding/link?sno=102&sadd=hyd&sadd=delhi&sname=raja*

Note: The sname hold raja.

- ✓ Q. If we pass mismatched data along with request param value then what happens?

e.g.

```
@GetMapping("/link")
public String getLinkData(Map<String, Object> map, @RequestParam int sno){
        System.out.println("Request Param: "+sno);
        return "show_params";
}
```

URL: *http://localhost:2525/MVCAnnoProj04-DataBinding/link?sno=xyz*

Raises the Exception: 400 bad requests
[org.springframework.web.method.annotation.MethodArgumentTypeMismatchException: Failed to convert value of type 'java.lang.String' to required type 'int'; nested exception is java.lang.NumberFormatException: For input string: "xyz"]

(OR)

```
@GetMapping("/link")
public String getLinkData(Map<String, Object> map, @RequestParam(required
                                                        = false) int sno){
        System.out.println("Request Param: "+sno);
        return "show_params";
}
```
URL: http://localhost:2525/MVCAnnoProj04-DataBinding/link?sno=xyz

Raises the Exception: 400 bad requests [org.springframework.web.method.annotation.MethodArgumentTypeMismatchException: Failed to convert value of type 'java.lang.String' to required type 'int'; nested exception is java.lang.NumberFormatException: For input string: "xyz"]

- ✓ Q. What happens for the following request URL and handler Method?

e.g.
```
@GetMapping("/link")
public String getLinkData(Map<String, Object> map, @RequestParam(required
                                                        = false) int sno){
        System.out.println("Request Param: "+sno);
        return "show_params";
}
```
URL: http://localhost:2525/MVCAnnoProj04-DataBinding/link?sno=

Note: Here default value of sno is " " and it can't be converted into numeric value.

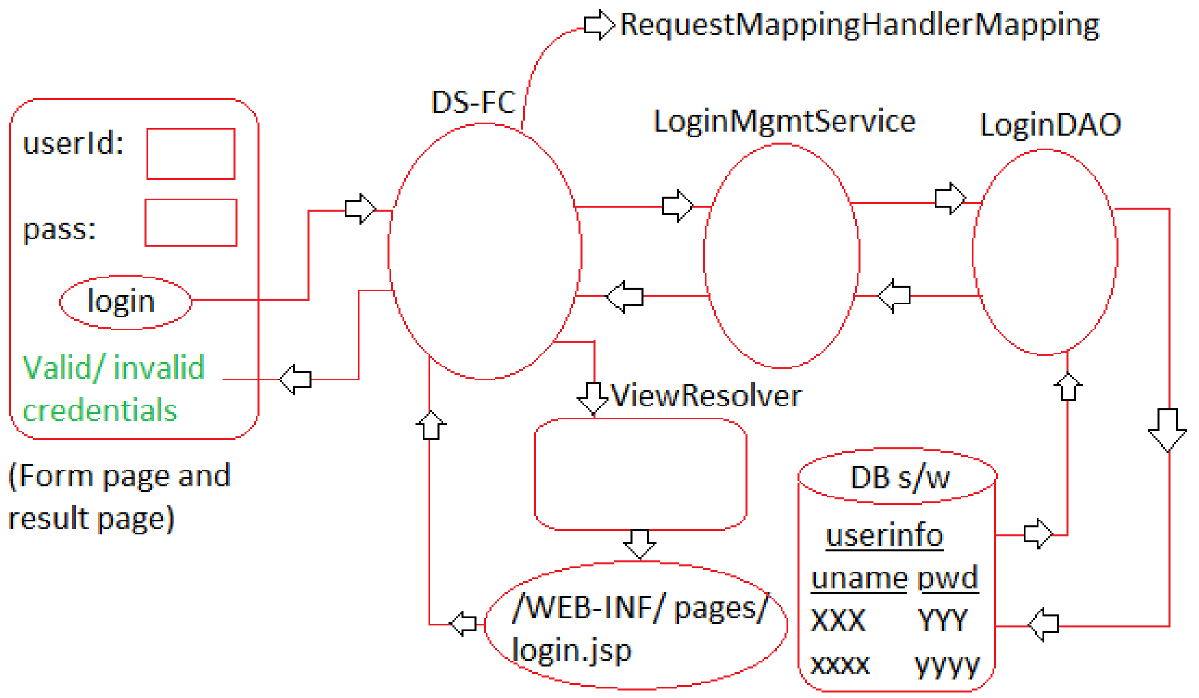
Raises the Exception: 400 bad requests [org.springframework.web.method.annotation.MethodArgumentTypeMismatchException: Failed to convert value of type 'java.lang.String' to required type 'int'; nested exception is java.lang.NumberFormatException: For input string: "xyz"]

To solve the above exception

e.g.
```
@GetMapping("/link")
public String getLinkData(Map<String, Object> map, @RequestParam(required
                                                        = false) Integer sno){
        System.out.println("Request Param: "+sno);
        return "show_params";
}
```
URL: http://localhost:2525/MVCAnnoProj04-DataBinding/link

(OR)
```
@GetMapping("/link")
public String getLinkData(Map<String, Object> map, @RequestParam(required
                                                    = false) String sno){
        System.out.println("Request Param: "+sno);
        return "show_params";
}
```
URL: http://localhost:2525/MVCAnnoProj04-DataBinding/link

## Login application using Annotation driven Spring MVC



RequestMappingHandlerMapping
DS-FC
LoginMgmtService
LoginDAO
userId:
pass:
login
Valid/ invalid credentials
(Form page and result page)
ViewResolver
DB s/w
userinfo
uname pwd
XXX YYY
xxxx yyyy
/WEB-INF/ pages/ login.jsp

Note:

- ✓ The reusable logics that are required in multiple modules o project will be placed in DB s/w as stored procedures/ functions.
- ✓ In spring MVC form launching (initial phase request-GET) and form submission (post back request-POST) will be handled by same controller with two different handler methods having same request path/ URI with two different modes of request (GET, POST).

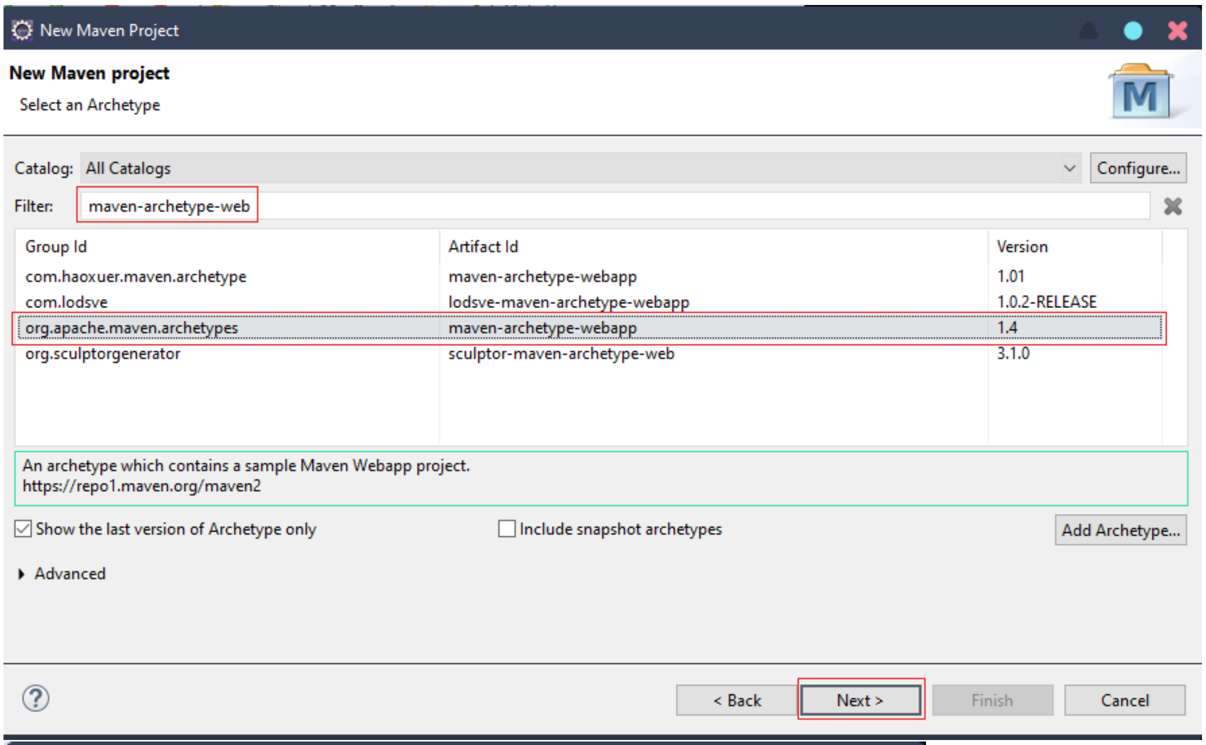
Procedure to develop Spring MVC web application using Eclipse + Maven:

Step 1: Create maven project by selecting "maven-archetype-webapp" as the archetype (Project templates).

File --> Maven Project --> Next --> Next --> select "maven-archetype-webapp" as archetype quick start --> provide details

Step 2: Do following things on the Project

- Open Project properties change java version to 13/14/15, and [v]Dynamic web module 4.0 in Project Facets section.
- Open Order and export tab in Project BUILDPATH and select unselected checkboxes.



New Maven Project

New Maven project

Select an Archetype

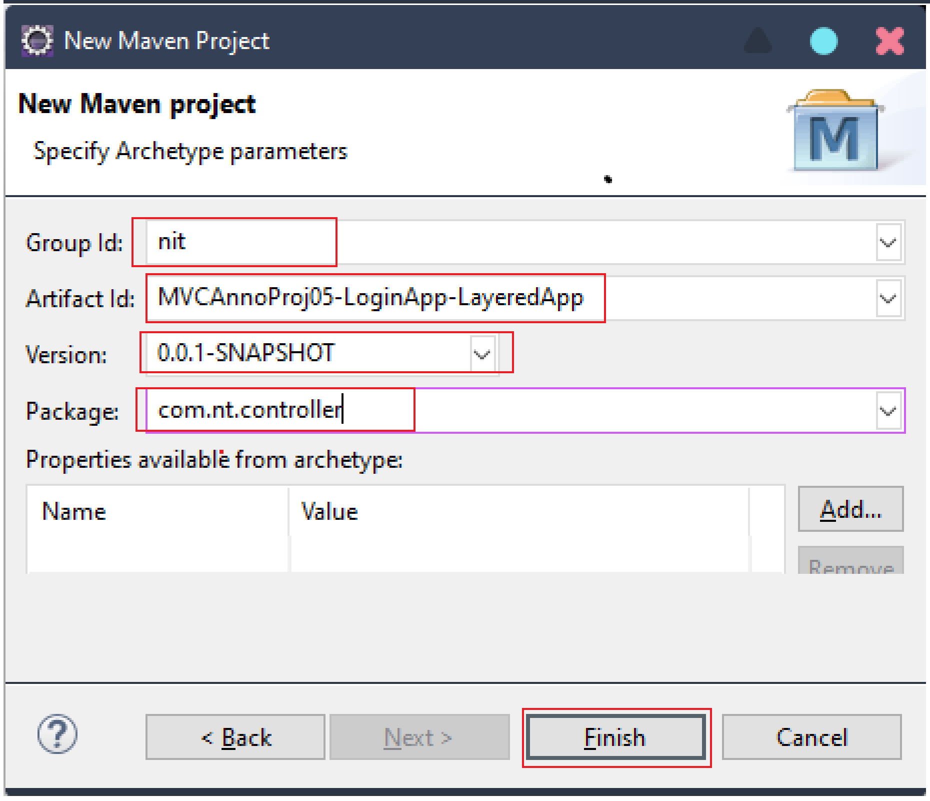
Catalog: All Catalogs

Filter: maven-archetype-web

| Group Id | Artifact Id | Version |
|---|---|---|
| com.haoxuer.maven.archetype | maven-archetype-webapp | 1.01 |
| com.lodsve | lodsve-maven-archetype-webapp | 1.0.2-RELEASE |
| org.apache.maven.archetypes | maven-archetype-webapp | 1.4 |
| org.sculptorgenerator | sculptor-maven-archetype-web | 3.1.0 |

An archetype which contains a sample Maven Webapp project.
https://repo1.maven.org/maven2

☑ Show the last version of Archetype only ☐ Include snapshot archetypes



New Maven Project

New Maven project

Specify Archetype parameters

Group Id: nit

Artifact Id: MVCAnnoProj05-LoginApp-LayeredApp

Version: 0.0.1-SNAPSHOT

Package: com.nt.controller

Properties available from archetype:

| Name | Value |
|---|---|

Step 3: Add the following dependencies (Jar files) in pom.xml file.
jstl1.2, spring web mvc 5.2.9, servlet-api 4.0.1, spring jdbc 5.2.9

Step 4: Create packages for source code in src/main/java folder and place HTML, JSP, Spring configuration file in webcontent and its sub folders accordingly.

Step 5: Run the Application.

Note: SimpleJdbcCall internally uses JdbcTemplate and simplifies PL/SQL procedure call by Generating query based on given Procedure Name, Map of IN param names and values and also calls PL/ SQL procedure and return Map object having OUT param names and values.

```
sjc (SimpleJdbcCall)
        |--> Ds (Dependent object)
        |--> setProcedureName(-): gives PL/SQP procedure name
        |--> Map object having IN Param names and values
        |--> Execute (Map object)
                |-- Map object (Gives out param names and values)
```

```
<jee:jndi-lookup id="ds" jndi-name="java:/comp/env/DsJndi"/>
```

Internally uses "JndiObjectFactoryBean" to get DS object from underlying server Jndi registry based given Jndi name and also makes the received DS object as Spring Bean with the bean id "ds".

```
<bean
id="dsTxMgmr"class="org.springframework.jdbc.datasource.DataSourceTransa
ctionManager">
        <property name="dataSource" ref="ds"/>
</bean>
```

Takes care of beginning Tx, committing Tx and rollback Tx automatically. If everything is smooth in the @Transactional method then it commits Tx. If something gone in the method (if exception is raised) rollbacks Tx.

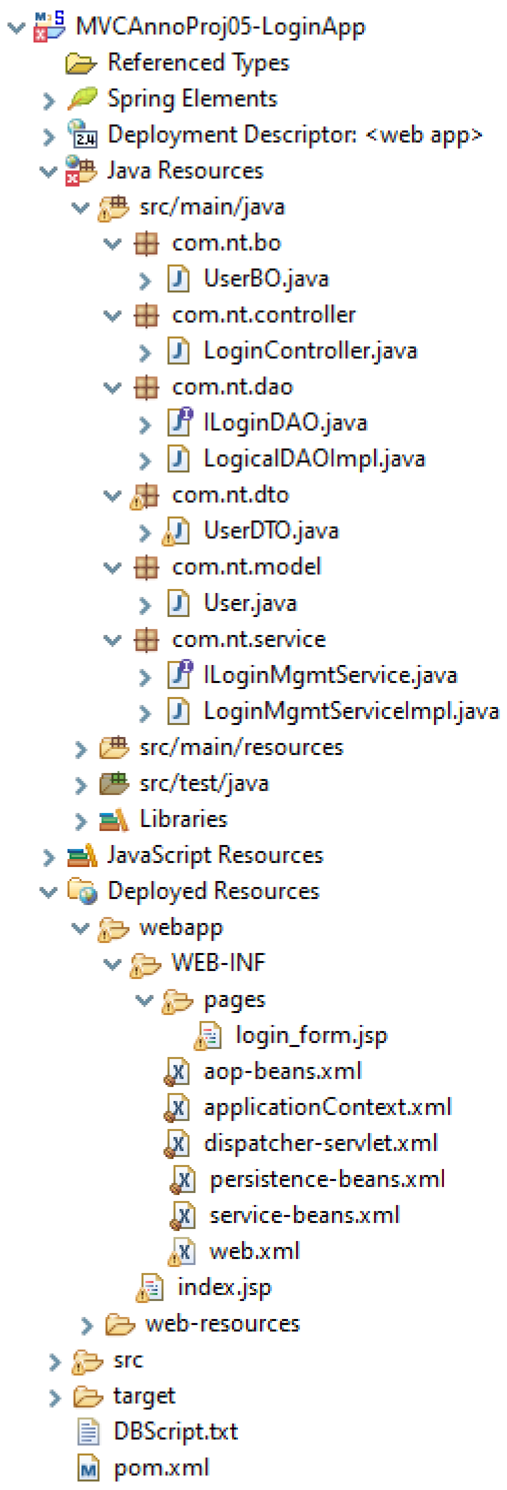
Note:

- ✓ If business methods are dealing bunch of related non-select persistence operations then we definitely need TxMgmt support to apply do everything or nothing principle on the code. (like transfer money withdraw, deposited operations).
- ✓ If business methods are dealing select persistence operations then it recommended to enable TxMgmt support on the business method to make DB s/w only giving "committed data" to Application.

## Directory Structure of MVCAnnoProj05-LoginApp:

- Develop the below directory structure and folder, package, class, XML, JSP, file after convert to web application and add the jar dependencies in pom.xml file then use the following code with in their respective file.

- Copy the web.xml, dispatcher-servlet.xml, applicationContext.xml from previous project.


```
MVCAnnoProj05-LoginApp
  Referenced Types
  Spring Elements
  Deployment Descriptor: <web app>
  Java Resources
    src/main/java
      com.nt.bo
        UserBO.java
      com.nt.controller
        LoginController.java
      com.nt.dao
        ILoginDAO.java
        LogicalDAOImpl.java
      com.nt.dto
        UserDTO.java
      com.nt.model
        User.java
      com.nt.service
        ILoginMgmtService.java
        LoginMgmtServiceImpl.java
    src/main/resources
    src/test/java
    Libraries
  JavaScript Resources
  Deployed Resources
    webapp
      WEB-INF
        pages
          login_form.jsp
        aop-beans.xml
        applicationContext.xml
        dispatcher-servlet.xml
        persistence-beans.xml
        service-beans.xml
        web.xml
      index.jsp
    web-resources
  src
  target
  DBScript.txt
  pom.xml
```

index.jsp

```
<jsp:forward page="login"/>
```

pom.xml

```
	<dependencies>
		<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>jstl</artifactId>
			<version>1.2</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>javax.servlet-api</artifactId>
			<version>4.0.1</version>
			<scope>provided</scope>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.springframework/spring-webmvc --
>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>5.2.9.RELEASE</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-jdbc</artifactId>
			<version>5.2.9.RELEASE</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.projectlombok/lombok -->
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<version>1.18.12</version>
			<scope>provided</scope>
		</dependency>
	</dependencies>
```

DBScript.txt

```
USERINFO (table)
------------------
CREATE TABLE "SYSTEM"."USERSINFO"
  ("USERNAME" VARCHAR2(20 BYTE) NOT NULL ENABLE,
        "PASSWORD" VARCHAR2(20 BYTE),
         CONSTRAINT "USERSINFO_PK" PRIMARY KEY ("USERNAME"));
------------------
Procedure
-----------------------
create or replace PROCEDURE P_AUTHENTICATION
(
  UNAME IN VARCHAR2
, PASS IN VARCHAR2
, RESULT OUT VARCHAR2
) AS
  CNT NUMBER(4);
BEGIN
   SELECT COUNT(*) INTO CNT FROM USERSINFO WHERE
USERNAME=UNAME AND PASSWORD=PASS;
   IF CNT <> 0 THEN
      RESULT:='VALID CREDENTIAL';
   ELSE
      RESULT:='INVALID CREDENTIAL';
   END IF;
END P_AUTHENTICATION;
-----------------------
```

service-beans.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"
        xsi:schemaLocation="http://www.springframework.org/schema/bean
s https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
                 http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-
4.3.xsd">
        <context:component-scan base-package="com.nt.service"/>
</beans>
```

persistence-beans.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:jee="http://www.springframework.org/schema/jee"
	xsi:schemaLocation="http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-4.3.xsd
		http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
		http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-
4.3.xsd">

	<context:component-scan base-package="com.nt.dao"/>

	<jee:jndi-lookup id="ds" jndi-name="java:/comp/env/DsJndi"/>

	<!-- Configure SimpleJdbcCall injecting DataSource -->
	<bean id="sjCall"
class="org.springframework.jdbc.core.simple.SimpleJdbcCall">
		<constructor-arg name="dataSource" ref="ds"/>
	</bean>
</beans>
```

aop-beans.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"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
		http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-
4.3.xsd">
	<bean id="dsTxMgmr"
class="org.springframework.jdbc.datasource.DataSourceTransactionManag
er">		<property name="dataSource" ref="ds"/>
	</bean>
</beans>
```

login_form.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1" isELIgnored="false"%>
<%@taglib uri="http://www.springframework.org/tags/form" prefix="frm"
%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
        <h1 style="color: red; text-align: center;">Login APP</h1>
        <frm:form modelAttribute="user">
                <table bgcolor="cyan" border="1" align="center">
                        <tr>
                                <td>User Name : </td>
                                <td><frm:input path="username"/></td>
                        </tr>
                        <tr>
                                <td>User Password : </td>
                                <td><frm:password path="password"/></td>
                        </tr>
                        <tr>
                                <td colspan="2"><input type="submit"
value="login"/></td>
                        </tr>
                </table>
        </frm:form>

        <br>

        <c:if test="${resultMsg ne null && !empty resultMsg}">
                <h2 style="color:green; text-align: center;">${resultMsg}</h2>
        </c:if>
```

UserBO.java

```
package com.nt.bo;

import lombok.Data;

@Data
public class UserBO {
        private String  username;
        private String password;
}
```

UserDTO.java

```
package com.nt.dto;

import java.io.Serializable;

import lombok.Data;

@Data
public class UserDTO implements Serializable {
	private String  username;
	private String password;
}
```

User.java

```
package com.nt.model;

import lombok.Data;

@Data
public class User {
	private String  username;
	private String password;
}
```

ILoginDAO.java

```
package com.nt.dao;

import com.nt.bo.UserBO;

public interface ILoginDAO {
	public String authentication(UserBO bo);
}
```

LoginDAOImpl.java

```
package com.nt.dao;

import java.util.HashMap;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.simple.SimpleJdbcCall;
import org.springframework.stereotype.Repository;
```

```java
import com.nt.bo.UserBO;

@Repository("loginDAO")
public class LogicalDAOImpl implements ILoginDAO {

	@Autowired
	private SimpleJdbcCall sjc;

	@Override
	public String authentication(UserBO bo) {
		Map<String, Object> inParams = null, outParam = null;
		//set Procedure name
		sjc.setProcedureName("P_AUTHENTICATION");
		//Prepare Map object having IN parma name and values
		inParams = new HashMap<>();
		inParams.put("UNAME", bo.getUsername());
		inParams.put("PASS", bo.getPassword());
		//call pl/sql procedure
		outParam = sjc.execute(inParams);
		return (String) outParam.get("RESULT");
	}

}
```

ILoginMgmtService.java

```java
package com.nt.service;

import com.nt.dto.UserDTO;

public interface ILoginMgmtService {
	public String login(UserDTO dto);
}
```

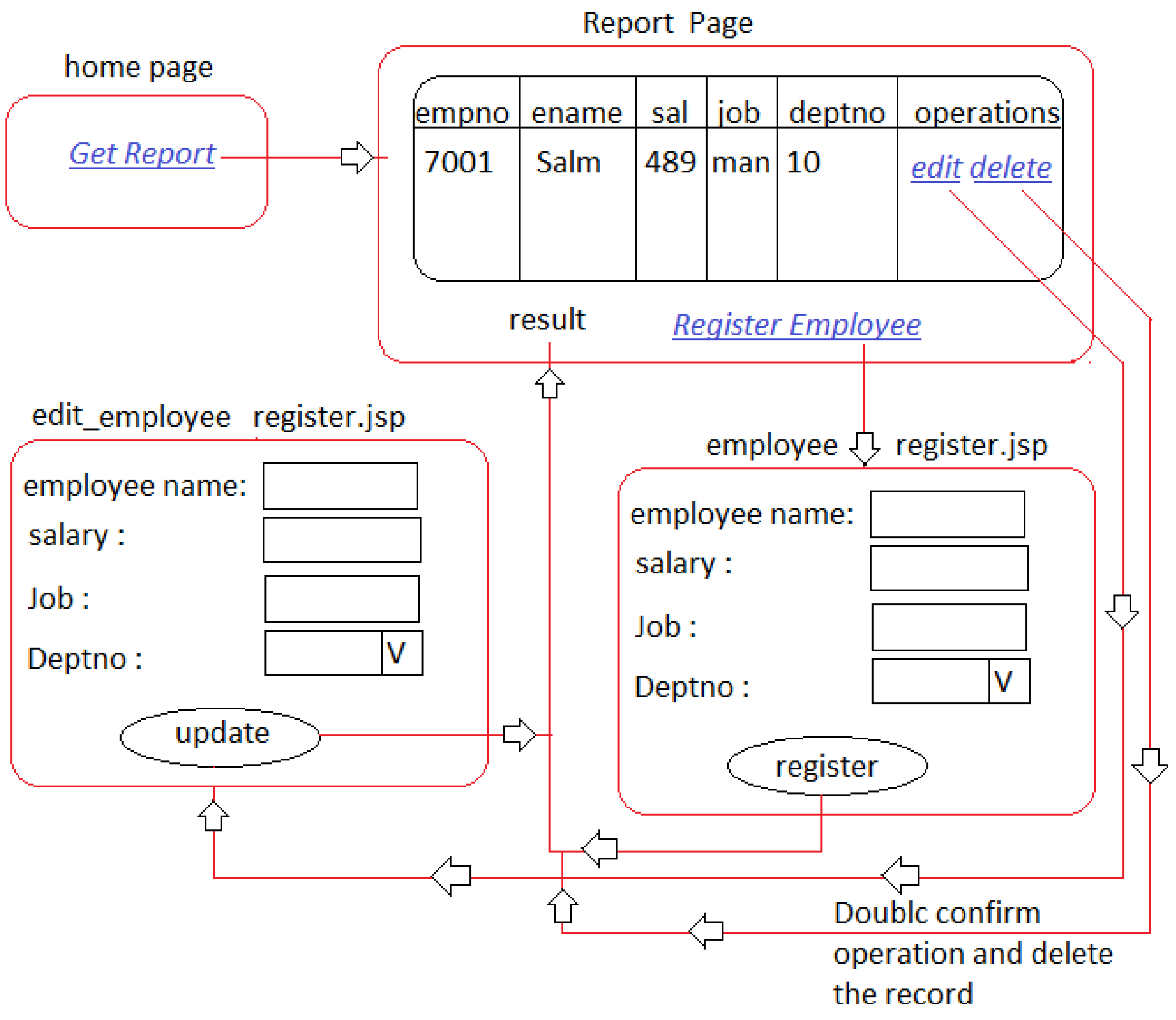
LoginMgmtServiceImpl.java

```java
package com.nt.service;

import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

import com.nt.bo.UserBO;
import com.nt.dao.ILoginDAO;
import com.nt.dto.UserDTO;

@Service("loginService")
public class LoginMgmtServiceImpl implements ILoginMgmtService {

	@Autowired
	private ILoginDAO dao;

	@Override
	@Transactional(propagation = Propagation.REQUIRED )
	public String login(UserDTO dto) {
		UserBO bo = null;
		String result = null;
		//convert dto to bo
		bo = new UserBO();
		BeanUtils.copyProperties(dto, bo);
		//use DAO
		result = dao.authentication(bo);
		return result;
	}

}
```

LoginController.java

```
package com.nt.controller;

import java.util.Map;

import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import com.nt.dto.UserDTO;
import com.nt.model.User;
import com.nt.service.ILoginMgmtService;

@Controller
public class LoginController {
        @Autowired
        private ILoginMgmtService service;


        @GetMapping("/login")
        public String showLoginForm(@ModelAttribute User userDetails) {
                userDetails.setUsername("ghost");
                return "login_form";
        }

        @PostMapping("/login")
        public String performingLogin(Map<String, Object> map,
@ModelAttribute User user) {
                UserDTO dto = null;
                String result = null;
                //convert model to dto
                dto = new UserDTO();
                BeanUtils.copyProperties(user, dto);
                // use service
                result = service.login(dto);
                //add result to map object
                map.put("resultMsg", result);
                //return LVN
                return "login_form";
        }


}
```

## Mini Project



Note: While working radio buttons, checkboxes, select boxes, list boxes we cannot get their bunch of initial values as dynamic from the bound the model class properties because they are designed hold selected values, not all the dynamic initial value. So, we need construct dynamic initial values from as reference data by @ModelAttribute method having List collection as the return type and we need specify that model attribute name as "items" value for the above components .

Controller class

```
        @ModelAttribute("deptsInfo")
        public List<Integer> populateDeptsNos(){
                //use service
                return service.fetchALLDeptNo();
          }
```

Form page
```
<frm:select path="deptNo">
        <frm:options items="${deptsInfo}"/>
</frm:select>
```

Note: @ModelAttribute is multipurpose annotation

a. To make java bean as model class or command class in Initial phase, post back handler methods.

```
@GetMapping("/welcome.htm")
public String showHomePage() {
        return "home";
}
```

b. To provide additional reference data as dynamic initial values for check boxes (group), radio buttons (group), select box and list box components.

```
@ModelAttribute("deptsInfo")
public List<Integer> populateDeptsNos(){
        //use service
        return service.fetchALLDeptNo();
}
```

Note:

- ✓ @ModelAttribute methods executes during initial phase of request to provide dynamic initial values for 4 special form components like checkboxes (group), radio buttons (group), select box and list box components of form launching.
- ✓ @ModelAttribute methods also executes during post back request to provide dynamic initial values with old select values for 4 special form components like checkboxes (group), radio buttons (group), select box and list box components towards form submission having Validation errors.

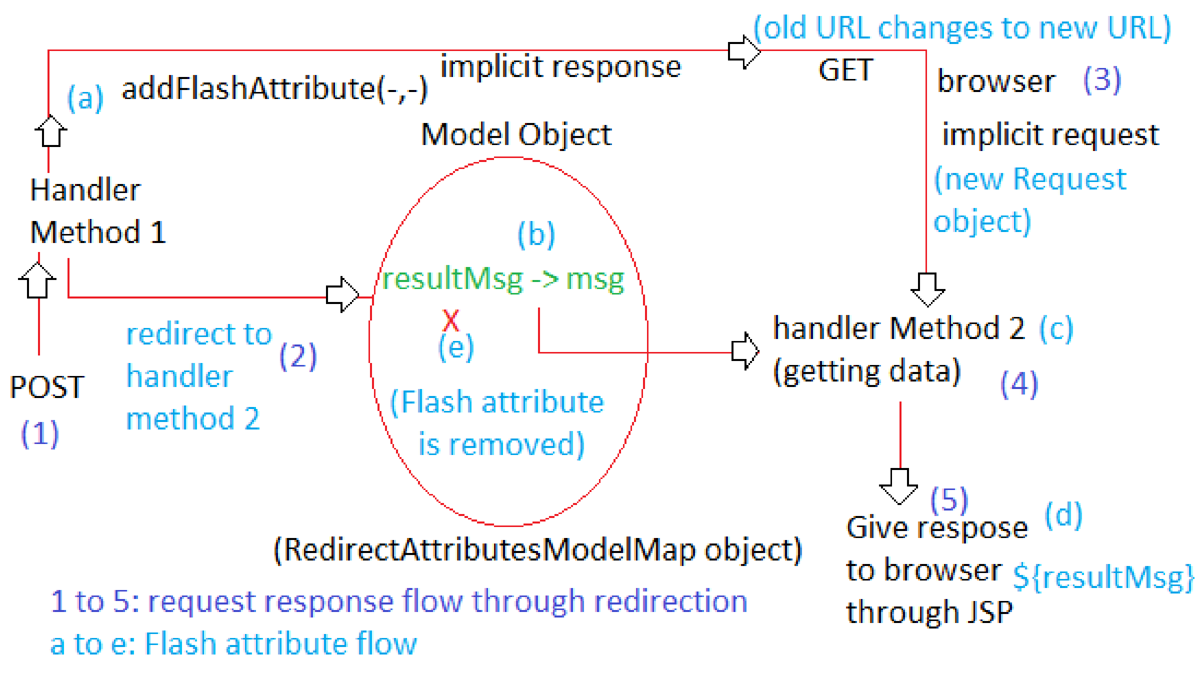
Q. What is doble Posting problem (duplicate form submission) and how to solve that problem using Spring MVC?

Ans. Submitting post mode request for multiple times either by pressing back button and submit or using refresh button will affect data of Application

negatively and this called double posting or duplicate form submission problem.
e.g.

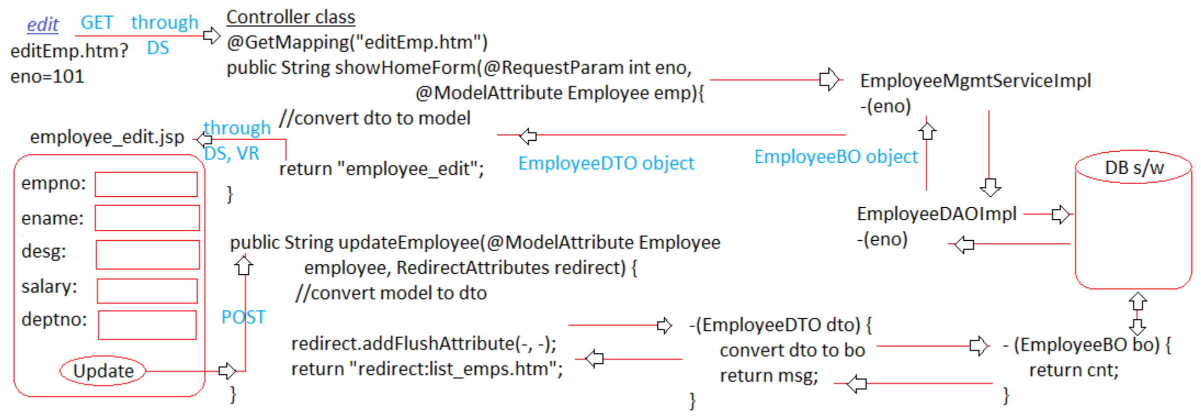
1. On, result page of credit/debit payment pressing refresh button or back with submit button
2. On, result page of student registration submission pressing refresh button.

- Servlet, JSP programming, we solve double posting problem by using session tokens concept.
- In spring MVC programming, we solve double posting problem by using PRG Pattern. (Post Redirect Get pattern).
- PRG Pattern say the request given to post back request handler method (POST) should be redirected GET request mode handler method, In this process the redirection activity changes URL in the browser address bar to GET mode request handler method request path. So, we press refresh button GET request executes which harmless (i.e. POST request of form submit will not execute for refresh button).

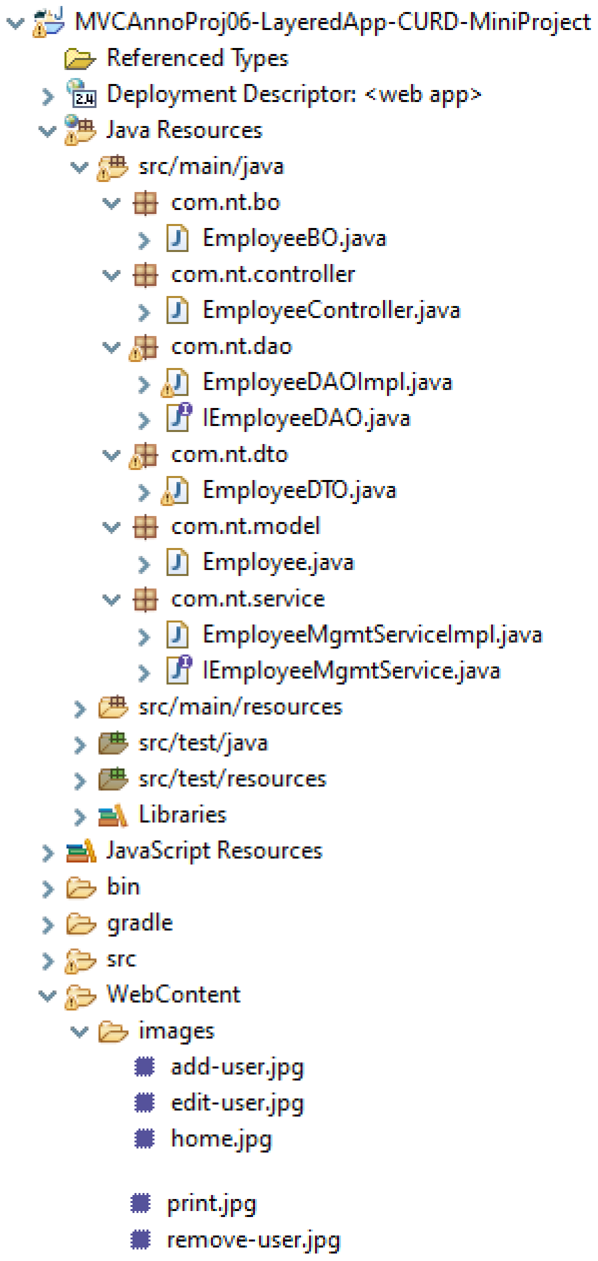

(old URL changes to new URL)
(a) addFlashAttribute(-,-)
implicit response
GET
browser (3)
implicit request
(new Request object)
Model Object
Handler Method 1
(b)
resultMsg -> msg
X
(e)
(Flash attribute is removed)
POST (1)
redirect to handler method 2
(2)
handler Method 2 (c)
(getting data)
(4)
(5)
Give respose to browser through JSP
(d)
${resultMsg}
(RedirectAttributesModelMap object)
1 to 5: request response flow through redirection
a to e: Flash attribute flow

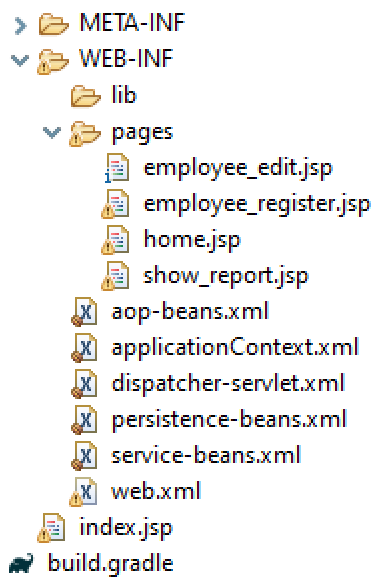
Note: Flash attribute is the model attribute that allocates memory in RedirectAttributes object to make data across the source handler method and target handler method of redirection after that data will be deleted from RedirectAttributes object. (see the above diagram).

## Edit/ Update operation in Mini Project:



## Directory Structure of MVCAnnoProj06-LayeredApp-CURD-MiniProject:


```
MVCAnnoProj06-LayeredApp-CURD-MiniProject
    Referenced Types
  > Deployment Descriptor: <web app>
  Java Resources
    src/main/java
      com.nt.bo
        > EmployeeBO.java
      com.nt.controller
        > EmployeeController.java
      com.nt.dao
        > EmployeeDAOImpl.java
        > IEmployeeDAO.java
      com.nt.dto
        > EmployeeDTO.java
      com.nt.model
        > Employee.java
      com.nt.service
        > EmployeeMgmtServiceImpl.java
        > IEmployeeMgmtService.java
    > src/main/resources
    > src/test/java
    > src/test/resources
    > Libraries
  > JavaScript Resources
  > bin
  > gradle
  > src
  WebContent
    images
        add-user.jpg
        edit-user.jpg
        home.jpg

        print.jpg
        remove-user.jpg
```



```
> META-INF
v WEB-INF
    lib
  v pages
      employee_edit.jsp
      employee_register.jsp
      home.jsp
      show_report.jsp
    aop-beans.xml
    applicationContext.xml
    dispatcher-servlet.xml
    persistence-beans.xml
    service-beans.xml
    web.xml
  index.jsp
build.gradle
```

- Develop the above directory structure and folder, package, class, XML, JSP, file after convert to web application and add the jar dependencies in build.gradle file then use the following code with in their respective file.
- Add following code with their respective files.
- Rest of copy from MVCAnnoProj03-LayeredApp-FetchEmployeeDetails.
- Change the url-pattern to *.htm IN web.xml

index.jsp

```
<jsp:forward page="welcome.htm"/>
```

home.jsp

```
<head>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.mi
n.css">
</head>
<body>
	<div class="container p-3 my-3 bg-primary text-white">
		<h1 style="text-align: center">Employee Details</h1>
	</div>
	<div class="container p-3 my-3 border">
		<h3 style="color: cyan; text-align: left">
			<a href="list_emps.htm">Get all Employees</a>
		</h3>
	</div>
</body>
```

show_report.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1" isELIgnored="false"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstr
ap.min.css">
</head>
<body>
        <div class="container p-3 my-3 bg-primary text-white">
                <h1 style="text-align: center">Employee
InformationReport</h1>
        </div>
        <div class="container">
                <c:choose>
                        <c:when test="${empsInfo ne null && !empty
empsInfo}">
                                <table class="table table-dark table-striped table-
sm">
                                        <tr>
                                                <th>Serial No.</th>
                                                <th>Employee ID</th>
                                                <th>Employee Name</th>
                                                <th>Designation</th>
                                                <th>Salary</th>
                                                <th>Gross Salary</th>
                                                <th>Net Salary</th>
                                                <th>Department No.</th>
                                        </tr>
                                        <c:forEach var="dto" items="${empsInfo}">
                                                <tr>
                                                        <td>${dto.serialNo}</td>
                                                        <td>${dto.empNo}</td>
                                                        <td>${dto.ename}</td>
                                                        <td>${dto.job}</td>
                                                        <td>${dto.sal}</td>
                                                        <td>${dto.grossSalary}</td>
                                                        <td>${dto.netSalary}</td>
                                                        <td>${dto.deptNo}</td>
                                                        <td><a
href="editEmp.htm?eno=${dto.empNo}"><img src="images/edit-user.jpg"
width="50" height="50"></a> <a

        href="deleteEmp.htm?eno=${dto.empNo}" onclick="confirm('Are you
sure to delete '${dto.ename})"><img

        src="images/remove-user.jpg" width="50" height="50"></a></td>
                                                </tr>
                                        </c:forEach>
                                </table>
                        </c:when>
                </c:choose>
                <br>
                <c:if test="${resultMsg ne null && !empty resultMsg}">
                        <marquee direction="right">
                                <h2 class="text-primary font-weight-
bold">${resultMsg}</h2>
                        </marquee>
                </c:if>
                <br>

                <a href="saveEmp.htm"><img src="images/add-user.jpg"
width="100" height="100"/></a> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
        <a href="welcome.htm"><img src="images/home.jpg" width="100"
height="70"/></a> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
        <a href="JavaScript:doPrint()"><img src="images/print.jpg"
                        width="70" height="70">
        </div>

        <script lang="JavaScript"">
                function doPrint() {
                        frames.focus();
                        frames.print();
                }
        </script>

</body>
</html>
```

employee_register.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1" isELIgnored="false"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://www.springframework.org/tags/form"
prefix="frm"%>
<html>
<head>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.mi
n.css">
</head>
<body>
        <div class="container p-3 my-3 bg-primary text-white">
                <h1 style="text-align: center">Employee Registration</h1>
        </div>
        <div class="container">
                <frm:form modelAttribute="empFrm">
                        <table class="table table-dark table-striped table-sm">
                                <tr>
                                        <th>Employee Name</th>
                                        <th><frm:input path="ename"/></th>
                                </tr>
                                <tr>
                                        <th>Employee Designation</th>
                                        <th><frm:input path="job"/></th>
                                </tr>
                                <tr>
                                        <th>Employee Salary</th>
                                        <th><frm:input path="sal"/></th>
                                </tr>
                                <tr>
                                        <th>Department Number</th>
                                        <th><frm:select path="deptNo">
                                                <frm:options
items="${deptsInfo}"/>
                                        </frm:select></th>
                                </tr>
                                <tr>
                                        <td colspan="2"><input
```

```
type="submit" value="Register"/></td>
                                        </tr>
                              </table>
                    </frm:form>
          </div>
          </body>
</html>
```

employee_edit.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
          pageEncoding="ISO-8859-1" isELIgnored="false"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://www.springframework.org/tags/form"
prefix="frm"%>
<html>
<head>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.mi
n.css">
</head>
<body>
          <div class="container p-3 my-3 bg-primary text-white">
                    <h1 style="text-align: center">Employee Edit Information</h1>
          </div>
          <div class="container">
                    <frm:form modelAttribute="employee">
                              <table class="table table-dark table-striped table-sm">
                                        <tr>
                                                  <th>Employee Id</th>
                                                  <th><frm:input path="empNo"
readonly="true"/></th>
                                        </tr>
                                        <tr>
                                                  <th>Employee Name</th>
                                                  <th><frm:input path="ename"/></th>
                                        </tr>
                                        <tr>
                                                  <th>Employee Designation</th>
                                                  <th><frm:input path="job"/></th>
                                        </tr>
```

```
					<tr>
						<th>Employee Salary</th>
						<th><frm:input path="sal"/></th>
					</tr>
					<tr>
						<th>Department Number</th>
						<th><frm:select path="deptNo">
							<frm:options
items="${deptsInfo}"/>
						</frm:select></th>
					</tr>
					<tr>
						<td colspan="2"><input
type="submit" value="Update Employee"/></td>
					</tr>
				</table>
	</frm:form>
	</div></body>
</html>
```

Employee.java

```java
package com.nt.model;

import lombok.Data;

@Data
public class Employee {
	private Integer empNo;
	private String ename;
	private String job;
	private Float sal;
	private Integer deptNo;
}
```

IEmployeeDAO.java

```java
	public int InsertEmployee(EmployeeBO bo);
	public List<Integer> getAllDeptNos();
	public int deleteEmployeeById(int id);
	public EmployeeBO getEmployeeById(int id);
	public int updateEmployee(EmployeeBO bo);
```

EmployeeDAOImpl.java

```
@Repository("empDAO")
public class EmployeeDAOImpl implements IEmployeeDAO {

	private static final String GET_ALL_EMPLOYEES = "SELECT EMPNO,
ENAME, JOB, SAL, DEPTNO FROM EMP";
	private static final String INSERT_EMPLOYEE = "INSERT INTO EMP
(EMPNO, ENAME, JOB, SAL, DEPTNO) VALUES (ENO_SEQ.NEXTVAL, ?, ?, ?,
?)";
	private static final String GET_ALL_DEPTSNO = "SELECT DISTINCT
DEPTNO FROM EMP WHERE DEPTNO IS NOT NULL";
	private static final String DELETE_EMP_BY_NO = "DELETE FROM EMP
WHERE EMPNO=?";
	private static final String GET_EMPLOYEE_BY_NO = "SELECT EMPNO,
ENAME, JOB, SAL, DEPTNO FROM EMP WHERE EMPNO=?";
	private static final String UPDATE_EMPLOYEE_BY_NO = "UPDATE
EMP SET ENAME=?, JOB=?, SAL=?, DEPTNO=? WHERE EMPNO=?";

	@Autowired
	private JdbcTemplate jt;

	@Override
	public List<EmployeeBO> getAllEmployees() {
		List<EmployeeBO> listBO = null;
		listBO = jt.query(GET_ALL_EMPLOYEES, new
ListEmployeeRowMapper());
		return listBO;
	}

	private class ListEmployeeRowMapper implements
ResultSetExtractor<List<EmployeeBO>> {
		@Override
		public List<EmployeeBO> extractData(ResultSet rs) throws
SQLException, DataAccessException {
			List<EmployeeBO> listBO = new ArrayList();
			// copy RS records to one BO class object
			while (rs.next()) {
				EmployeeBO bo = new EmployeeBO();
				bo.setEmpNo(rs.getInt(1));
				bo.setEname(rs.getString(2));
```

```
                                bo.setJob(rs.getString(3));
                                bo.setSal(rs.getFloat(4));
                                bo.setDeptNo(rs.getInt(5));
                                listBO.add(bo);
                        }
                        return listBO;
                }
        } // inner class

        @Override
        public int InsertEmployee(EmployeeBO bo) {
                int count = 0;
                count = jt.update(INSERT_EMPLOYEE, bo.getEname(),
bo.getJob(), bo.getSal(), bo.getDeptNo());
                return count;
        }

        @Override
        public List<Integer> getAllDeptNos() {
                List<Integer> deptNosList = new ArrayList<>();
                ;
                List<Map<String, Object>> listMap = null;
                // execute query
                listMap = jt.queryForList(GET_ALL_DEPTSNO);
                // store in list
                listMap.forEach(e -> {
                        deptNosList.add((Integer) ((BigDecimal)
e.get("deptNo")).intValue());
                });
                return deptNosList;
        }

        @Override
        public int deleteEmployeeById(int id) {
                int count = 0;
                count = jt.update(DELETE_EMP_BY_NO, id);
                return count;
        }

        // Get EmployeeBO approach 1
```

```
	/*@Override
	public EmployeeBO getEmployeeById(int id) {
		EmployeeBO bo = null;
		//use Jt
		bo = jt.queryForObject(GET_EMPLOYEE_BY_NO, new
EmployeeRowMapper(), id);
		return bo;
	}

	private class EmployeeRowMapper implements
RowMapper<EmployeeBO> {
		@Override
		public EmployeeBO mapRow(ResultSet rs, int rowNum) throws
SQLException {
			EmployeeBO bo = null;
			bo = new EmployeeBO();
			bo.setEmpNo(rs.getInt(1));
			bo.setEname(rs.getString(2));
			bo.setJob(rs.getString(3));
			bo.setSal(rs.getFloat(4));
			bo.setDeptNo(rs.getInt(5));
			return bo;
		}
	}*/

	// Get EmployeeBO approach 2
	/*@Override
	public EmployeeBO getEmployeeById(int id) {
		EmployeeBO empbo = null;
		// use Jt
		empbo = jt.queryForObject(GET_EMPLOYEE_BY_NO, (rs,
rowNum) -> {
			EmployeeBO bo = null;
			bo = new EmployeeBO();
			bo.setEmpNo(rs.getInt(1));
			bo.setEname(rs.getString(2));
			bo.setJob(rs.getString(3));
			bo.setSal(rs.getFloat(4));
			bo.setDeptNo(rs.getInt(5));
			return bo;
```

```
                }, id);
                return empbo;
        }*/

        // Get EmployeeBO approach 3
        @Override
        public EmployeeBO getEmployeeById(int id) {
                EmployeeBO empbo = null;
                // use Jt
                empbo = jt.queryForObject(GET_EMPLOYEE_BY_NO, new
BeanPropertyRowMapper<EmployeeBO>(EmployeeBO.class), id);
                return empbo;
        }

        @Override
        public int updateEmployee(EmployeeBO bo) {
                int count = 0;
                count = jt.update(UPDATE_EMPLOYEE_BY_NO,
                                        bo.getEname(), bo.getJob(),
                                        bo.getSal(), bo.getDeptNo(),
                                        bo.getEmpNo());
                return count;
        }

}
```

IEmployeeMgmtService.java

```
package com.nt.service;

import java.util.List;

import com.nt.dto.EmployeeDTO;

public interface IEmployeeMgmtService {
        public List<EmployeeDTO> fetchAllEmployees();
        public String registerEmployee(EmployeeDTO dto);
        public List<Integer> fetchALLDeptNo();
        public String removeEmployeeById(int id);
        public EmployeeDTO fetchEmployeeById(int id);
        public String ModifyEmployeeById(EmployeeDTO dto);
}
```

EmployeeMgmtServiceImpl.java

```
@Service("empService")
public class EmployeeMgmtServiceImpl implements
IEmployeeMgmtService {

        @Autowired
        private IEmployeeDAO dao;

        @Override
        public List<EmployeeDTO> fetchAllEmployees() {
                List<EmployeeBO> listBO = null;
                List<EmployeeDTO> listDTO = new ArrayList<>();
                //use DAO
                listBO = dao.getAllEmployees();
                //convert listBO to listDTO
                listBO.forEach(bo->{
                        EmployeeDTO dto = new EmployeeDTO();
                        BeanUtils.copyProperties(bo, dto);
                        dto.setSerialNo(listDTO.size()+1);
                        dto.setGrossSalary(dto.getSal()+dto.getSal()*0.3f);

        dto.setNetSalary(dto.getGrossSalary()+dto.getGrossSalary()*0.1f);
                        dto.setSal(Math.round(dto.getSal()));
                        listDTO.add(dto);
                });

                return listDTO;
        }

        @Override
        public String registerEmployee(EmployeeDTO dto) {
                EmployeeBO bo = null;
                int count = 0;
                //convert DTO to bo
                bo = new EmployeeBO();
                BeanUtils.copyProperties(dto, bo);
                //use dao
                count = dao.InsertEmployee(bo);
                return count!=0?"Employee Registered":"Employee not
Registered";
        }
```

```
	@Override
	public List<Integer> fetchALLDeptNo() {
		// use dao
		return dao.getAllDeptNos();
	}

	@Override
	public String removeEmployeeById(int id) {
		int count = 0;
		//use DAO
		count = dao.deleteEmployeeById(id);
		return count==0?id+" Employee not found for deletion":id+" 
Employee found and deleted";
	}

	@Override
	public EmployeeDTO fetchEmployeeById(int id) {
		EmployeeDTO dto = null;
		EmployeeBO bo = null;
		//use DAO
		bo = dao.getEmployeeById(id);
		//convert bo to dto
		dto =  new EmployeeDTO();
		BeanUtils.copyProperties(bo, dto);
		return dto;
	}

	@Override
	public String ModifyEmployeeById(EmployeeDTO dto) {
		EmployeeBO bo = null;
		int count = 0;
		//convert DTO to BO
		bo = new EmployeeBO();
		BeanUtils.copyProperties(dto, bo);
		//use Service
		count = dao.updateEmployee(bo);
		return count==0?dto.getEmpNo()+" employee details are not 
found to update":dto.getEmpNo()+" employee details are found to update";
	}

}
```

EmployeeController.java

```
@Controller
public class EmployeeController {

	@Autowired
	private IEmployeeMgmtService service;

	@GetMapping("/welcome.htm")
	public String showHomePage() {
		return "home";
	}

	@GetMapping("/list_emps.htm")
	public String showEmployee(Map<String, Object> map) {
		List<EmployeeDTO> listDTO = null;
		//use service
		listDTO = service.fetchAllEmployees();
		//keep result in model attribute
		map.put("empsInfo", listDTO);
		return "show_report";
	}
	@GetMapping("/saveEmp.htm")
	public String
showEmpRegistrationPage(@ModelAttribute("empFrm") Employee emp) {
		return "employee_register";
	}
	@PostMapping("/saveEmp.htm")
	public String saveEmployee(RedirectAttributes redirect,
@ModelAttribute("empFrm") Employee emp) {
		EmployeeDTO dto = null;
		String result = null;
		//convert model to dto
		dto = new EmployeeDTO();
		BeanUtils.copyProperties(emp, dto);
		//use service
		result = service.registerEmployee(dto);
		//keep in result in flash attribute special map object
		redirect.addFlashAttribute("resultMsg", result);
		//return LVN
		return "redirect:list_emps.htm";
	}
	@GetMapping("/deleteEmp.htm")
	public String removeEmployee(RedirectAttributes redirect,
@RequestParam int eno) {
		String result = null;
		//use service
		result = service.removeEmployeeById(eno);
		//add result to flash attribute
		redirect.addFlashAttribute("resultMsg", result);
		return "redirect:list_emps.htm";
	}
	@GetMapping("/editEmp.htm")
	public String showEditForm(@ModelAttribute Employee emp,
@RequestParam int eno) {
		EmployeeDTO dto = null;
		//use Service
		dto = service.fetchEmployeeById(eno);
		//convert DTO to Model
		BeanUtils.copyProperties(dto, emp);
		return "employee_edit";
	}
	@PostMapping("/editEmp.htm")
	public String updateEmployee(@ModelAttribute Employee
employee, RedirectAttributes redirect) {
		EmployeeDTO dto = null;
		String result = null;
		//convert model to dto
		dto = new EmployeeDTO();
		BeanUtils.copyProperties(employee, dto);
		//use Service
		result = service.ModifyEmployeeById(dto);
		//add to flash attributr
		redirect.addFlashAttribute("resultMsg", result);
		return "redirect:list_emps.htm";
	}
	@ModelAttribute("deptsInfo")
	public List<Integer> populateDeptsNos(){
		//use service
		return service.fetchALLDeptNo();
	}
}
```

## Enable the Transactional Management in the Project:

persistence-beans.xml

```
	<context:component-scan base-package="com.nt.dao"/>
	
	<!-- Get ServerManaged JDBC DataSoure object from JNDI registry of 
Underlying Server -->
	<jee:jndi-lookup id="ds" jndi-name="java:/comp/env/DsJndi"/>
	
	<!-- Configure JdbcTemplate injecting DataSource -->
	<bean id="template" 
class="org.springframework.jdbc.core.JdbcTemplate">
		<constructor-arg name="dataSource" ref="ds"/>
	</bean>
```

persistence-beans.xml

```
	<!-- Configure TxManager -->
	<bean id="dsTxMgmr" 
class="org.springframework.jdbc.datasource.DataSourceTransactionManag
er">
		<constructor-arg ref="ds"/>
	</bean>
	<tx:annotation-driven transaction-manager="dsTxMgmr"/>
```

EmployeeMgmtServiceImpl.java

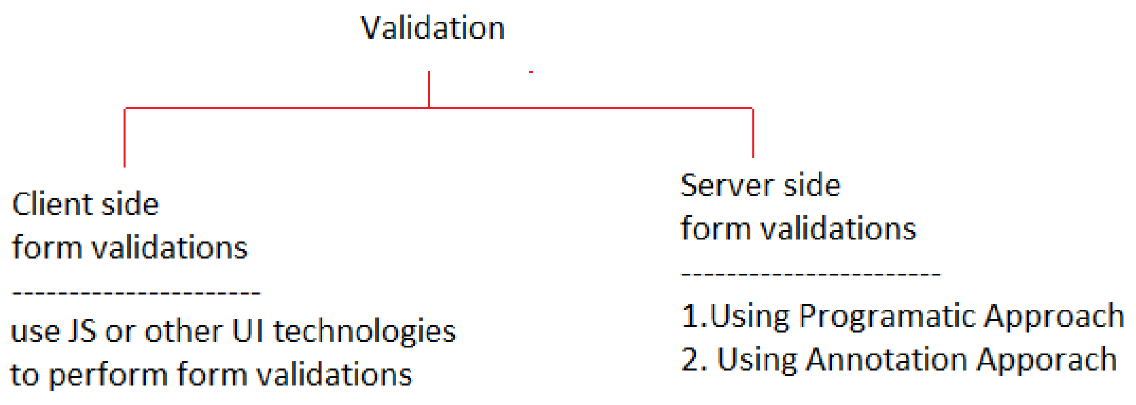
```
	@Override
	@Transactional(propagation = Propagation.REQUIRED, readOnly = 
true)
	public List<EmployeeDTO> fetchAllEmployees() {
	
		…………..
		
	}
```

Note: Write the following line on each method of EmployeeMgmtServiceImpl.java (service class) for enabling the transaction management and write readOnly = true for select operation method and readOnly = false for non-select operation method

@Transactional(propagation = Propagation.***REQUIRED***, readOnly = **true/ false**)

## Form Validations in Spring MVC Application

- Verifying the pattern and format of the form data is called form validation There are two types of form validations.


Validation
Client side
form validations
----------------------
use JS or other UI technologies
to perform form validations
Server side
form validations
-----------------------
1.Using Programatic Approach
2. Using Annotation Apporach

1. Using Programmatic Approach:
   - We develop separate validator class having form validation logic.
   - Allows to enable server-side form validations only when client-side form validations are not done.
2. Using Annotation Approach:
   - We use validation API, hibernate validator API annotations here.
   - Allowing to enable server-side form validations only when client-side form validations are not done is not possible.

Note: The best practice of form validation is written both client side and server-side form validations but enable server-side form validations only when client-side form validations are not one (i.e. if java script of browser is disabled).

Steps to develop server-side form validation logic (Programmatic Approach):
Step 1: Create properties file (com.nt.commons.validation.properties) having form validation error messages and configuration file in dispatcher-servlet.xml.
validation.properties

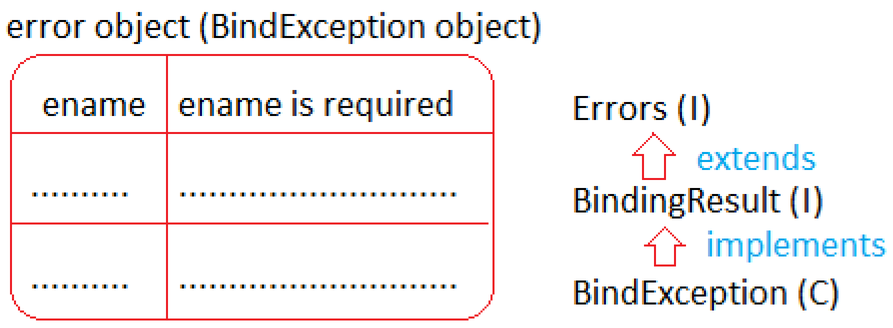
```
#Validation error messages for Programetic approach
emp.name.required = Employee Name is required
emp.name.length = Employee Name must have minimum 5 and maximum 10 characters
emp.job.required = Employee Designation is required
emp.job.length = Employee Designation must have minimum 5 and maximum 9 characters
emp.sal.range = Employee Salary range must be between 10000 to 100000
emp.sal.required = Employee Salary is required
```

```
#Type mismatch
typeMismatch.sal = * Salary must be numeric type
```

dispatcher-servlet.xml

```
	<!-- configure property files -->
	<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSourc
e">
		<property name="basename"
value="com/nt/commons/validation"/>
	</bean>
```

Step 2: Develop validator class (com.nt.validator.EmployeeValidator.java) by implementing validator (I) and write logic in validate (-, -) method



EmployeeValidator.java

```
package com.nt.validator;

import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

import com.nt.model.Employee;

@Component("empValidator")
public class EmployeeValidator implements Validator {

	@Override
	public boolean supports(Class<?> clazz) {
		//useful to check correct command/ model class is received
		return clazz.isAssignableFrom(Employee.class);
	}

	@Override
	public void validate(Object target, Errors errors) {
		Employee emp = null;
		//Type casting on command class object
		emp = (Employee) target;
		//write form validation logic
		if (emp.getEname()==null||emp.getEname().isEmpty()) {
			errors.rejectValue("ename", "emp.name.required");
		} else if 
(emp.getEname().length()<5||emp.getEname().length()>10) {
			errors.rejectValue("ename", "emp.name.length");
		}

		if (emp.getJob()==null||emp.getJob().isEmpty()) {
			errors.rejectValue("job", "emp.job.required");
		} else if (emp.getJob().length()<5||emp.getJob().length()>9) {
			errors.rejectValue("job", "emp.job.length");
		}

		if (emp.getSal()==null) {
			errors.rejectValue("sal", "emp.sal.required");
		} else if (emp.getSal()<10000||emp.getSal()>100000) {
			errors.rejectValue("sal", "emp.sal.range");
		}
	}
}
```

Step 3: Configure Validator class in Spring bean file using <context: component-scan> and also inject to controller class using @Autowired.

dispatcher-servlet.xml

```
	<context:component-scan base-package="com.nt.controller, 
com.nt.validator"/>
```

EmployeeController.java

```
	@Autowired
	private EmployeeValidator validator;
```

Step 4: Write following code in post back handler methods of both employee registration and editing.

EmployeeController.java

```
	@PostMapping("/saveEmp.htm")
	public String saveEmployee(RedirectAttributes redirect,
@ModelAttribute("empFrm") Employee emp,
				BindingResult errors) {
		EmployeeDTO dto = null;
		String result = null;

		// Perform Form validation
		if (validator.supports(emp.getClass()))
			validator.validate(emp, errors);
		// if form validation errors are there launch form page
		if (errors.hasErrors())
			return "employee_register";

		// convert model to dto
		dto = new EmployeeDTO();
		BeanUtils.copyProperties(emp, dto);
		// use service
		result = service.registerEmployee(dto);
		// keep in result in flash attribute special map object
		redirect.addFlashAttribute("resultMsg", result);
		// return LVN
		return "redirect:list_emps.htm";
	}

	@PostMapping("/editEmp.htm")
	public String updateEmployee(RedirectAttributes redirect,
@ModelAttribute Employee employee, BindingResult errors) {
		EmployeeDTO dto = null;
		String result = null;

		// Perform Form validation
		if (validator.supports(employee.getClass()))
			validator.validate(employee, errors);
		// if form validation errors are there launch form page
		if (errors.hasErrors())
			return "employee_edit";

		// convert model to dto
		dto = new EmployeeDTO();
		BeanUtils.copyProperties(employee, dto);
```

```
                // use Service
                result = service.ModifyEmployeeById(dto);
                // add to flash attributr
                redirect.addFlashAttribute("resultMsg", result);
                return "redirect:list_emps.htm";
        }
```

Step 5: Place <frm:errors> in form pages to display form validation error messages

employee_register.jsp and employee_edit.jsp

```
<tr>
        <th>Employee Name</th>
        <th><frm:input path="ename"/><frm:errors cssClass="text-danger"
path="ename"/></th>
</tr>
<tr>
        <th>Employee Designation</th>
        <th><frm:input path="job"/><frm:errors cssClass="text-danger"
path="job"/></th>
</tr>
<tr>
        <th>Employee Salary</th>
                <th><frm:input path="sal"/><frm:errors cssClass="text-danger"
path="sal"/></th>
</tr>
```

Note:

- ✓ <frm:errors cssClass="*text-danger*" path="*xxx*"/> tag we are write against text boxes in form page.
- ✓ Here *xxx* is logical name or property name given in validator class.
- ✓ DS creates empty "errors" object (BindException object) gives to validator class and gets errors object with validator error messages and sends to UI (JS) by keeping in request scope, the <frm:errors> tag internally reads the error messages from "errors" object.

Spring MVC Application can have 3 types of errors:

1. Form validation errors
2. Type mismatch errors
3. Application logic errors

Type mismatch errors:
Will come when form data cannot be stored in model class property
place typeMismatch.<propertY>=<msg> in properties file to get error message.

```
#Type mismatch
  typeMismatch.sal = * Salary must be numeric type
```

Application logic errors (This are business logic errors):
In controller class post back method like saveEmplovee(-,-,-) or editEmplovee(-,-,-) method.

```
//business logic error
if(emp.getJob().equalsIgnoreCase("Netaji")||emp.getJob().equalsIgnore
                                                    Case("Actor"))
        errors.rejectValue("job", "blocked.jobs");
```

```
#Business logic error
blocked.jobs = * Netaji and Actor are not allowed to register
```

Steps to develop server-side form validation logic:
Step 1: Create a Java script file (WebContent/js/validation.js) with the validation logic.
validation.js

```
function  validate(frm){
        alert("js");
        let ename=frm.ename.value;
        let job=frm.job.value;
        let sal=frm.sal.value;
        let flag=true;

        if(ename==""){
                flag=false;
                document.getElementById("enameId").innerHTML="Employee
name is mandatory(js)";
        }
        else if(ename.length<5 || ename.length>10){
                flag=false;
                document.getElementById("enameId").innerHTML="Employee
name must have min of 5 chars and max of 10 chars (js)";
```

```
          }

          if(job==""){
                    flag=false;
                    document.getElementById("jobId").innerHTML="Employee job
is mandatory(js)";
          }
          else if(job.length<5 || job.length>9){
                    flag=false;
                    document.getElementById("jobId").innerHTML="Employee job
must have min of 5 chars and max of 9 chars (js)";
          }
          if(sal==""){
                    flag=false;
                    document.getElementById("salId").innerHTML="Employee sal
is mandatory(js)";
          }
          else if(isNaN(sal)){
                    flag=false;
                    document.getElementById("salId").innerHTML="Employee sal
is must be numeric vlaue(js)";
          }
          else if(sal<10000 || sal>100000){
                    flag=false;
                    document.getElementById("salId").innerHTML="Employee
salary must be in the range between 10000 to 100000(js)";
          }
          return flag;
}
```

Step 2: Link the Java script file with employee_edit.jsp and employee_register.jsp.

employee_edit.jsp

```
<script language="JavaScript" src="js/validation.js">
</script>
<frm:form modelAttribute="employee" onsubmit="return validate(this)">
          <table class="table table-dark table-striped table-sm">
                    <tr>
                              <th>Employee Id</th>
                    <th><frm:input path="empNo" readonly="true" /></th>
            </tr>
            <tr>
                    <th>Employee Name</th>
                    <th><frm:input path="ename" /> <frm:errors
                                        cssClass="text-danger"
path="ename" /><span id="enameId"></span></th>
            </tr>
            <tr>
                    <th>Employee Designation</th>
                    <th><frm:input path="job" /> <frm:errors id="jobId"
                                        cssClass="text-danger"
path="job" /><span id="jobId"></span></th>
            </tr>
            <tr>
                    <th>Employee Salary</th>
                    <th><frm:input path="sal" /> <frm:errors id="salId"
                                        cssClass="text-danger"
path="sal" /><span id="salId"></span></th>
            </tr>
```

employee_register.jsp

```
<script language="JavaScript" src="js/validation.js">
</script>
<frm:form modelAttribute="empFrm" onsubmit="return validate(this)">
        <table class="table table-dark table-striped table-sm">
            <tr>
                    <th>Employee Name</th>
                    <th><frm:input path="ename" /> <frm:errors
                                        cssClass="text-danger"
path="ename" /><span id="enameId"></span></th>
            </tr>
            <tr>
                    <th>Employee Designation</th>
                    <th><frm:input path="job" /> <frm:errors
                                        cssClass="text-danger"
path="job" /><span id="jobId"></span></th>
            </tr>
            <tr>
```

```
                              <th>Employee Salary</th>
                              <th><frm:input path="sal" /> <frm:errors
                                                                cssClass="text-danger"
path="sal" /><span id="salId"></span></th>
                    </tr>
```

Note: While taking handler method signature @ModelAttribute Model class name and BindingResult errors parameters should take back to back, otherwise we get problem while dealing type mismatch errors.

```
        @PostMapping("/editEmp.htm")
        public String updateEmployee(RedirectAttributes redirect,
                @ModelAttribute Employee employee, BindingResult errors) {
                …….
        }
```

Enabling Server-side form validations only when client-side form validations are not done:

Step 1: Add hidden box in form pages (employee_register.jsp, employee_edit.jsp).

```
        <frm:hidden path="vflag"/>
```

Step 2: Take property in model class to hold hidden box value.

```
        @Data
        public class Employee {
                private String vflag="no";
```

Step 3: Write statement in Java script indicating client-side Java script is executed.

```
        /*changing "no" to "yes" in JS indicating
         * client side form validations are done*/
          frm.vflag.value = "yes";
```

Step 4: Write the following in post handler methods saveEmp(-, -,  -) , updateEmployee(-,-,-) methods to perform server side validation based vflag value.

```
        // Enable server-side form validation only when client-side form
        validation is not done.
        if (emp.getVflag().equalsIgnoreCase("no")) {
                // Perform Form validation
                if (validator.supports(emp.getClass()))
                validator.validate(emp, errors);
        }
```

Note:

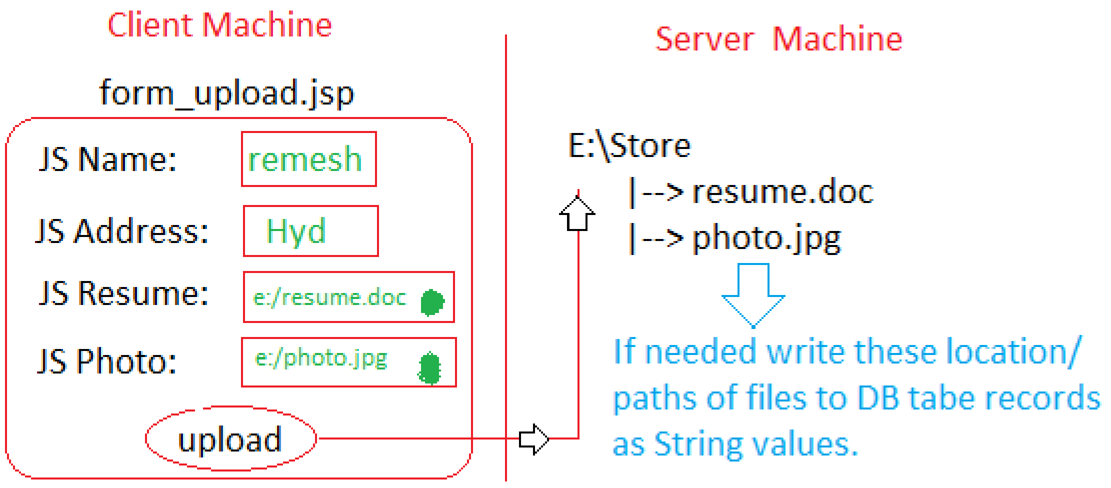
- ✓ To disable JS in chrome: Menu -> settings -> search for java script -> site settings -> Java script -> blocked.
- ✓ To disable JS in FireFox: type about:config in browser -> accept risk -> search for java script -> javascript.enabled -> false.

## File Uploading and Downloading

- Sending files from client machine file system to Server file system is called file uploading and reverse is called file downloading.
- Spring MVC gives built-in support for file uploading and downloading.

Use cases: Matrimony apps, job portal apps, Social networking apps, Profile management apps and etc.

## File Uploading



Note: Saving LOBs (Larges object -files) directly in DB table as BLOB/CLOB column values are bad practice can be done only for standalone Apps. So, best practice is saved them on server machine file system and write their locations to DB table columns as string value.

Steps to perform File Uploading:

Step 1: Take file uploading components in form page.

```
<input type="file" name="resume">
<input type="file" name="photo"/>
```

Step 2: Design form page having POST request mode/ method (to carry unlimited amount of data) and also having "multipart/form-data" as the "enctype".

```
<frm:from action="...." method="POST" modelAttribute="
                              enctype="multipart/form-data">
      <input type="file" name="resume">
      <input type="file" name="photo"/>
</frm:form>
```

Note: enctype="multipart/form-data", To send signal to server that form page sends different types or MIME types of content.

Step 3: Develop model class/ command class having properties to hold uploaded files content (as InputStreams) take the properties of type "MultipartFile" (org.springframework.web.multipart).

```
public class JobSeekerInfo {
      private String jsName;
      private String jsAddrs;
      private MultipartFile resume;
      private MultipartFile photo;
      //setters && getters
      ................
      ..............
}
```

Step 4: Write logic in Post back handler method of controller class by using streams to receive uploaded files content from model class object and write them to server machine file system use IOStream file copy operations.

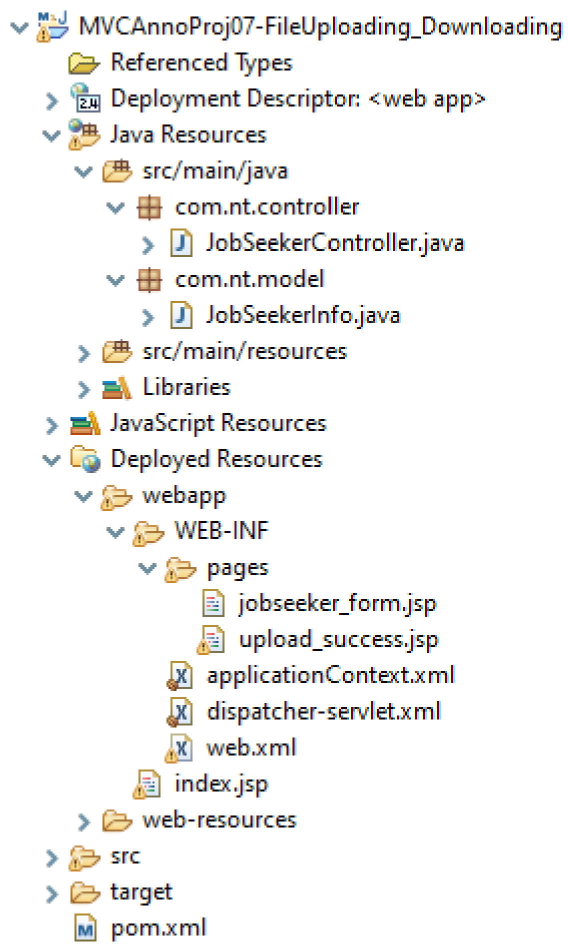
Step 5: configure "CommonsMultipartResolver" as spring bean in dispatcher-servlet.xml having fixed bean id to alert spring MVC environment "multipartResolver" and DS to receive multipart MIME data along with the request.

Note: This allows to specify restrictions related to file uploading.

```
    <!-- Resolver configuration -->
    <bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResol
ver"/>
```

Alerts DS and other components to check weather current request is coming with multipart/form-data enctype or not, if coming it makes DS and other components to receive the uploaded files content to store them in the Model class object MultipartFile type properties.

## Directory Structure of MVCAnnoProj07-FileUploading_Downloading:



```
MVCAnnoProj07-FileUploading_Downloading
    Referenced Types
  > Deployment Descriptor: <web app>
  v Java Resources
    v src/main/java
      v com.nt.controller
        > JobSeekerController.java
      v com.nt.model
        > JobSeekerInfo.java
    > src/main/resources
    > Libraries
  > JavaScript Resources
  v Deployed Resources
    v webapp
      v WEB-INF
        v pages
            jobseeker_form.jsp
            upload_success.jsp
          applicationContext.xml
          dispatcher-servlet.xml
          web.xml
        index.jsp
    > web-resources
  > src
  > target
    pom.xml
```

- Develop the above directory structure and folder, package, class, XML, JSP, file after convert to web application and add the jar dependencies in pom.xml file.
- Copy the common files like applicationContext.xml, dispatcher-servlet.xml and web.xml from the other project
- Add the following code with their respective files.

index.jsp

```
<jsp:forward page="upload"/>
```

web.xml

```
	<context-param>
		<param-name>UploadStore</param-name>
		<param-value>E:/JAVA/Upload_File_Location</param-value>
	</context-param>
```

dispatcher-servlet.xml

```
	<!-- Resolver configuration -->
	<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartRe
solver"/>
```

jobseeker_form.jsp

```
<%@taglib uri="http://www.springframework.org/tags/form"
prefix="frm"%>
<h1 style="color: red; text-align: center;">File Uploading Page</h1>
<frm:form modelAttribute="jsFrm" enctype="multipart/form-data">
	<table align="center" bgcolor="cyan">
		<tr>
			<td>Job Seeker Name : </td>
			<td><frm:input path="jsName"/></td>
		</tr>
		<tr>
			<td>Job Seeker Address : </td>
			<td><frm:input path="jsAddress"/></td>
		</tr>
		<tr>
			<td>Job Seeker Resume : </td>
			<td><input type="file" name="resume"/></td>
		</tr>
		<tr>
			<td>Job Seeker Photo : </td>
			<td><input type="file" name="photo"/></td>
		</tr>
		<tr>
			<td><input type="reset" value="Cancel"/></td>
			<td><input type="submit" value="Upload"/></td>
		</tr>
	</table>
</frm:form>
```

upload_success.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<h1 style="color: red; text-align: center;">Result Page</h1>
<b>The uploaded file names are : ${resumeFileName} &
${photoFileName}</b>
<br><br>
<a href="upload">Home</a>
```

JobSeekerInfo.java

```
package com.nt.model;

import org.springframework.web.multipart.MultipartFile;

import lombok.Data;

@Data
public class JobSeekerInfo {
	private String jsName;
	private String jsAddress;
	private MultipartFile resume;
	private MultipartFile photo;
}
```

pom.xml

```
	<dependencies>
		<!--
https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>javax.servlet-api</artifactId>
			<version>4.0.1</version>
			<scope>provided</scope>
		</dependency>
		<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>jstl</artifactId>
			<version>1.2</version>
		</dependency>
		<!-- https://mvnrepository.com/artifact/commons-
io/commons-io -->
		<dependency>
			<groupId>commons-io</groupId>
			<artifactId>commons-io</artifactId>
			<version>2.8.0</version>
		</dependency>
		<!-- https://mvnrepository.com/artifact/commons-
fileupload/commons-fileupload -->
		<dependency>
			<groupId>commons-fileupload</groupId>
			<artifactId>commons-fileupload</artifactId>
			<version>1.4</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.springframework/spring-webmvc --
>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>5.2.9.RELEASE</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.projectlombok/lombok -->
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<version>1.18.16</version>
			<scope>provided</scope>
		</dependency>
	</dependencies>
```

JobSeekerController.java

```
package com.nt.controller;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Map;
import javax.servlet.ServletContext;

import org.apache.commons.io.IOUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import com.nt.model.JobSeekerInfo;

@Controller
public class JobSeekerController {

        @Autowired
        private ServletContext sc;

        @GetMapping("/upload")
        public String showUploadForm(@ModelAttribute("jsFrm")
JobSeekerInfo info) {
                return "jobseeker_form";
        }

        @PostMapping("/upload")
        public String uploadFiles(Map<String, Object> map,
@ModelAttribute("jsFrm") JobSeekerInfo info) throws Exception {
                String folderLocation = null, resumeName=null,
photoName=null;
                File file = null;
                InputStream resumeIS = null, photoIS = null;
                OutputStream resumeOS = null, photoOS = null;
                //Get upload folder location from web.xml file context param
                folderLocation = sc.getInitParameter("UploadStore");
                //Check weather that folder is available or not, if not create a
new folder
                file = new File(folderLocation);
                //Check store folder availability
                if (!file.exists())
                        //if not there create store folder
                        file.mkdirs();

                //get Uploaded file names
                resumeName = info.getResume().getOriginalFilename();
```

```
                    photoName = info.getPhoto().getOriginalFilename();
                    //create InputStream pointing uploaded file context
                    resumeIS = info.getResume().getInputStream();
                    photoIS = info.getPhoto().getInputStream();
                    //Create OutputStream pointing to empty destination files
                    resumeOS = new
FileOutputStream(folderLocation+"/"+resumeName);
                    photoOS = new
FileOutputStream(folderLocation+"/"+photoName);
                    //copy uploaded context to destination
                    IOUtils.copy(resumeIS, resumeOS);
                    IOUtils.copy(photoIS, photoOS);
                    //Keeps the name of the uploaded files as model attributes
                    map.put("resumeFileName", resumeName);
                    map.put("photoFileName", photoName);

                    //close streams
                    resumeIS.close();
                    resumeOS.close();
                    photoIS.close();
                    photoOS.close();
                    return "upload_success";
          }

}
```

```
<!-- Resolver configuration -->
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResol
ver">
        <property name="maxUploadSize" value="#{350*1024*1024}"/>
        <property name="maxUploadSizePerFile" value="#{20*1024*1024}"/>
</bean>
```
To apply restrictions uploading file sizes will be applied all the files.

Note: To apply restrictions on file types and file size on specific files we need to write t0 Streams customized logic in controller classes
(SPEL is used there to perform arithmetic and logical operations in xml file)

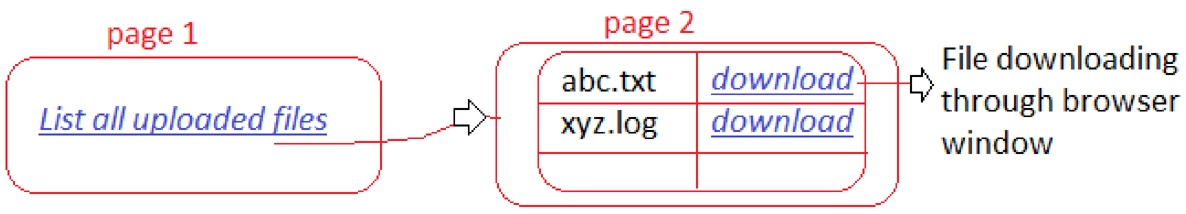
SPEL -> Spring expression Language
Syntax: #{<expr>}

Note: Can be used in "value" attribute of Spring bean configuration file also in @Value () annotation.

To write specific condition for each file write the below code in Controller class

```
//Check condition for Resume
File resumeFile = new File(resumeName);
if (resumeFile.length()>10*1024*1024)
        throw new IllegalStateException();
if (!resumeName.endsWith("pdf")||!resumeName.endsWith("doc"))
        throw new IllegalStateException();
```

## File Downloading



Note: Controller classes gets the ServletContainer managed ServletContext, ServletConfig objects through @Autowired kept on class level instance variable because they are same across the multiple requests and gets request, response objects as the parameters of handler methods because they will change request to request.

Controller class

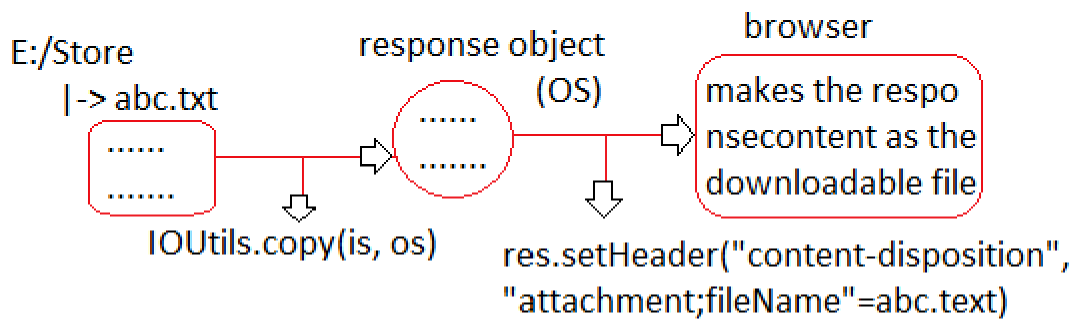
```
@Autowired
private ServletContext sc;
@Autowired
private ServletConfig cg;
@Get/PostMapping("/handle")
public String handleForm(HttpServletRequqest req,HttpServletResponse res){
        ...........
}
```

- The basic thumb rule of file downloading is getting file content into response object and give instruction to browser for making that file content as the downloadable file.

Note:

- ✓ "content-disposition" gives instruction to browser how to handle the received the content.

- ✓ If the value is "inline" then it displays the received content on the browser itself,
- ✓ If the value is "attachment" then it makes the received content as downloadable file.


E:/Store
|-> abc.txt
......
.......
response object
(OS)
......
.......
browser
makes the respo nsecontent as the downloadable file
IOUtils.copy(is, os)
res.setHeader("content-disposition", "attachment;fileName"=abc.text)

- ✓ If certain file types are not having file MIME types like zip files, rar file, icon file and etc. then we can universal/generic mime type called "application/octet-stream".
- ✓ if the handler method returns "null" as the LVN then DS thinks that response is going to browser directly from handler method. So, no need of involving ViewResolver.

- Create service package com.nt.service with a interface IFileMgmtService.java and an implementation class FileMgmtServiceImple.java and create a xml file service-beans.xml with a jsp file show_file.jsp .
- Add the following code with their respective file.

applicationContext.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"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd">
	<import resource="service-beans.xml"/>
</beans>
```

IFileMgmtService.java

```
package com.nt.service;
import java.util.List;
public interface IFileMgmtService {
	public List<String> fetchAllFiles(String uploadStore);
}
```

FileMgmtServiceImpl.java

```java
package com.nt.service;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Service;

@Service("fileService")
public class FileMgmtServiceImpl implements IFileMgmtService {

	@Override
	public List<String> fetchAllFiles(String uploadStore) {
		File file = null, content[]=null;
		List<String> fileNameList = null;
		//Create File object representing uploadStore folder
		file = new File(uploadStore);
		//get all Files and sub folders of uploadStore folder
		content = file.listFiles();
		fileNameList = new ArrayList<>();
		for (File f : content) {
			if (f.isFile())
				fileNameList.add(f.getName());
		}
		return fileNameList;
	}
}
```

service-beans.xml

```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"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">

	<context:component-scan base-package="com.nt.service"/>

</beans>
```

service-beans.xml

```
	@Autowired
	private IFileMgmtService service;

	@GetMapping("list_files")
	public String showAllFiles(Map<String, Object> map) {
		String uploadStore = null;
		List<String> fileNameList = null;
		//get uploadStore Folder name form web.xml file as context
param value
		uploadStore = sc.getInitParameter("UploadStore");
		//use service
		fileNameList = service.fetchAllFiles(uploadStore);
		//keep results in model attribute
		map.put("filesList", fileNameList);
		//return LVN
		return "show_file";
	}

	@GetMapping("/download")
	public String downloadFile(@RequestParam("fname") String
fileName, HttpServletResponse res) throws Exception {
		File file = null;
		String filePath = null, mimeType=null;
		InputStream is = null;
		OutputStream os = null;
		//get Path of the Downloadable file
		filePath = sc.getInitParameter("UploadStore")+"/"+fileName;
		//Create file object pointing to the file to be download
		file = new File(filePath);
		//set file length as response content length
		res.setContentLengthLong(file.length());
		//get MIMEt type of the file
		mimeType = sc.getMimeType(filePath);
		res.setContentType(mimeType==null?"application/octet-
stream":mimeType);
		//create inputStream pointing to the file to be downloaded
		is = new FileInputStream(filePath);
		//create output Stream pointing response object
		os = res.getOutputStream();
		//Give instruction to browser to make there received content
```

```
        as download
                res.setHeader("content-
disposition","attachement;filename="+fileName);
                //copy file content to response object
                IOUtils.copy(is, os);
                //take LVN is null
                return null;
        }
```

Show_file.jsp

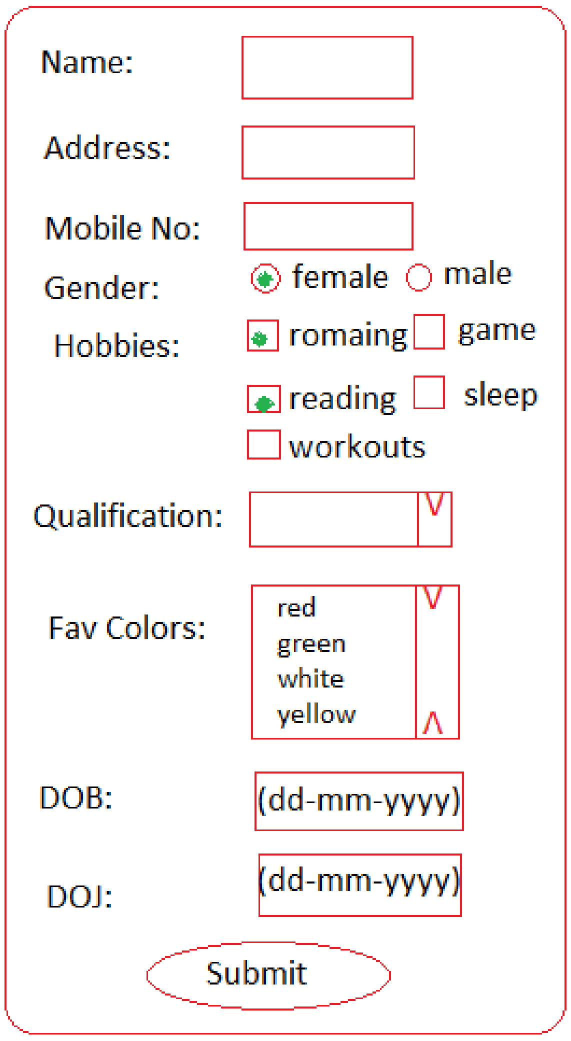
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1" isELIgnored="false"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<h1 style="color: blue; text-align: center;">Show all files to Download</h1>
<c:choose>
        <c:when test="${filesList ne null && !empty filesList}">
                <table align="center" bgcolor="pink">
                        <c:forEach var="fileName" items="${filesList}">
                                <tr>
                                        <td>${fileName}</td>
                                        <td><a
href="download?fname=${fileName}">Download</a></td>
                                </tr>
                        </c:forEach>
                </table>
        </c:when>
        <c:otherwise>
                <h1 style="color: red; text-align: center;">No Files found to
download</h1>
        </c:otherwise>
</c:choose>
```

## Working with @ModelAttribute for dynamic initial Data and @InitBinder for configure PropertyEditor for Spring MVC Application to get used in DataBinding:

- To bind End-user supplied different pattern date values from the form page to Model class object properties/ attributes the built-in convertors or PropertyEditor not sufficient. So, we need work customer editors. To link to these PropertyEditor we need to use @InitBinder methods.

- @ModelAttrbute (-) methods are useful to display dynamic initial values form radio buttons(group), checkboxes(group), select box, list box.

person_form.jsp



Model class for the formpage

```
public class Person{
          private String name;
          private String address;
          private long mobileNo;
          private String gender="female"
          private String[] hobbies=new
               String[]{"sleeping", "reading"};
          private String qualification;
          private String[] colors-new String[]
                              {"red","white"};
          private java.util.Date dob, doj;
          //get setters && getters
          ...............
          .................
}
```

Note:

`<frm:radiobuttons path="gender" items="${gendersList}"/>`

- frm:radiobuttons*: Gives multiple radion buttons that are grouped into single unit.*
- path="*gender*"*: Grouped radio buttons name. The model class property name to which selected radio button value should be bound.*
- items="${gendersList}"*: The ModelAttribute name that acts as reference Data to display bunch of radio buttons with values and labels. (collected from @ModelAttribute("genderList") method of controller class).*

Internal generated code:

```
<input type="radio" name="gender" value="female" checked> female
<input type="radio" name="gender" value="male" > male
```

Note:

- ✓ @ModelAttrbute methods executes for every request i.e. for the initial phase request that launches form page and for the post back request that submits the form page these methods supplied reference data is very useful while displaying form page normally or with validation error messages because in both cases the radio buttons, checkboxes, list boxes, select boxes should get dynamic initial values.
- ✓ By default only mm/dd/yyyy pattern date values will be bound with Model class object java.util.Date type properties by taking the support of built-in property editors, for other date patterns we need to register Custom PropertyEditor explicitly with ServletRequestDataBinder object (It is responsible to bind form data to Model class object) by using @InitBinder method as shown below.

```
@InitBinder
public void myInitBinder(ServletRequestDataBinder binder) {
        System.out.println("PersonController.myInitBinder()");
        SimpleDateFormat sdf = null;
        sdf =new SimpleDateFormat("dd-mm-yyyy");
        binder.registerCustomEditor(Date.class, new
                                        CustomDateEditor(sdf, true));
}
```

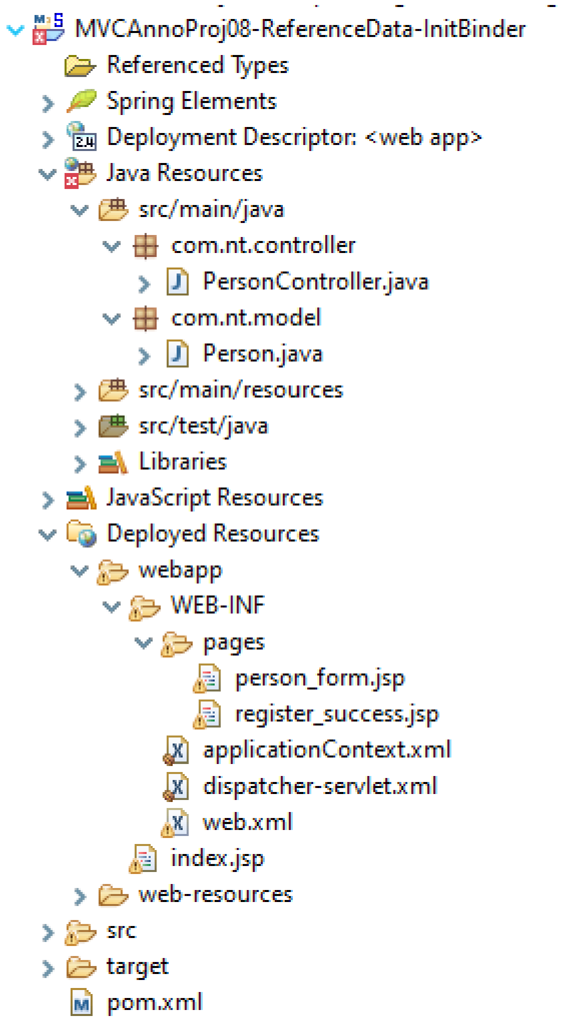
CustomDateEditor: Readymade Property editor to convert given pattern String date values to java.util.Date class object

- ✓ @InitBinder method executes in form launching to display initial values form components according to property editor conversion also executes in form submission to convert the form component supplied String values as required for model class property.
- ✓ we can use HTML5 components in Spring MVC form tag library by specifying "type" attribute in <frm:input> tag

```
<tr>
        <td>DOB (dd-mm-yyyy):</td>
        <td><frm:input type="date" path="dob" /></td>
</tr>
<tr>
        <td>DOJ (dd-mm-yyyy):</td>
        <td><frm:input type="date" path="doj" /></td>
</tr>
<tr>
        <td>Salary Range:</td>
```

```
            <td><frm:input type="range" min="10000" max="100000"
path="salary" /></td>
        </tr>
```

## Directory Structure of MVCAnnoProj08-ReferenceData-InitBinder:



```
MVCAnnoProj08-ReferenceData-InitBinder
    Referenced Types
  > Spring Elements
  > Deployment Descriptor: <web app>
  v Java Resources
    v src/main/java
      v com.nt.controller
        > PersonController.java
      v com.nt.model
        > Person.java
    > src/main/resources
    > src/test/java
    > Libraries
  > JavaScript Resources
  v Deployed Resources
    v webapp
      v WEB-INF
        v pages
            person_form.jsp
            register_success.jsp
          applicationContext.xml
          dispatcher-servlet.xml
          web.xml
        index.jsp
    > web-resources
  > src
  > target
    pom.xml
```

- Develop the above directory structure and folder, package, class, XML, JSP, file after convert to web application.
- Copy the common files like applicationContext.xml, dispatcher-servlet and web.xml, pom.xml from the other project
- Add the following code with their respective files.

index.jsp

```
<jsp:forward page="person"/>
```

person_form.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://www.springframework.org/tags/form"
prefix="frm"%>
<h1 style="color:red; text-align: center;">Person Details Form</h1>
<frm:form modelAttribute="perFrm">
        <table align="center" bgcolor="cyan">
                <tr>
                        <td>Person Name:</td>
                        <td><frm:input path="name" /></td>
                </tr>
                <tr>
                        <td>Person Address:</td>
                        <td><frm:input path="address" /></td>
                </tr>
                <tr>
                        <td>Person Mobile Number:</td>
                        <td><frm:input path="mobileNo" /></td>
                </tr>
                <tr>
                        <td>Person Gender:</td>
                        <td><frm:radiobuttons path="gender"
items="${gendersList}" /></td>
                </tr>
                <tr>
                        <td>Person Hobbies:</td>
                        <td><frm:checkboxes path="hobbies"
items="${hobbiesList}" /></td>
                </tr>
                <tr>
                        <td>Person Qualification:</td>
                        <td><frm:select path="qualification">
                                        <frm:options items="${qualificationsList}" />
                                </frm:select></td>
                </tr>
                <tr>
                        <td>Favorite color:</td>
                        <td><frm:select path="colors" multiple="multiple">
                                        <frm:options items="${colorsList}" />
                        </frm:select></td>
                </tr>
                <tr>
                        <td>DOB (dd-mm-yyyy):</td>
                        <td><frm:input type="date" path="dob" /></td>
                </tr>
                <tr>
                        <td>DOJ (dd-mm-yyyy):</td>
                        <td><frm:input type="date" path="doj" /></td>
                </tr>
                <tr>
                        <td>Salary Range:</td>
                        <td><frm:input type="range" min="10000"
max="100000" path="salary" /></td>
                </tr>
                <tr>
                        <td colspan="2"><input type="submit" value="Register"
/></td>
                </tr>
        </table>
</frm:form>
```

register_success.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>

<h1 style="color: red; text-align: center;">Result Page</h1>
Model Class object data or form data : ${perFrm}
<br>
<a href="person">Home</a>
```

Person.java

```
package com.nt.model;

import lombok.Data;

@Data
public class Person {
        private String name;
        private String address;
```

```
	private long mobileNo;
	private String gender="female";
	private String[] hobbies=new String[]{"sleeping", "reading"};
	private String qualification;
	private String[] colors=new String[]{"red", "white"};
	private java.util.Date dob, doj;
	public Float salary;
}
```

PersonController.java

```
package com.nt.controller;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.Map;

import org.springframework.beans.propertyeditors.CustomDateEditor;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.ServletRequestDataBinder;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import com.nt.model.Person;

@Controller
public class PersonController {

	@GetMapping("/person")
	public String showHomePage(@ModelAttribute("perFrm") Person 
person) {
		return "person_form";
	}

	@PostMapping("/person")
	public String handleForm(Map<String, Object> map, 
@ModelAttribute("perFrm") Person per) {
		System.out.println("PersonController.handleForm()");
		return "register_success";
	}
```

```
        @ModelAttribute("gendersList")
        public List<String> populateGenders() {
                List<String> gendersList = List.of("Female", "Male");
                return gendersList;
        }

        @ModelAttribute("hobbiesList")
        public List<String> populateHobbies() {
                List<String> hobbiesList = List.of("reading", "romaing",
"playing", "workouts", "sleeping");
                return hobbiesList;
        }

        @ModelAttribute("qualificationsList")
        public List<String> populateQualifications() {
                List<String> qualificationsList = List.of("BE", "MCA", "BCA",
"M.Tech", "M.Sc");
                return qualificationsList;
        }

        @ModelAttribute("colorsList")
        public List<String> populateColors() {
                List<String> colorsList = List.of("Blue", "white", "Green", "Red",
"Yellow", "Orange");
                return colorsList;
        }

        @InitBinder
        public void myInitBinder(ServletRequestDataBinder binder) {
                System.out.println("PersonController.myInitBinder()");
                SimpleDateFormat sdf = null;
                sdf =new SimpleDateFormat("dd-mm-yyyy");
                binder.registerCustomEditor(Date.class, new
CustomDateEditor(sdf, true));
        }

}
```

## I18N in Spring MVC

- Making our App working for different locales is called working with I18N.
- We can take care the following things while dealing with I18N.
    a. presentation labels
    b. date, time formats
    c. number formats
    d. currency symbols
    e. indentation
- Spring MVC gives built-in support to work with I18N

Step 1: Prepare multiple Locale specific Properties files having base file with common keys and different values collected from google Translator.

App.properties (base)
App_fr_FR.properties
App_de_DE.properties
App_hi_IN.properties
App_zh_CN.properties

- Base name should be placed in all properties file.
- properties files should have same keys but values can be changed.

Step 2: Configure base properties file in dispatcher-servlet.xml

```
        <!-- Configure properties file -->
        <bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
                <property name="basename" value="com/nt/commons/App"/>
        </bean>
```

Step 3: Enable I18n on spring MVC App by configuring "SessionLocaleResolver" having fixed bean id "localeResolver"

```
        <!-- Configure Resolver to enable I18N on the Application -->
        <bean id="localeResolver"
class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
                <property name="defaultLocale" value="en"></property>
        </bean>
```

Enables I18n on the spring MVC web application, also takes default locale in this order.

- Session attribute "locale" value (if not there then).
- defaultLocale property value (injected) (if not thee then).
- request header "accept" value.

Note: If given locale specific properties file is not available then it takes base properties as default file.

Step 4: Configure "LocaleChangeInterceptor" to change I18n settings for every request-based locale info that is coming along with request having given request param name.

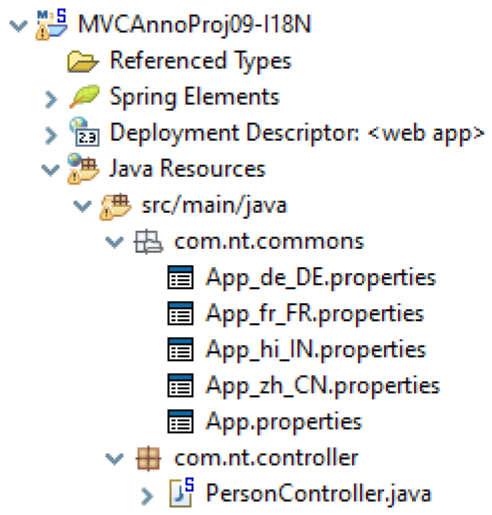
```
        <!-- Configure properties file -->
        <bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
                <property name="basename" value="com/nt/commons/App"/>
        </bean>
```

Step 5: Map the above Handler Interceptor to Controller class through handler Mapping.

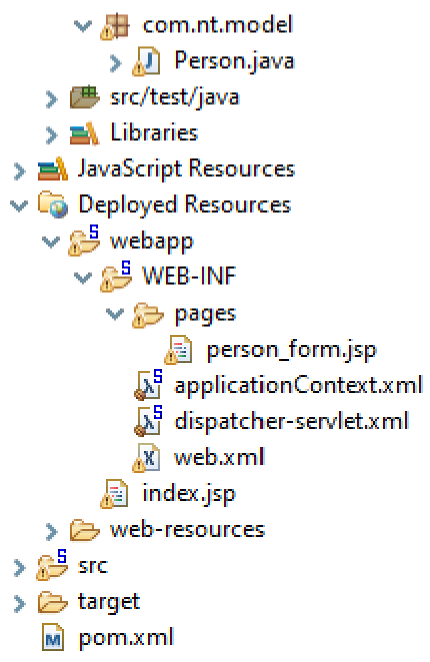
```
        <!-- Configure Handler mapping -->
        <bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestMap
pingHandlerMapping">
                <property name="interceptors">
                        <array>
                                <ref bean="lci"/>
                        </array>
                </property>
        </bean>
```

Step 6: Read message from active properties file by specifying keys and display them on the webpage using <spring:message> tag. (Belongs to generic tag library).

Directory Structure of MVCAnnoProj09-I18N:



```
MVCAnnoProj09-I18N
    Referenced Types
  > Spring Elements
  > Deployment Descriptor: <web app>
  v Java Resources
    v src/main/java
      v com.nt.commons
          App_de_DE.properties
          App_fr_FR.properties
          App_hi_IN.properties
          App_zh_CN.properties
          App.properties
      v com.nt.controller
        > PersonController.java
```



```
        com.nt.model
            Person.java
    src/test/java
    Libraries
JavaScript Resources
Deployed Resources
    webapp
        WEB-INF
            pages
                person_form.jsp
            applicationContext.xml
            dispatcher-servlet.xml
            web.xml
        index.jsp
    web-resources
src
target
pom.xml
```

- Develop the above directory structure and folder, package, class, XML, JSP, file after convert to web application.
- Copy the common files like applicationContext.xml, dispatcher- and web.xml, pom.xml, index.jsp from the previous project
- Add the following code with their respective files.

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"
        xsi:schemaLocation="http://www.springframework.org/schema/bean
s https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
                http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-
4.3.xsd">

        <!-- Configure Handler mapping -->
        <bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestM
appingHandlerMapping">
                <property name="interceptors">
                        <array>
                                <ref bean="lci"/>
                        </array>
                </property>
        </bean>

        <!-- Controller -->
        <context:component-scan base-package="com.nt.controller"/>

        <!-- Configure View Resolver -->
        <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver
">
                <property name="prefix" value="/WEB-INF/pages/"/>
                <property name="suffix" value=".jsp"/>
        </bean>

        <!-- Configure Resolver to enable I18N on the Application -->
        <bean id="localeResolver"
class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
                <property name="defaultLocale" value="en"></property>
        </bean>

        <!-- Configure properties file -->
        <bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSourc
e">
                <property name="basename"
value="com/nt/commons/App"/>
        </bean>

        <!-- Configure LocalChangeInterceptor to change Local Settings for
every request -->
        <bean id="lci"
class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
                <property name="paramName" value="lang"/>
        </bean>

</beans>
```

person_form.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="ISO-8859-1"%>
```

```
<%@taglib uri="http://www.springframework.org/tags/form"
prefix="frm"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>

<h1 style="color:red; text-align: center;"><spring:message
code="person.title"/></h1>
<frm:form modelAttribute="perFrm">
	<table align="center" bgcolor="cyan">
		<tr>
			<td><spring:message code="person.name"/>:</td>
			<td><frm:input path="name" /></td>
		</tr>
		<tr>
			<td><spring:message code="person.address"/>:</td>
			<td><frm:input path="address" /></td>
		</tr>
		<tr>
			<td><spring:message code="person.mobileNo"/>:</td>
			<td><frm:input path="mobileNo" /></td>
		</tr>
		<tr>
			<td colspan="2"><input type="submit"
value="<spring:message code="person.submit.cap"/>" /></td>
		</tr>
	</table>
</frm:form>
<div style="text-align: center;">
	<a href="?lang=fr_FR">French</a> &nbsp;&nbsp;
	<a href="?lang=hi_IN">German</a> &nbsp;&nbsp;
	<a href="?lang=fr_FR">Hindi</a> &nbsp;&nbsp;
	<a href="?lang=zh_CN">Chinese</a> &nbsp;&nbsp;
</div>
```

App.properties

```
#Base File english
person.title=Person Registration page
person.name=Person Name
person.address=Person Address
person.mobileNo=Person Mobile Number
person.submit.cap=Registration Person
```

App_de_DE.properties

```
#Base File German
person.title = Seite zur Personenregistrierung
person.name = Personenname
person.address = Personenadresse
person.mobileNo = Handynummer der Person
person.submit.cap = Registrierungspersona
```

App_fr_FR.properties

```
#Base File French
person.title = Page d'inscription de la personne
person.name = Nom de la personne
person.address = Adresse de la personne
person.mobileNo = Numéro de mobile de la personne
person.submit.cap = Personne inscritea
```

App_hi_IN.properties

```
#Base File Hindi
person.title = \u0935\u094D\u092F\u0915\u094D\u0924\u093F
\u092A\u0902\u091C\u0940\u0915\u0930\u0923
\u092A\u0943\u0937\u094D\u0920
person.name = \u0935\u094D\u092F\u0915\u094D\u0924\u093F
\u0915\u093E \u0928\u093E\u092E
person.address = \u0935\u094D\u092F\u0915\u094D\u0924\u093F
\u0915\u093E \u092A\u0924\u093E
person.mobileNo = \u0935\u094D\u092F\u0915\u094D\u0924\u093F
\u092E\u094B\u092C\u093E\u0907\u0932 \u0928\u0902\u092C\u0930
person.submit.cap = \u092A\u0902\u091C\u0940\u0915\u0930\u0923
\u0935\u094D\u092F\u0915\u094D\u0924\u093F
```

App_zh_CN.properties

```
#Base File Chinese
person.title =\u4EBA\u5458\u6CE8\u518C\u9875\u9762
person.name =\u4EBA\u540D
person.address =\u4EBA\u5458\u5730\u5740
person.mobileNo =\u4EBA\u5458\u624B\u673A\u53F7\u7801
person.submit.cap =\u6CE8\u518C\u4EBA
```

Person.java

```
package com.nt.model;

import java.util.Date;

import org.springframework.format.annotation.DateTimeFormat;

import lombok.Data;

@Data
public class Person {
	private String name;
	private String address;
	private Long mobileNo;
}
```

Person.java

```
package com.nt.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;

import com.nt.model.Person;

@Controller
public class PersonController {

	@GetMapping("/person")
	public String showFormPage(@ModelAttribute("perFrm") Person
person) {
		return "person_form";
	}

}
```

To Format Date and Number, we have to use format JSTL taglib like below we have to use

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
	pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://www.springframework.org/tags/form"
prefix="frm"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
```

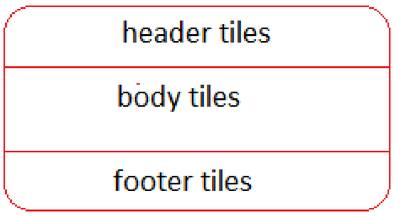
```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

<fmt:setLocale value="${pageCotext.response.locale}"/>
<jsp:useBean id="dt" class="java.util.Date"/>
<fmt:formatDate var="fdt" value="${dt}" dateStyle="FULL" type="both"/>
Formatted Date: ${fdt}
<br>
<fmt:formatNumber var="fno" value="3456789080878654"
type="currency"/>
Fomratted Number: ${fno}
```

## Spring MVC with Tiles

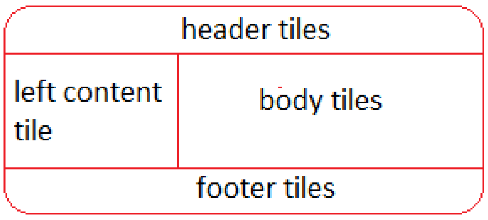
- A Tile is logical patriation in the web page like header tile, footer tile, body tile and etc.
- Tiles framework is third party framework to build web pages having tiles with layout control and composite view design pattern.
- Composite view design pattern says every web page comes as the output given by multiple web components (view components) together.
- Layout control means all web pages are designed based on the common layout having tiles and these tile values are supplied by different JSP pages (view components) more every. if modify anything in the layout/ template page, it will reflect to all the web pages.

### Popular Layouts:

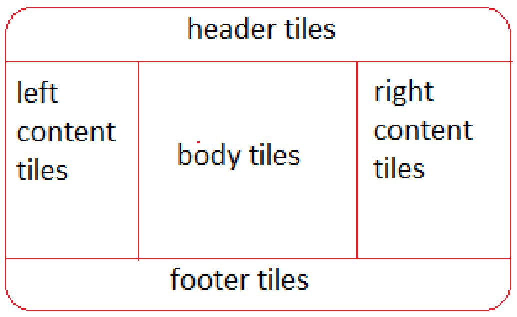
Classic Layout



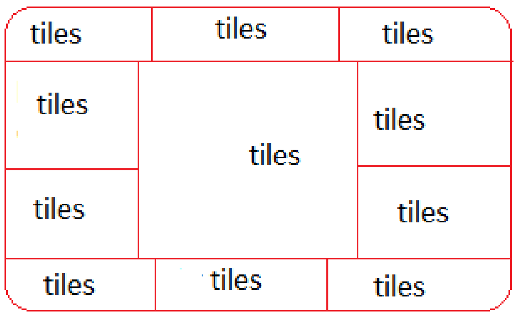
Two Column Layout



Three Column Layout



Circular Layout



Advantages of tiles framework:

a. Layout control because of layout page/ template page.
b. Composite view design pattern implementation.
c. Uniform look for web pages.
d. We can integrate this framework with Struts, JSF, Spring MVC and etc.
e. Inheritance b/w tile definitions is possible.
   and etc.

## Steps to work Spring MVC with Tiles

Step 1: First decide how many web pages are the in website (4 web pages)

Step 2: Design layout page/ template page having tiles with the support tiles tag library.

/WEB-INF/pages/layout.jsp (classic layout) (j) (10)

```
<%@taglib uri="........................." prefix="tiles"%>
<table border="0" width="100%" height="100%">
        <tr height="20%">
                <td><tiles:insertAttribute name="header"/>
        </tr>
        <tr height="70%">
                <td><tiles:insertAttribute name="body"/>
        </tr>
        <tr height="10%">
                <td><tiles:insertAttribute name="footer"/>
        </tr>
</table>
```

Executes this layout.jsp having following tile values to display the web page. (j)

Header -> /WEB-INF/pages/header.jsp
footer -> /WEB-INF/pages/footer. html
body -> /WEB-INF/pages/page1_body.jsp (10)

Executes this layout.jsp having following tile values to display the web page.

header -> /WEB-INF/pages/header.jsp
footer -> /WEB-INF/pages/footer. html
body -> /WEB-INF/pages/page2_body.jsp

Step 3: Activate Tiles framework by configure "TilesConfigurer" as spring bean in dispatcher-servlet.xml file

dispatcher-servlet.xml

```
<bean class="org.springframework.web.servlet.view.tiles3.TilesConfigurer">
        <property name="definitions">
                <list>
                        <value>/WEB-INF/tiles.xml</value>
                </list>             tile definition configuration file
        </property>
</bean>
```

Step 4: Develop tile definitions in tile definition configuration file specifying one tile definition for each web page.

WEB-INF/tiles.xml

```
<tiles-definitions>
        <definition name="page1Def" template="/WEB-
                                        INF/pages/layout.jsp"/>
                <put-attribute name="header" value="/WEB-
                                        INF/pages/header.jsp"/>
                <put-attribute name="body" value="/WEB-
                                        INF/pages/page1_body.jsp"/>
                <put-attribute name="footer" value="/WEB-
                                        INF/pages/footer.html"/>
        </definition>
        <definition name="page2Def" template="/WEB-
                                        INF/pages/layout.jsp"/>
                <put-attribute name="header" value="/WEB-
                                        INF/pages/header.jsp"/>
                <put-attribute name="body" value="/WEB-
                                        INF/pages/page2_body.jsp"/>
                <put-attribute name="footer" value="/WEB-
                                        INF/pages/footer.html"/>
        </definition>
        <definition name="page3Def" template="/WEB-
                                        INF/pages/layout.jsp"/>
                <put-attribute name="header" value="/WEB-
                                        INF/pages/header.jsp"/>
                <put-attribute name="body" value="/WEB-
                                        INF/pages/page3_body.jsp"/>
                <put-attribute name="footer" value="/WEB-
                                        INF/pages/footer.html"/>
        </definition>
</tile-definitions>
```

improved code using tiles inheritance

```
<tile-definitions>
        <definition name="baseDef" template="/WEB-INF/pages/layout.jsp/>
                <put-attribute name="header" value="/WEB-
```

```
                                        INF/pages/header.jsp"/>
                <put-attribute name="body" value=""/>
                <put-attribute name="footer" value="/WEB-
                                        INF/pages/footer.html"/>
        </definition>     (i)
        <definition name="page1D" template="/WEB-INF/pages/layout.jsp"/>
                <put-attribute name="body" value="/WEB-
                                        INF/pages/page1_body.jsp"/>
        </definition>              (9)
        <definition name="page2D" template="/WEB-INF/pages/layout.jsp"/>
                <put-attribute name="body" value="/WEB-
                                        INF/pages/page2_body.jsp"/>
        </definition>
        <definition name="page3D" template="/WEB-INF/pages/layout.jsp"/>
                <put-attribute name="body" value="/WEB-
                                        INF/pages/page3_body.jsp"/>
        </definition>
</tile-definitions>
```

Step 5: Configure "TilesViewResolver" to render web pages based on tile definitions.

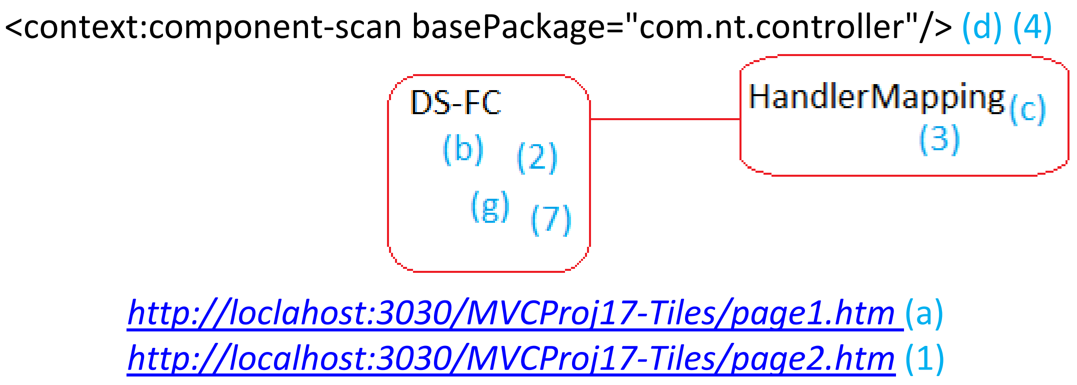
dispatcher-servlet.xml

```
<bean                                                        (8)
 class="org.springframework.web.servlet.view.tiles3.TilesViewResolver"/> (h)
```

Step 6: Develop Controller classes having "tile definition" name as the LVN in the handler methods.

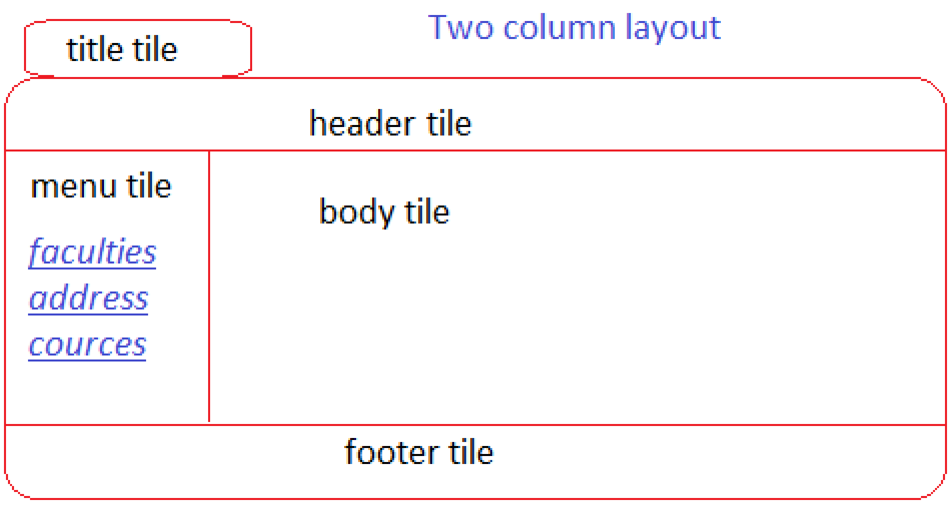
```
@Controller
public class PersonController{
        @GetMapping("/page1") (e) (5)
        public String showPage1(){
        (f) (6)    return "page1Def"; // tile definition name as the LVN
        }
        @GetMapping("/page2")
        public String showPage2(){
                return "page2Def"; // tile definition name as the LVN
        }
        @GetMapping("/page3")
        public String showPage3(){
                return "page3Def"; // tile definition name as the LVN
        }
        @GetMapping("/page4")
        public String showPage4(){
                return "page4Def"; // tile definition name as the LVN
        }
}
```

Step 6: Develop all view components (HTML, JSP files) having logics.

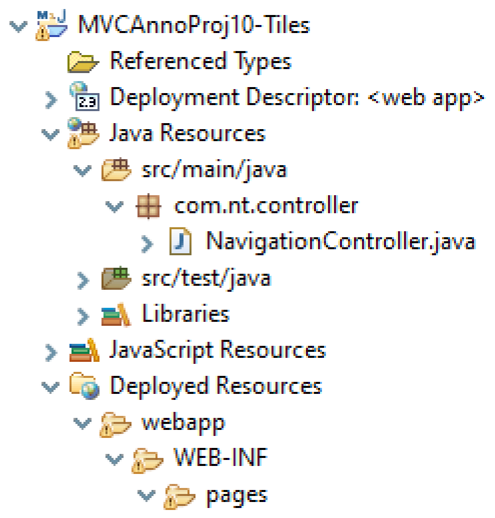


<context:component-scan basePackage="com.nt.controller"/> (d) (4)
DS-FC
(b) (2)
(g) (7)
HandlerMapping (c)
(3)
http://loclahost:3030/MVCProj17-Tiles/page1.htm (a)
http://localhost:3030/MVCProj17-Tiles/page2.htm (1)

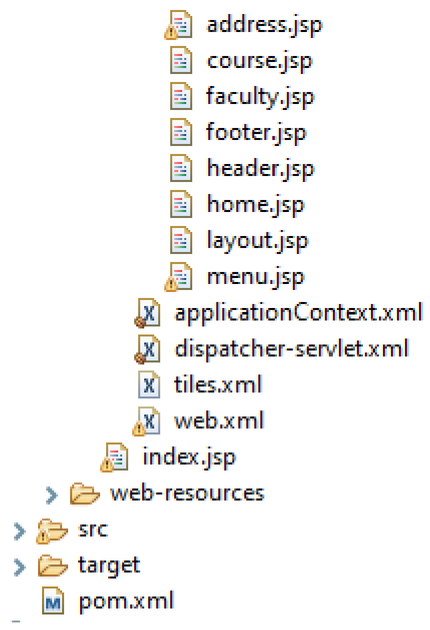
## Spring MVC with Tiles Example Web Application:



## Directory Structure of MVCAnnoProj10-Ties:



```
MVCAnnoProj10-Tiles
    Referenced Types
  > Deployment Descriptor: <web app>
  v Java Resources
      v src/main/java
          v com.nt.controller
              > NavigationController.java
      > src/test/java
      > Libraries
  > JavaScript Resources
  v Deployed Resources
      v webapp
          v WEB-INF
              v pages
```


```
            address.jsp
            course.jsp
            faculty.jsp
            footer.jsp
            header.jsp
            home.jsp
            layout.jsp
            menu.jsp
          applicationContext.xml
          dispatcher-servlet.xml
          tiles.xml
          web.xml
        index.jsp
    > web-resources
> src
> target
  pom.xml
```

- Develop the above directory structure and folder, package, class, XML, JSP, file after convert to web application.
- Copy the common files like applicationContext.xml, web.xml, pom.xml (add tiles-jsp.<version>.jar), index.jsp from the other project.
- Add the following code with their respective files.

tiles.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tiles-definitions PUBLIC
    "-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN"
    "http://tiles.apache.org/dtds/tiles-config_3_0.dtd">

<tiles-definitions>
        <definition name="baseDef" template="/WEB-INF/pages/layout.jsp">
                <put-attribute name="title" value="" type="string"/>
                <put-attribute name="header" value="/WEB-
INF/pages/header.jsp"/>
                <put-attribute name="footer" value="/WEB-
INF/pages/footer.jsp"/>
                <put-attribute name="body" value=""/>
                <put-attribute name="menu" value="/WEB-
INF/pages/menu.jsp"/>
        </definition>
        <definition name="homeDef" extends="baseDef">
                <put-attribute name="title" value="Home Page"
type="string"/>
```

```
                <put-attribute name="body" value="/WEB-
INF/pages/home.jsp"/>
        </definition>
        <definition name="facultyDef" extends="baseDef">
                <put-attribute name="title" value="Faculty details"
type="string"/>
                <put-attribute name="body" value="/WEB-
INF/pages/faculty.jsp"/>
        </definition>
        <definition name="addressDef" extends="baseDef">
                <put-attribute name="title" value="Address page"
type="string"/>
                <put-attribute name="body" value="/WEB-
INF/pages/address.jsp"/>
        </definition>
        <definition name="courseDef" extends="baseDef">
                <put-attribute name="title" value="Course page"
type="string"/>
                <put-attribute name="body" value="/WEB-
INF/pages/course.jsp"/>
        </definition>
</tiles-definitions>
```

dispatcher-servlet.xml

```
        <!-- Activate tiles frameworks container -->
        <bean
class="org.springframework.web.servlet.view.tiles3.TilesConfigurer">
                <property name="definitions">
                        <array>
                                <value>/WEB-INF/tiles.xml</value>
                        </array>
                </property>
        </bean>

        <!-- Controller -->
        <context:component-scan base-package="com.nt.controller"/>
        <!-- Configure View Resolver -->
        <bean
class="org.springframework.web.servlet.view.tiles3.TilesViewResolver"/>
```

menu.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<br>
<a href="faculty">Faculties</a><br>
<a href="address">Address</a><br>
<a href="course">Cources</a><br>
```

layout.jsp

```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
<head>
        <title><tiles:insertAttribute name="title"/></title>
</head>
<table style="height: 100%; width: 100%; border: 0;">
        <tr height="20%" bgcolor="cyan">
                <td colspan="2"><tiles:insertAttribute name="header"/></td>
        </tr>
        <tr height="70%" bgcolor="pink">
                <td width="30%"><tiles:insertAttribute name="menu"/></td>
                <td width="70%"><tiles:insertAttribute name="body"/></td>
        </tr>
        <tr height="10%" bgcolor="yellow">
                <td colspan="2"><tiles:insertAttribute name="footer"/></td>
        </tr>
</table>
```

home.jsp

```
<h1 style="color: red; text-align: center;">Welcome to Naresh IT</h1>
```

header.jsp

```
<marquee direction="left"><h1 style="color: red; text-align: center;
background-color: cyan;">Naresh Technology</h1></marquee>
```

footer.jsp

```
<b><i style="text-align: center;">&copy; All right reserved 2012-
2020</i></b>
```

faculty.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<h1 style="color: red; text-align: center;">Faculties page</h1>
<ul>
<li>HK sir </li>
<li>Natraz sir </li>
<li>Vijay sir </li>
<li>Raghu sir </li>
<li>Murali sir </li>
<li>Sudhakar sir </li>
<li>Vekatesh sir </li>
</ul>
```

course.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<h1 style="color: red; text-align: center;">Cources page</h1>
<ul>
        <li>Complete Java </li>
        <li>Complete .Net </li>
        <li>Complete Python</li>
        <li>Oracle</li>
        <li>Complete UI technologies</li>
        <li>C</li>
        <li>AWS and Devops</li>
        <li>Data Structure</li>
</ul>
```

address.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<h1 style="color: red; text-align: center;">Cources page</h1>
<pre>
<h2>India – Hyderabad Office</h2>
2nd Floor, Durga Bhavani Plaza, Ameerpet, HyderabadTel: +91 40 2374 6666
(IN – Hyderabad)
Tel: +91 40 2373 4842 (IN – Hyderabad)
+91 9000994007
```

```
+91 9000994008
for Projects : +91 9000994005
Email: info@nareshit.com
<h2>India – Chennai Office</h2>
2nd Floor Plot No.172 & 173, Above Axis Bank, Behind PTC Bus Stop, OMR,
Thoraipakkam, Tamil Nadu, Chennai – 600097.
Mobile/Whats App:  +91 9566042345
Email: chennai@nareshit.com
<h2>USA Office</h2>
5007 Arbor View Pkwy NW Acworth, GA, 30101
Ph: +1 404-232-9879, +1 248-522-6925
Email: sriram@nareshit.com
for Online Training
Mobile/Whats App:
+91 81 79 19 1999
+91 92 93 22 6789
Email: online@nareshit.com
<h2> India – Software Development Office</h2>
Nacre Software Services Pvt Ltd,
#7-1-212/A/69, Plot No:84,
Shivabagh, Ameerpet,
Hyderabad – 500038.
Tel: +91 40 4015 1017
Email: info@nacreservices.com
</pre>
```

NavigationController.java

```java
package com.nt.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class NavigationController {

	@GetMapping("/home")
	public String showHome() {
		return "homeDef";
	}

	@GetMapping("/address")
```

```
        public String showAddress() {
                return "addressDef";
        }

        @GetMapping("/faculty")
        public String showFaculty() {
                return "facultyDef";
        }

        @GetMapping("/course")
        public String showcourse() {
                return "courseDef";
        }

}
```

## Working with ViewResolver and Views

- In Spring MVC View an abstract entity i.e. we can take anything as view component like JSP files, Servlet components, HTML files, Thymeleaf, freemarker, velocity, Spring beans (Java classes) and etc.
- To make Java class as View component we need to make java class implementing "View (I)" and should configure that class as Spring bean, we should also use "BeanNameViewResolver"
- To render response in the form of pdf docs or excel docs we need to use third party APIs like POI (excel), Text (pdf) in our components writing java code in jsp pages is very bad practice.
- So, prefer taking java class/ Spring beans as view comps to place third party code.
- Developing Java classes/ Spring beans as view components, it is recommended to not implement View (I) directly prefer extending them from AbstractXxxView classes which internally takes care of common logics. The classes are like AbstractXlsView, AbstractPdfView and etc.

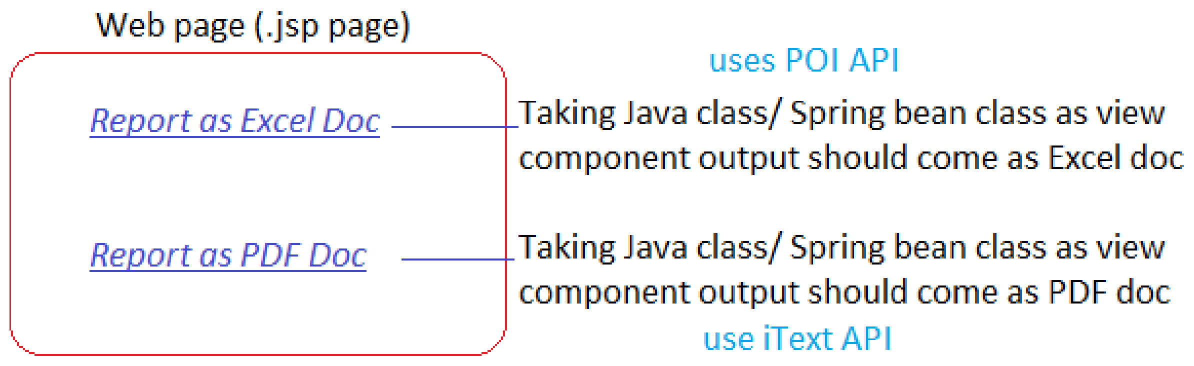
```
@Component("excelView")
public class MyExcelView extenes AbstractXlsView {
        public void buildExcelDocument(Map<String,Object> model,
                        org.apache.poi.ss.usermodel.Workbook workbook,
                        HttpServletRequest request, HttpServletResponse
                                        response) throws Exception {
                ...................
        }
}

@Component("pdfView")
public class MyPdfView extends AbstractPdfView {
        public void buildPdfDocument(Map<String,Object> model,
                                com.lowagie.text.Document document,
                                        com.lowagie.text.pdf.PdfWriter writer,
                        HttpServletRequest request, HttpServletResponse
                                                response) throws Exception {
                ………………..
        }
}
```

dispatcher-servlet.xml

```
<bean class="pkg.BeanNameViewResolver"/>
su ports ViewResolver chaining
//other view resolvers configure like InternalResourceviewResolver
```

Note: BeanNameViewResolver - Takes the LVN given by Controller class to DS and searches for view comp class/spring bean whose bean id is matching with LVN, then takes that View class object to call render (-) method on it. This render (-) internally calls buildXxxDocument(…….) methods

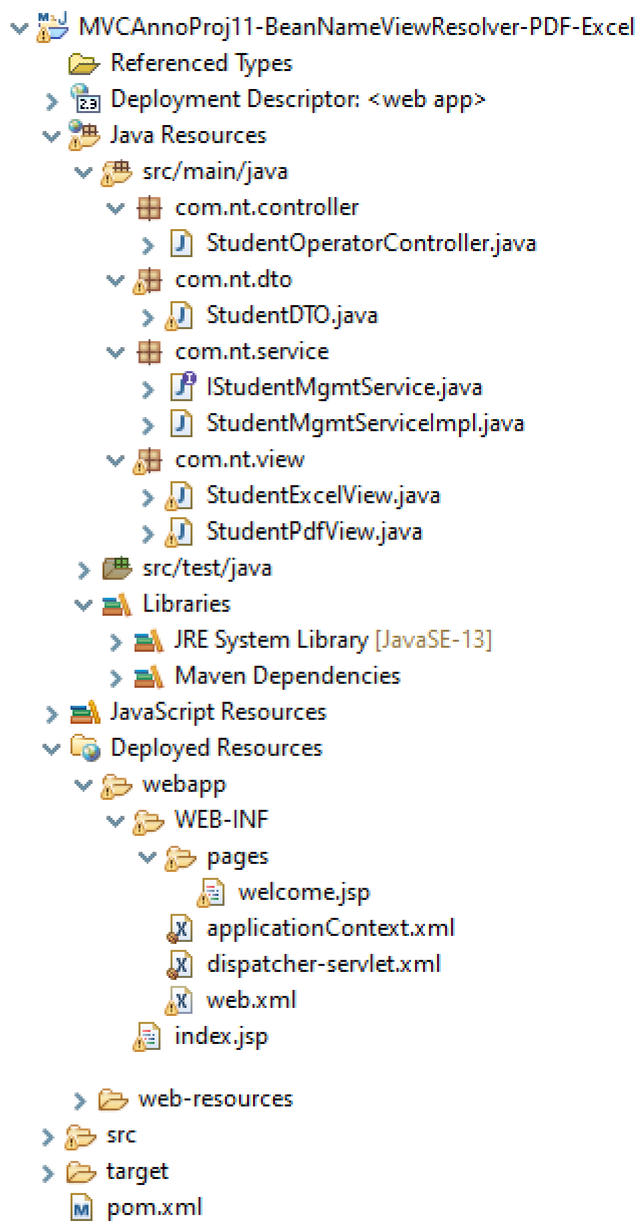
## Example Application:


Web page (.jsp page)
Report as Excel Doc
Report as PDF Doc
uses POI API
Taking Java class/ Spring bean class as view component output should come as Excel doc
Taking Java class/ Spring bean class as view component output should come as PDF doc
use iText API

## Directory Structure of MVCAnnoProj11-BeanNameViewResolver-PDF-Excel:

- Develop the below directory structure and folder, package, class, XML, JSP, file after convert to web application.
- Copy the common files like applicationContext.xml, web.xml, pom.xml, dispatcher-servlet.xml, index.jsp from the other project.

- Add the following code with their respective files.


```
MVCAnnoProj11-BeanNameViewResolver-PDF-Excel
    Referenced Types
  > Deployment Descriptor: <web app>
  v Java Resources
    v src/main/java
      v com.nt.controller
        > StudentOperatorController.java
      v com.nt.dto
        > StudentDTO.java
      v com.nt.service
        > IStudentMgmtService.java
        > StudentMgmtServiceImpl.java
      v com.nt.view
        > StudentExcelView.java
        > StudentPdfView.java
    > src/test/java
    v Libraries
      > JRE System Library [JavaSE-13]
      > Maven Dependencies
  > JavaScript Resources
  v Deployed Resources
    v webapp
      v WEB-INF
        v pages
            welcome.jsp
          applicationContext.xml
          dispatcher-servlet.xml
          web.xml
        index.jsp

    > web-resources
  > src
  > target
    pom.xml
```

pom.xml

```
        <dependencies>
                <!--
https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>javax.servlet-api</artifactId>
                        <version>4.0.1</version>
                        <scope>provided</scope>
                </dependency>
                <!--
```

```
https://mvnrepository.com/artifact/org.springframework/spring-webmvc --
>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>5.2.9.RELEASE</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.projectlombok/lombok -->
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<version>1.18.16</version>
			<scope>provided</scope>
		</dependency>
		<!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->
		<dependency>
			<groupId>org.apache.poi</groupId>
			<artifactId>poi</artifactId>
			<version>4.1.2</version>
		</dependency>
		<!-- https://mvnrepository.com/artifact/com.lowagie/itext -->
		<dependency>
			<groupId>com.lowagie</groupId>
			<artifactId>itext</artifactId>
			<version>2.1.7</version>
		</dependency>
	</dependencies>
```

dispatcher-servlet.xml

```
	<!-- Controller -->
	<context:component-scan base-package="com.nt.controller,
com.nt.view"/>

	<!-- Configure View Resolver -->
	<bean
class="org.springframework.web.servlet.view.BeanNameViewResolver"/>
	<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver
">
```

```
            <property name="prefix" value="/WEB-INF/pages/"/>
            <property name="suffix" value=".jsp"/>
        </bean>
```

welcome.jsp

```
<h1 style="color: cyan; text-align: center;">Home Page</h1>
<a href="pdf_report"><h1 style="color: red; text-align: center;">Student
Report as PDF Doc</h1></a>
<br>
<a href="excel_report"><h1 style="color: red; text-align: center;">Student
Report as Excel Doc</h1></a>
```

StudentDTO.java

```
package com.nt.dto;

import java.io.Serializable;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class StudentDTO implements Serializable {
        private int sno;
        private String sname;
        private String address;
        private float avg;
}
```

StudentOperatorController.java

```
package com.nt.controller;

import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

import com.nt.dto.StudentDTO;
import com.nt.service.IStudentMgmtService;

@Controller
```

```
public class StudentOperatorController {

        @Autowired
        private IStudentMgmtService service;

        @GetMapping("/home")
        public String showHome() {
                return "welcome";
        }

        @GetMapping("/pdf_report")
        public String showPdfReport(Map<String, Object> map) {
                //use service
                List<StudentDTO> listDTO = service.getAllStudent();
                //Keep in mode attribute
                map.put("studentsInfo", listDTO);
                return "pdfView";
        }

        @GetMapping("/excel_report")
        public String showExcelReport(Map<String, Object> map) {
                //use service
                List<StudentDTO> listDTO = service.getAllStudent();
                //Keep in mode attribute
                map.put("studentsInfo", listDTO);
                return "excelView";
        }

}
```

IStudentMgmtService.java

```
package com.nt.service;

import java.util.List;

import com.nt.dto.StudentDTO;

public interface IStudentMgmtService {
        public List<StudentDTO> getAllStudent();
}
```

StudentMgmtServiceImpl.java

```
package com.nt.service;

import java.util.List;

import org.springframework.stereotype.Service;

import com.nt.dto.StudentDTO;

@Service("stuService")
public class StudentMgmtServiceImpl implements IStudentMgmtService {

        public List<StudentDTO> getAllStudent() {
                return List.of(new StudentDTO(101, "Raja", "Hyd", 67.8f),
                                new StudentDTO(102, "Ramesh", "MP", 77.8f),
                                new StudentDTO(103, "Lokesh", "Delhi", 68.8f),
                                new StudentDTO(104, "Hari", "Vizg", 87.8f));
        }

}
```

StudentExcelView.java

```
package com.nt.view;

import java.util.List;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.view.document.AbstractXlsView;

import com.lowagie.text.Paragraph;
import com.nt.dto.StudentDTO;

@Component("excelView")
public class StudentExcelView extends AbstractXlsView {


        @Override
        protected void buildExcelDocument(Map<String, Object> model,
Workbook workbook, HttpServletRequest request,
                HttpServletResponse response) throws Exception {
        List<StudentDTO> listDTO = null;
        Paragraph para = null;
        // get model attribute data
        listDTO = (List<StudentDTO>) model.get("studentsInfo");
        Sheet sheet = workbook.createSheet("User List");

        Row header = sheet.createRow(0);
        header.createCell(0).setCellValue("ID");
        header.createCell(1).setCellValue("Name");
        header.createCell(2).setCellValue("Address");
        header.createCell(3).setCellValue("Avreage");

        int rowNum = 1;

        for (StudentDTO dto : listDTO) {
            Row row = sheet.createRow(rowNum++);
            row.createCell(0).setCellValue(dto.getSno());
            row.createCell(1).setCellValue(dto.getSname());
            row.createCell(2).setCellValue(dto.getAddress());
            row.createCell(3).setCellValue(dto.getAvg());
        }
    }

}
```

StudentPdflView.java

```
package com.nt.view;

import java.awt.Color;
import java.util.List;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.view.document.AbstractPdfView;

import com.lowagie.text.Document;
import com.lowagie.text.Font;
import com.lowagie.text.Paragraph;
import com.lowagie.text.Table;
import com.lowagie.text.pdf.PdfWriter;
import com.nt.dto.StudentDTO;

@Component("pdfView")
public class StudentPdfView extends AbstractPdfView {

        @Override
        protected void buildPdfDocument(Map<String, Object> model,
Document document, PdfWriter writer,
                        HttpServletRequest request, HttpServletResponse
response) throws Exception {
                List<StudentDTO> listDTO = null;
                Paragraph para = null;
                // get model attribute data
                listDTO = (List<StudentDTO>) model.get("studentsInfo");
                //create and add paragraph
                para = new Paragraph("Student Details Report", new
Font(Font.HELVETICA, 24, Font.BOLDITALIC, Color.CYAN));
                document.add(para);

                Table table = new Table(4);
                table.addCell("ID");
                table.addCell("Name");
                table.addCell("Address");
                table.addCell("Average");

                for (StudentDTO dto : listDTO) {
                        table.addCell(String.valueOf(dto.getSno()));
                        table.addCell(dto.getSname());
                        table.addCell(dto.getAddress());
                        table.addCell(String.valueOf(dto.getAvg()));
                }
                document.add(table);
        }

}
```

## 100% code Driven Spring MVC

- Avoid xml configuration completely.
- web.xml file alternate is WebApplicationInitializer class.
- dispatcher-servlet.xml file alternate is WebMVCAppConfig class (@Configuration class).
- applicationContet.xml file alternate is RootAppConfig class

(@Configuration class)

Basic Thumb rule for 100% Code Driven Apps:

- Configure user-defined classes as spring beans using stereo type annotations and link them with @Configuration class using @ComponentScan annotation.
- Configure pre-defined classes as spring beans using @Bean methods of @Confiugration class.

WebMVCAppConfig.java

```
package com.nt.config;
@Configuration
@ComponentScan (basePackages="....")
@EnableWebMVC
public class WebMVCAppConfig {
        ………. more other beans using @Bean methods.
        ……… for ViewResolver, TilesConfigurer and etc.
}
```

RooAppConfig.java

```
@Configuration
@ComponentScan (basePackages= "……")    (or)
@Import ({PersistenConfig.class, ServiceConfig.class, ...}
public class RootAppConfig {
        ….. @Bean methods
        …… for DS, JdbcTemplate and etc.
}
```

Note: @EnableWebMVC Automatically registers multiple pre-defined classes with DS managed IoC container they are,

This class registers the following HandlerMappings:

- RequestMappingHandlerMapping ordered at 0 for mapping requests to annotated controller methods.
- HandlerMapping ordered at 1 to map URL paths directly to view names.
- BeanNameUrlHandlerMapping ordered at 2 to map URL paths to controller bean names.
- HandlerMapping ordered at Integer.MAX_VALUE-1 to serve static resource requests.

- HandlerMapping ordered at Integer.MAX_VALUE to forward requests to the default servlet.

Registers these HandlerAdapters:

- RequestMappingHandlerAdapter for processing requests with annotated controller methods.
- HttpRequestHandlerAdapter for processing requests with HttpRequestHandlers.
- SimpleControllerHandlerAdapter for processing requests with interface-based Controllers.

Registers a HandlerExceptionResolverComposite with this chain of exception resolvers:

- ExceptionHandlerExceptionResolver for handling exceptions through ExceptionHandler methods.
- ResponseStatusExceptionResolver for exceptions annotated with ResponseStatus.
- DefaultHandlerExceptionResolver for resolving known Spring exception types

Registers an AntPathMatcher and a UrlPathHelper to be used by:

- the RequestMappingHandlerMapping,
- the HandlerMapping for ViewControllers
- and the HandlerMapping for serving resources

Note that those beans can be configured with a PathMatchConfigurer.

Both the RequestMappingHandlerAdapter and the ExceptionHandlerExceptionResolver are configured with default instances of the following by default:

- a ContentNegotiationManager
- a DefaultFormattingConversionService
- an OptionalValidatorFactoryBean if a JSR-303 implementation is available on the classpath
- a range of HttpMessageConverters depending on the third-party libraries available on the classpath.

3 ways of configuring Servlet component:

a. Declarative approach (<servlet>, <servlet-mapping> tags) [for Predefined Servlet components where xml configuration is allowed].
b. Annotation [@WebServlet] approach (for user-defined servlet components).
c. Programmatic Approach/ Java code or config Approach/ Dynamic Registration approach [Using sc.addServlet(-,-) method] [For pre-defined Servlet components where xml configured are not allowed].

Note: In 100%Code, Spring Boot mode of Applications development there is no possibility of working with web.xml file So we need to use Programmatic Approach to configuration/ register predefined DispatcherServlet component class.

WebApplicationInitializer class to configure DS, ContextLoaderListener in spring MVC App as alternate to web.xml:

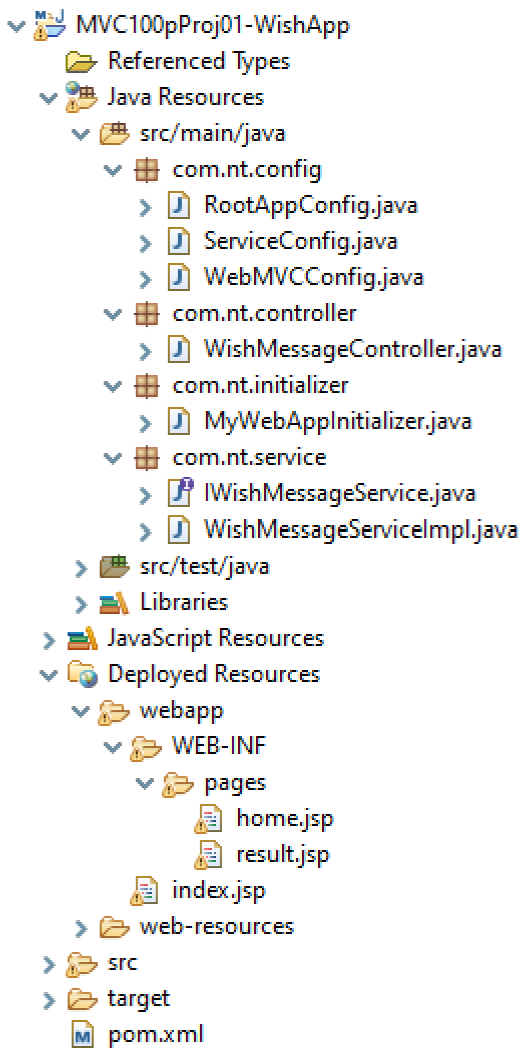
(a) Deployment of Spring MVC application
(b) Identifies no web.xml file and also identifies no @WebServlet servlet components
(c) Creates One ServletContext object

MyWebAppInitializer.java
```
package com.nt.initializer;
public class MyWebAppInitializer implements org.sf.web.
                    (h)                   WebApplicationInitializer {
        public void onStartup (ServletContext sc) throws ServletException {
                //create parent IoC Container
                AnnotationConfigWebApplicationContext parentCtx=
                        new AnnotationConfigWebApplicationContext ();
                parentCtx.register(RootAppConfig.class); (i) pre-instantiation of
                                                           singleton scope beans
                //create ContextLoaderListener object having Parent IoC
                //container
                ContextLoaderListener listener = new
                                        ContextLoaderListener(parentCtx);
                //register ContextLoaderListener with ServletContainer
                sc.addListener(listener); (j)
                //create child Container
                AnnotationConfigWebApplicationContext childCtx=
                        new AnnotationConfigWebApplicationContext ();
                childCtx.register(WebMvcAppConfig.class); (k) pre-instantiation of
                                        singleton scope beans in presentation tier
                //Create DispatcherServlet object having Child IOC container
                DispatcherServlet servlet=new DispatcherServlet(childCtx);
                //register DispatcherServlet with ServletContainer
                ServletRegistration.Dynamic
                            dyna=sc.addServlet("dispatcher", servlet); (l)
                dyna.addMapping("/");
                dyna.setLoadOnStartup(2);
        } //method
}//class
```

```
WEB-INF/lib folder of Spring MVC App
        |--> *.jar (spring-web-<ver>.jar and etc.)
                |
                |       (d) ServletContainer looks for META-INF/services/
                |       javax.servlet.ServletContainerInitializer file in all the jar
                |       files that are added to WEB-INF/lib folder and finds it
                |       spring-web-<ver>.jar file
                |
        javax.servlet.ServletContainerInitializer (file) (META-INF/services)
                org.springframework.web.SpringServletContainerInitializer
                        (e) ServletContainer loads this, creates object
                        and calls onStartup(-,-) having ServletContext object

                                    (f)
                        p v onStartup (Set<…> set, ServletContext sc) {
                             …….. calls onStartup(sc) on our
                            (g) WebApplicationInitializer class by searching
                             it in CLASSPATH (WEB-INF/classes)
                        }
```

Note: SpringServletContainerInitializer class Delegates the ServletContext object to any WebApplicationInitializer (I) implementations present on the application CLASSPATH, i.e. the onStartup(-,-) of SpringServletContainerInitializer class internally calls onStartup(-) our WebApplicationInitializer (I) implementation class onStartup(-) method.

## Directory Structure of MVC100p01-WishApp:



```
MVC100pProj01-WishApp
    Referenced Types
    Java Resources
        src/main/java
            com.nt.config
                RootAppConfig.java
                ServiceConfig.java
                WebMVCConfig.java
            com.nt.controller
                WishMessageController.java
            com.nt.initializer
                MyWebAppInitializer.java
            com.nt.service
                IWishMessageService.java
                WishMessageServiceImpl.java
        src/test/java
        Libraries
    JavaScript Resources
    Deployed Resources
        webapp
            WEB-INF
                pages
                    home.jsp
                    result.jsp
            index.jsp
        web-resources
    src
    target
    pom.xml
```

- Develop the above directory structure and folder, package, class, JSP, file after convert to web application.
- Copy the pom.xml, index.jsp from the other project.
- Add the following code with their respective files.

### home.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<h1 style="color: red; text-align: center">Welcome to Wish App</h1><br>
<h2 style="color: cyan; text-align: left">
        <a href="wish">Click here to get te wish message</a>
</h2>
```

result.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1" isELIgnored="false"%>
<h1 style="color: red; text-align: center">Welcome to Wish App</h1><br>
<h2 style="color: cyan; text-align: center">
        ${wishMessage}
</h2><br>
<a href="welcome.htm">Home</a>
```

IWishMessageService.java

```
package com.nt.service;

public interface IWishMessageService {
        public String getWishMessage();
}
```

WishMessageService.java

```
package com.nt.service;
import java.time.LocalDateTime;
import org.springframework.stereotype.Service;
@Service("webService")
public class WishMessageServiceImpl implements IWishMessageService {
        @Override
        public String getWishMessage() {
                LocalDateTime ldt = null;
                int hours = 0;
                //get Calendar class obejct
                ldt = LocalDateTime.now();
                //get hour of the day
                hours = ldt.getHour();
                if (hours<12)
                        return "Good Morning";
                else if (hours<16)
                        return "Good Afternoon";
                else if (hours<20)
                        return "Good Evening";
                else
                        return "Good Night";
        }
}
```

MyWebAppInitializer.java

```
package com.nt.initializer;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration;
import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.ContextLoaderListener;
import
org.springframework.web.context.support.AnnotationConfigWebApplicatio
nContext;
import org.springframework.web.servlet.DispatcherServlet;
import com.nt.config.RootAppConfig;
import com.nt.config.WebMVCConfig;

public class MyWebAppInitializer implements WebApplicationInitializer {

	@Override
	public void onStartup(ServletContext sc) throws ServletException {
		// create Parent IoC container
		AnnotationConfigWebApplicationContext parentCtx = new
AnnotationConfigWebApplicationContext();
		parentCtx.register(RootAppConfig.class);
		//create contextLoaderListener
		ContextLoaderListener listener = new
ContextLoaderListener(parentCtx);
		//register listener with container
		sc.addListener(listener);
		//create child IoC container
		AnnotationConfigWebApplicationContext childCtx = new
AnnotationConfigWebApplicationContext();
		childCtx.register(WebMVCConfig.class);
		//create DispatcherServlet class object
		DispatcherServlet servlet = new DispatcherServlet(childCtx);
		//register DS with servlet container
ServletRegistration.Dynamic registration = sc.addServlet("ds", servlet);
		//provide url pattern
		registration.addMapping("/");
		registration.setLoadOnStartup(2);
	}
}
```

WishMessageController.java

```
package com.nt.controller;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

import com.nt.service.IWishMessageService;

@Controller
public class WishMessageController {

	@Autowired
	private IWishMessageService service;

	@GetMapping("/welcome")
	public String showHome() {
		return "home";
	}

	@GetMapping("/wish")
	public String showWishMessage(Map<String, Object> map) {
		//user Servicer
		String result = service.getWishMessage();
		map.put("wishMessage", result);
		return "result";
	}

}
```

RootConfig.java

```
package com.nt.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;

@Configuration
@Import(ServiceConfig.class)
public class RootAppConfig {

}
```

ServiceConfig.java

```
package com.nt.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.nt.service")
public class ServiceConfig {

}
```

WebMVCConfig.java

```
package com.nt.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import
org.springframework.web.servlet.config.annotation.EnableWebMvc;
import
org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.nt.controller")
public class WebMVCConfig {

	//ViewResolver configuration
	@Bean
	public ViewResolver CreaterIRVR() {
		InternalResourceViewResolver irvr = new
InternalResourceViewResolver();
		irvr.setPrefix("/WEB-INF/pages/");
		irvr.setSuffix(".jsp");
		return irvr;
	}

}
```

Note:

- ✓ AbstractAnnotationConfigDispatcherServletInitializer is pre-defined WebApplicationInitializer to register a DispatcherServlet and use Java-based Spring configuration. Implementations are required to implement:
  getRootConfigClasses() -> for "root" application context (non-web infrastructure) configuration.
  getServletConfigClasses() -> for DispatcherServlet application context (Spring MVC infrastructure) configuration.
  getServletMappings() -> To provide URL pattern to DS configuration.

- ✓ The onStartup (-) of AbstractAnnotationConfigDispatcherServletInitializer internally takes care of creating both parent, Child IoC containers and ContextLoaderListener and DS registrations having IoC container and etc. This method internally calls the above 3 methods to get inputs like RootAppConfig, WebMVCAppConfig configuration classes and DS URL pattern.

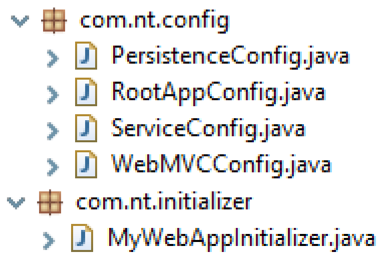
MyWebAppInitializer.java

```
package com.nt.initializer;

import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatc
herServletInitializer;

import com.nt.config.RootAppConfig;
import com.nt.config.WebMVCConfig;

public class MyWebAppInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

	@Override
	public Class<?>[] getRootConfigClasses() {
		//will be used by super class onStartup(-) to get Configuration
class name  for Parent IoC container
		return new Class[] {RootAppConfig.class};
	}

	@Override
	protected Class<?>[] getServletConfigClasses() {
		//will be used by super class onStartup(-) to get Configuration
class name  for child IoC container
```

```
                    return new Class[] {WebMVCConfig.class};
        }

        @Override
        protected String[] getServletMappings() {
                //Will be used by super class onStartup(-) to get DC URL
pattern
                return new String[] {"/"};
        }

}
```

Directory Structure of MVC100pProj02-LoginApp:

- Copy paste the MVCAnnoProj05-LoginApp application.
- Delete all the xml files.
- Add the following package and class.
- for these classes copy the code from previous project.



```
com.nt.config
    PersistenceConfig.java
    RootAppConfig.java
    ServiceConfig.java
    WebMVCConfig.java
com.nt.initializer
    MyWebAppInitializer.java
```

- Add the following code in their respective files.
- Rest of code is same as the copied project and previous project

RootConfig.java

```
package com.nt.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;

@Configuration
@Import({ServiceConfig.class, PersistenceConfig.class})
public class RootAppConfig {

}
```

RootConfig.java

```
package com.nt.config;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.simple.SimpleJdbcCall;
import org.springframework.jndi.JndiObjectFactoryBean;

@Configuration
@ComponentScan(basePackages = "com.nt.dao")
public class PersistenceConfig {

	@Bean
	public JndiObjectFactoryBean createJOFB() {
		JndiObjectFactoryBean jofb = new JndiObjectFactoryBean();
		jofb.setJndiName("java:/comp/env/DsJndi");
		return jofb;
	}

	@Bean
	public SimpleJdbcCall createSJC() {
		return new  SimpleJdbcCall((DataSource)
createJOFB().getObject());
	}

}
```

## Spring Boot MVC

- Gives embedded servers,
  - Apache Tomcat
  - Eclipse jetty
  - Undertow
- 100% Code based configurations (XML configurations are avoided).
- AutoConfiguration.
- Live Server Reloading (Here changes done in source will be recognized internally).
- Most of the Spring WEB MVC flow components will be generated internally through auto configuration.
- We can work both with Embedded Servers and external servers.

- We can develop web application either as standalone applications (jar file) (Embedded servers) and deployable web applications (war file) (Embedded servers/ External servers).
  and etc.

Thumb rule:

- Configure user-defined classes as Spring beans using stereo type annotations.
- Configure pre-defined classes as Spring beans using @Bean methods of @Configuration classes, if they are not coming through AutoConfiguration (we can give inputs to AutoConfiguration process using application.properties file)

Spring Boot MVC takes care of following thing internally:

a. DispatcherServlet configure with “/" URL pattern by giving one readymade web application Initializer class in the Deployment Directory structure.
b. RequestMappingHandlerMapping will come as handler mapping automatically.
c. So, many ready Servlet Filters to show web pages having error messages.
d. Embedded Server (default tomcat).
e. InternalResourceviewResolver by collecting the inputs from application.properties file (prefix, suffix).
   and etc.

Note:

- ✓ Controller classes, Service classes, DAO classes, AOP classes and etc. should be developed by the Programmers.
- ✓ If your system is not having external server installation then only use Embedded servers otherwise avoid them because Embedded servers do not support connection pooling, security, logging and etc. features.

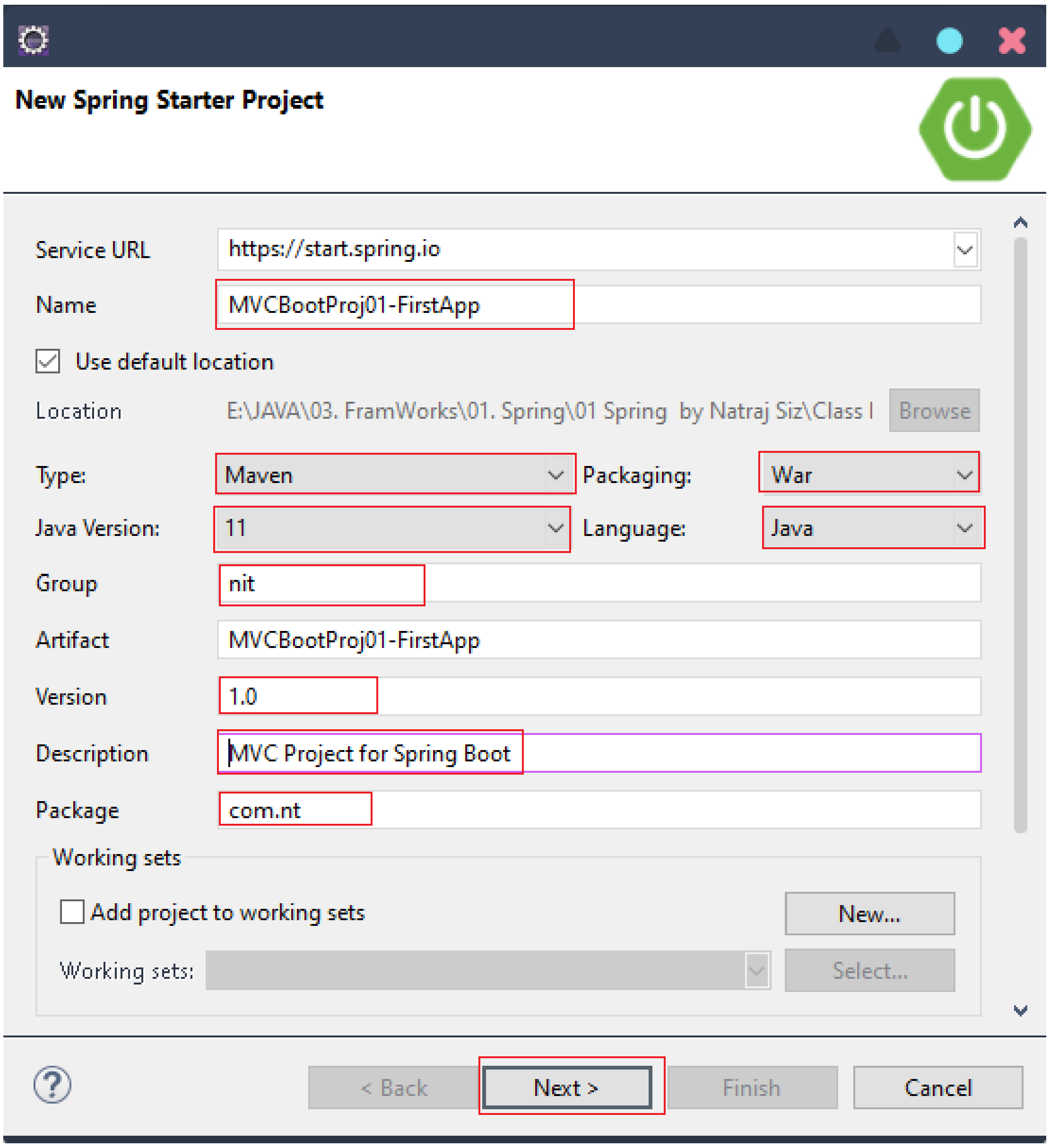
## Procedure to develop First Spring Boot MVC Application

Step 1: Create Spring Starter Project having the following details.

File --> new --> Spring Starter Project, and fill the following details like

| | |
|---|---|
| Type: Maven | Packaging: War |
| Java Version: 11 | Language: Java |

And give the project name and all other details like Package, then click on Next, Choose the spring-boot-starter-web and required stater then click on Next, Finish



New Spring Starter Project

Service URL: https://start.spring.io
Name: MVCBootProj01-FirstApp
☑ Use default location
Location: E:\JAVA\03. FramWorks\01. Spring\01 Spring by Natraj Siz\Class I
Type: Maven
Packaging: War
Java Version: 11
Language: Java
Group: nit
Artifact: MVCBootProj01-FirstApp
Version: 1.0
Description: MVC Project for Spring Boot
Package: com.nt
Working sets
☐ Add project to working sets
Working sets:

Step 2: Add tomcat-jasper jar file to pom.xml.

```
        <!--
https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-
embed-jasper -->
        <dependency>
                <groupId>org.apache.tomcat.embed</groupId>
                <artifactId>tomcat-embed-jasper</artifactId>
                <version>9.0.39</version>
        </dependency>
```

Note: This is not required if you are running the application on External Tomcat server. Required only while working with Embedded Tomcat server.

Step 3: Understanding Generated Directory structure of Spring Boot MVC Application.

ServletInitializer.java

```
package com.nt;

import org.springframework.boot.builder.SpringApplicationBuilder;
import
org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

	@Override
	protected SpringApplicationBuilder
configure(SpringApplicationBuilder application) {
		return
application.sources(MvcBootProj01FirstAppApplication.class);
	}

}
```

Note:

- ✓ In Spring Boot MVC, we can work only one IoC container i.e. there is no provision to work with two IOC container as we did in other approaches.
- ✓ In Spring Boot MVC Applications, there is no provision to write Business tier or presentation tier comps outside the Spring environment. So, two IoC containers provision is not given and not required.

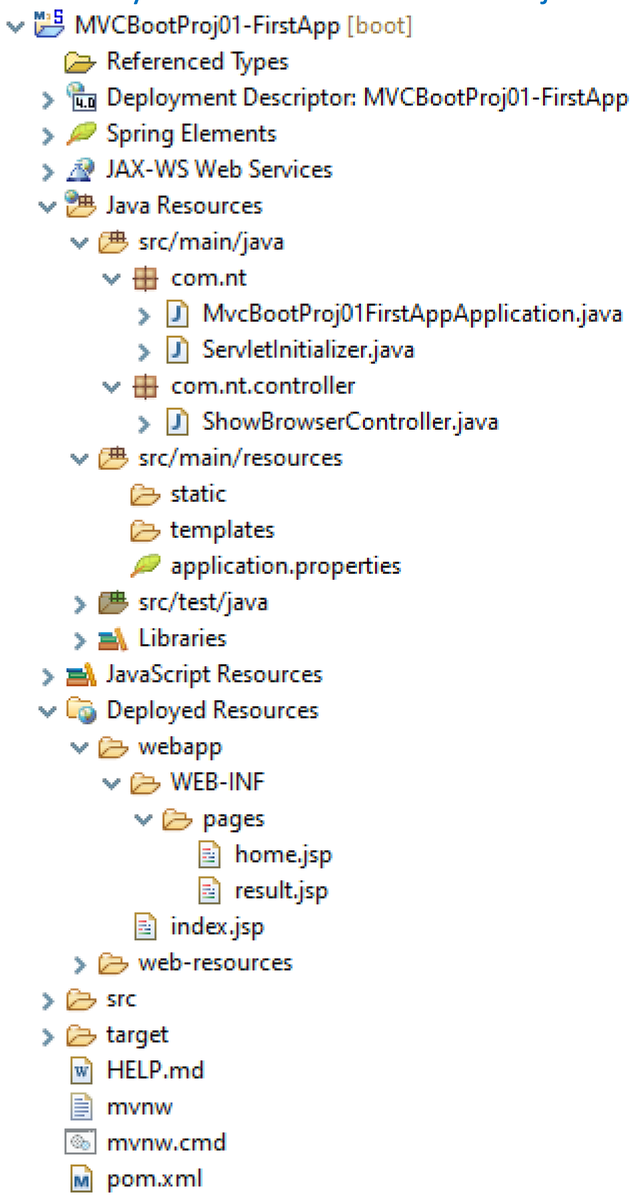
Step 4: Develop the source code According your requirement by keeping AutoConfiguration in mind.

Step 5: Run the web Application on external server.
In pom.xml --> comment tomcat-embedded-jasper dependency tag (because that is only for embedded Tomcat server).
Right click on the Project --> Run on Server --> choose Tomcat Server

Step 6: Test the web application
http://10ca1host:3030/MVCProj22-BootMVC-FirstApp

Note: If you are running Spring boot MVC Application on external server there is no need of placing main (-) method starter/main/Configuration class where @SpringBootApplicaton is coming.

## Directory Structure of MVCBootProj01-FirstApp:



```
MVCBootProj01-FirstApp [boot]
    Referenced Types
  > Deployment Descriptor: MVCBootProj01-FirstApp
  > Spring Elements
  > JAX-WS Web Services
  v Java Resources
    v src/main/java
      v com.nt
        > MvcBootProj01FirstAppApplication.java
        > ServletInitializer.java
      v com.nt.controller
        > ShowBrowserController.java
    v src/main/resources
        static
        templates
        application.properties
    > src/test/java
    > Libraries
  > JavaScript Resources
  v Deployed Resources
    v webapp
      v WEB-INF
        v pages
            home.jsp
            result.jsp
          index.jsp
    > web-resources
  > src
  > target
    HELP.md
    mvnw
    mvnw.cmd
    pom.xml
```

- Develop the above directory structure using Spring Starter Project option and package and classes also.
- Many jars dependencies will come automatically in pom.xml, and as per our required we can add also in <dependencies> tag.
- Then write the following code with in their respective file.

pom.xml

```xml
		<dependency>
			<groupId>org.apache.tomcat.embed</groupId>
			<artifactId>tomcat-embed-jasper</artifactId>
			<version>9.0.39</version>
		</dependency>
```

home.jsp

```
<h1 style="color: blue; text-align: center;">
        <a href="browser">Get Browser Details</a>
</h1>
```

result.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1" isELIgnored="false"%>

<b><i>Current browser name: ${browser}</i></b>
<br>
<a href="welcome">home</a>
```

application.properties

```
#View Resolver configuration
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

ShowBrowserController.java

```
package com.nt.controller;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class ShowBrowserController {

        @GetMapping("/welcome")
        public String showHomePage() {
                return "home";
        }

        @GetMapping("/browser")
        public String getBrowserName(Map<String, Object> map,
HttpServletRequest req) {
                //get Browser name
                String brName = req.getHeader("user-agent");
                //keep browser name as model attribute
                map.put("browser", brName);
                return "result";
        }

}
```

## Running Spring Boot MVC Application on Embedded Tomcat server:

a. Make sure tomcat-embedded-jasper jar/dependency is added to pom.xml file.
b. Make sure that main (-) is placed in main Configuration/ starter class where @SpringBootApplication is coming.
c. Change the embedded tomcat server port number using application.properties file.

application.properties

```
#Embedded tomcat server port number
server.port=4040
```

d. Run the Application as spring Boot App/ Java Application
by using main class where @SpringBootApplication is placed
(Here application runs as standalone App using embedded tomcate).

e. Test the web application
http://localhost:4040/welcome
(Type this URL by opening browser window)

Note:

- ✓ Default welcome file will not work here (Embedded Tomcat)
- ✓ ServletInitializer.java file is not required while running the Application as standalone App using embedded Tomcat.

- To make Spring boot MVC app working with Server managed JDBC connection pool add the following entry.

application.properties

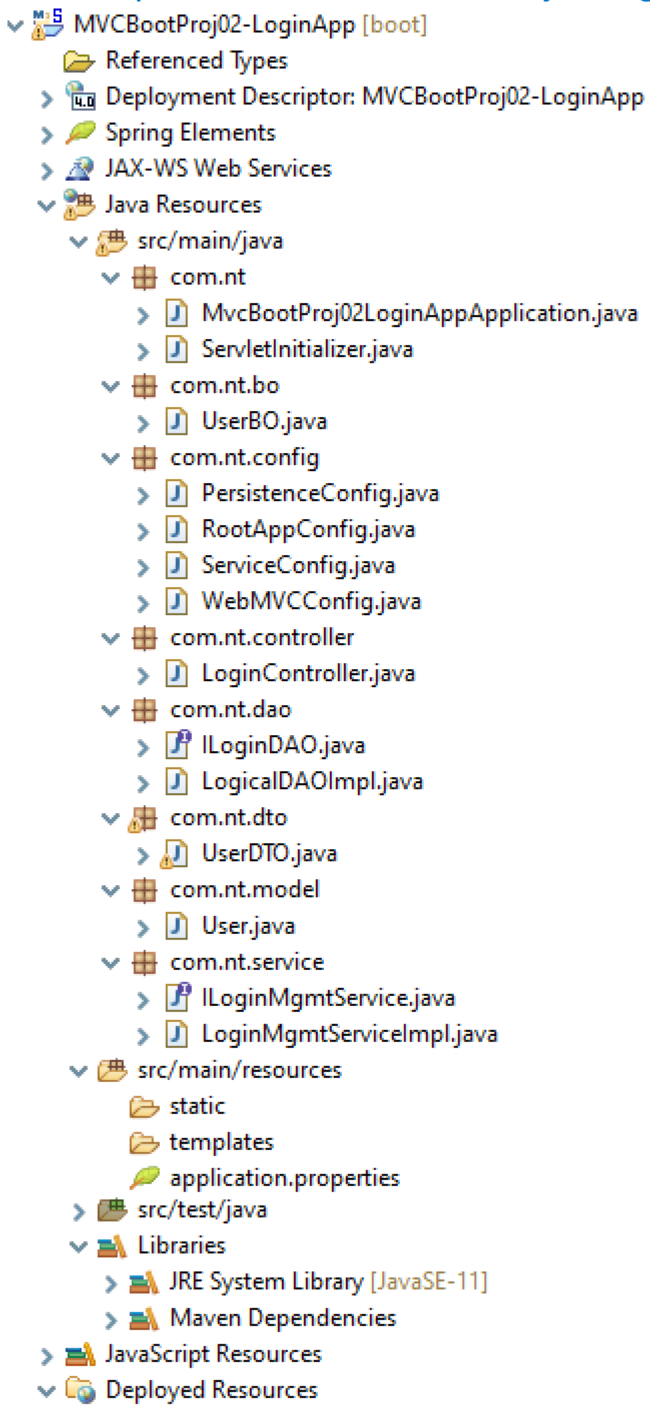
```
#To work server managed JDBC connection pool
spring.datasource.jndi-name=java:/comp/env/DsJndi
```

- Spring boot MVC gives InternalResourceviewResolver as the default ViewResolver to supply inputs that view resolver like prefix and suffix, we need to write following entries in application.properties.

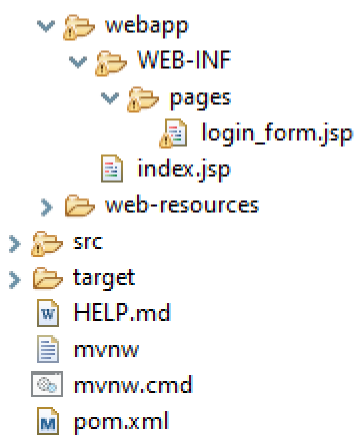
application.properties

```
#View Resolver configuration
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

## Directory Structure of MVCBootProj02-LoginApp:



```
MVCBootProj02-LoginApp [boot]
    Referenced Types
    Deployment Descriptor: MVCBootProj02-LoginApp
    Spring Elements
    JAX-WS Web Services
    Java Resources
        src/main/java
            com.nt
                MvcBootProj02LoginAppApplication.java
                ServletInitializer.java
            com.nt.bo
                UserBO.java
            com.nt.config
                PersistenceConfig.java
                RootAppConfig.java
                ServiceConfig.java
                WebMVCConfig.java
            com.nt.controller
                LoginController.java
            com.nt.dao
                ILoginDAO.java
                LogicalDAOImpl.java
            com.nt.dto
                UserDTO.java
            com.nt.model
                User.java
            com.nt.service
                ILoginMgmtService.java
                LoginMgmtServiceImpl.java
        src/main/resources
            static
            templates
            application.properties
        src/test/java
        Libraries
            JRE System Library [JavaSE-11]
            Maven Dependencies
    JavaScript Resources
    Deployed Resources
```


```
webapp
  WEB-INF
    pages
      login_form.jsp
    index.jsp
  web-resources
src
target
HELP.md
mvnw
mvnw.cmd
pom.xml
```

- Develop the above directory structure using Spring Starter Project option and package and classes also.
- During this choose the following starter dependencies
  - Lombok
  - JDBC API
  - Oracle Driver
  - Spring Web
- Many jars dependencies will come automatically in pom.xml, and as per our required we can add also in <dependencies> tag here we have to add JSTL jar.
- Then write the following code with in their respective file.
- Copy the rest of code from MVC100pProj02-LoginApp application.

PersistenceConfig.java

```java
package com.nt.config;

import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.simple.SimpleJdbcCall;
@Configuration
public class PersistenceConfig {
	@Autowired
	private DataSource ds;
	@Bean
	public SimpleJdbcCall createSJC() {
		return new  SimpleJdbcCall(ds);
	}
}
```

application.properties

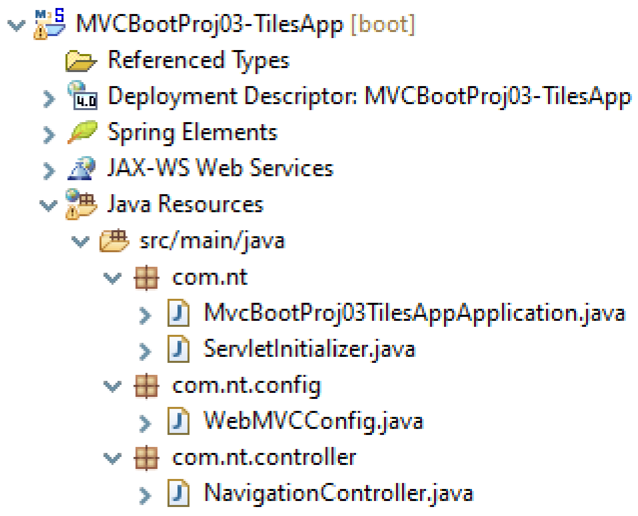
```
#To work server managed JDBC connection pool
spring.datasource.jndi-name=java:/comp/env/DsJndi

#View Resolver configuration
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

- To view other than InternalResourceviewResolver in Spring boot MVC Application, we need to configuration that ViewResolver as @Bean method in any configuration class

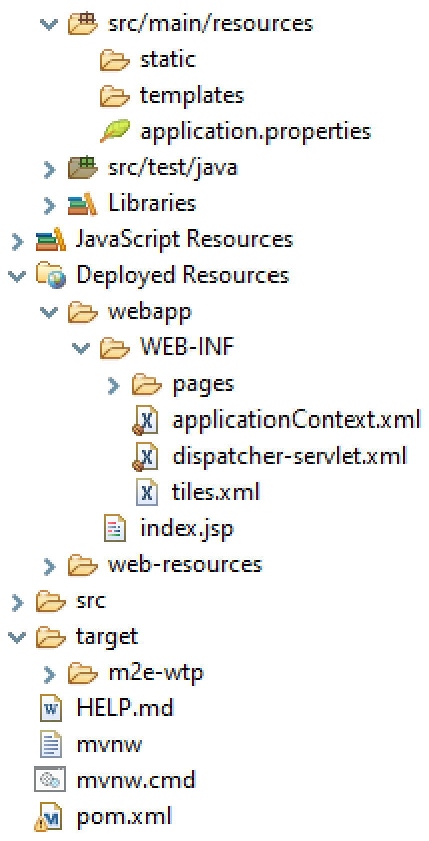
```
@Bean
public TilesViewResolver createVR() {
        return new TilesViewResolver ();
}
```

- In Spring Boot MVC the following special classes should be configured as Spring beans using @Bean methods because they will not come through auto configuration
  a. Resolvers like CommonsMultipartResolver (In file uploading), SessionLocaleResolver (I18n)
  b. Configures like TilesConfigurer
  c. Other than InternalResourceviewResolver like TilesViewResolver, BeanNameViewResolver, ResourceBundleViewResolver and etc. and etc.

## Directory Structure of MVCBootProj03-TilesApp:



```
MVCBootProj03-TilesApp [boot]
  Referenced Types
  Deployment Descriptor: MVCBootProj03-TilesApp
  Spring Elements
  JAX-WS Web Services
  Java Resources
    src/main/java
      com.nt
        MvcBootProj03TilesAppApplication.java
        ServletInitializer.java
      com.nt.config
        WebMVCConfig.java
      com.nt.controller
        NavigationController.java
```



```
src/main/resources
    static
    templates
    application.properties
src/test/java
Libraries
JavaScript Resources
Deployed Resources
  webapp
    WEB-INF
      pages
      applicationContext.xml
      dispatcher-servlet.xml
      tiles.xml
    index.jsp
  web-resources
src
target
  m2e-wtp
HELP.md
mvnw
mvnw.cmd
pom.xml
```

- Develop the above directory structure using Spring Starter Project option and package and classes also.
- During this choose the following starter dependencies
  - Spring Web
- Many jars dependencies will come automatically in pom.xml, and as per our required we can add also in <dependencies> tag here we have to add JSTL jar and tiles jar.
- Then write the following code with in their respective file.
- Copy the rest of code from MVCAnnoProj10-Tiles application.

WebMVCConfig.java

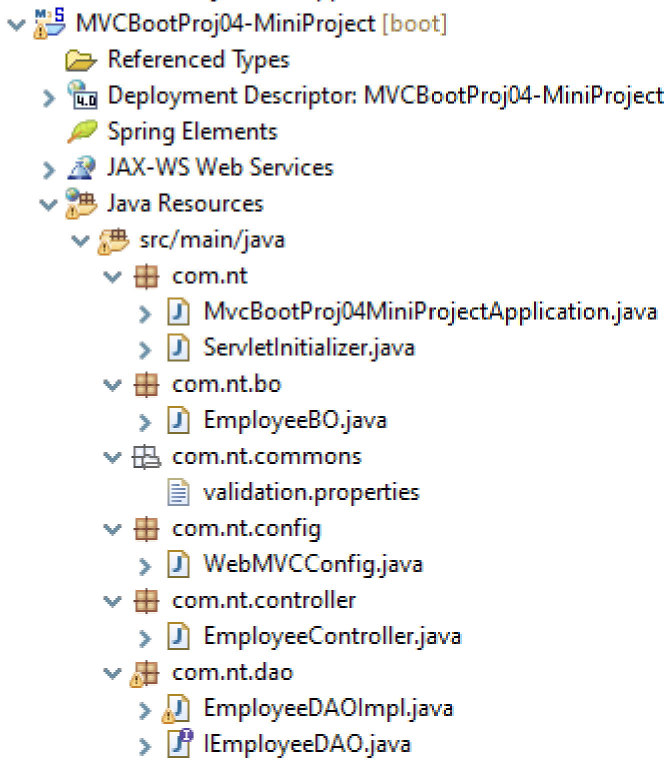
```java
package com.nt.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.tiles3.TilesConfigurer;
import org.springframework.web.servlet.view.tiles3.TilesViewResolver;

@Configuration
public class WebMVCConfig {
        @Bean
        public TilesConfigurer createrTilesConfigurer() {
                TilesConfigurer configurer = new TilesConfigurer();
                configurer.setDefinitions("/WEB-INF/tiles.xml");
                return configurer;
        }

        @Bean
        public TilesViewResolver createVR() {
                return new TilesViewResolver();
        }

}
```

application.properties

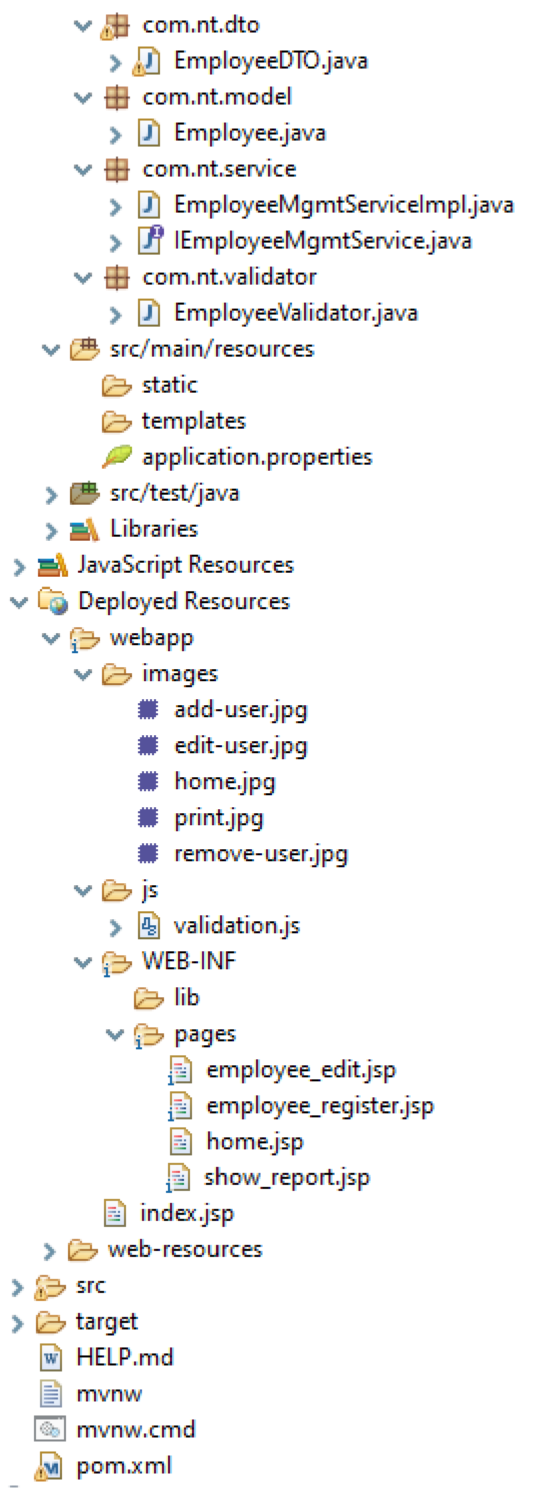
```
#View Resolver configuration
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

Note: The additional ViewResolver that are added as @Bean Methods will always added before the AutoConfiguration based InternalResourceviewResolver to support View Resolver Chaining.

Directory Structure of MVCBootProj04-MiniProject:



```
MVCBootProj04-MiniProject [boot]
    Referenced Types
    Deployment Descriptor: MVCBootProj04-MiniProject
    Spring Elements
    JAX-WS Web Services
    Java Resources
        src/main/java
            com.nt
                MvcBootProj04MiniProjectApplication.java
                ServletInitializer.java
            com.nt.bo
                EmployeeBO.java
            com.nt.commons
                validation.properties
            com.nt.config
                WebMVCConfig.java
            com.nt.controller
                EmployeeController.java
            com.nt.dao
                EmployeeDAOImpl.java
                IEmployeeDAO.java
```



```
        com.nt.dto
          EmployeeDTO.java
        com.nt.model
          Employee.java
        com.nt.service
          EmployeeMgmtServiceImpl.java
          IEmployeeMgmtService.java
        com.nt.validator
          EmployeeValidator.java
    src/main/resources
        static
        templates
        application.properties
    src/test/java
    Libraries
  JavaScript Resources
  Deployed Resources
    webapp
        images
          add-user.jpg
          edit-user.jpg
          home.jpg
          print.jpg
          remove-user.jpg
        js
          validation.js
        WEB-INF
          lib
          pages
            employee_edit.jsp
            employee_register.jsp
            home.jsp
            show_report.jsp
        index.jsp
    web-resources
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

- Develop the above directory structure using Spring Starter Project option and package and classes also.
- During this choose the following starter dependencies
    - Lombok
    - JDBC API
    - Oracle Driver
    - Spring Web

- Many jars dependencies will come automatically in pom.xml, and as per our required we can add also in <dependencies> tag here we have to add JSTL jar.
- Then write the following code with in their respective file.
- Copy the rest of code from MVCAnnoProj06-LayeredApp-CURD-MiniProject application.

application.properties

```
#To work server managed JDBC connection pool
spring.datasource.jndi-name=java:/comp/env/DsJndi

#View Resolver configuration
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

WebMVCConfig.java

```
package com.nt.config;

import org.springframework.context.annotation.Configuration;
import
org.springframework.context.support.ResourceBundleMessageSource;

@Configuration
public class WebMVCConfig {

	public ResourceBundleMessageSource createRBMS() {
		ResourceBundleMessageSource source = new
ResourceBundleMessageSource();
		source.setBasename("com/nt/commons/validation");
		return source;
	}

}
```

Note: In Spring boot starter project we can add new starters (in eclipse IDE) by using right click on project --> spring --> add starters -->

(or)

right click in pom.xml --> spring add starters -->

(or)

ctrl + space in pom.xml --> add starter -->

(or) collect info from mvnrepository.com and paste in pom.xml file.

## Directory Structure of MVCBootProj05-MiniProjectDataJPA:

- Copy past the MVCBootProj04-MiniProject and the name from Web Project Setting to MVCBootProj05-MiniProject-DataJPA.
- Delete the DAO package and create a package called com.nt.repository with a java class EmployeeRepository.java
- Add Spring Data JPA starter.
- Then write the following code with in their respective file.

application.properties

```
#To work server managed JDBC connection pool
spring.datasource.jndi-name=java:/comp/env/DsJndi

#View Resolver configuration
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp

#Spring data JPA properties
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDial
ect
```

EmployeeRepository.java

```
package com.nt.repository;

import java.util.List;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;

import com.nt.bo.EmployeeBO;

public interface EmployeeRepository extends
CrudRepository<EmployeeBO, Integer> {
	@Query("SELECT DISTINCT deptNo FROM EmployeeBO  WHERE
deptNo IS NOT NULL")
	public List<Integer> getAllDeptNos();

}
```

EmployeeMgmtServiceImpl.java

```
@Service("empService")
public class EmployeeMgmtServiceImpl implements
IEmployeeMgmtService {

	@Autowired
	private EmployeeRepository empRepo;

	@Override
	@Transactional(propagation = Propagation.REQUIRED, readOnly =
true)
	public List<EmployeeDTO> fetchAllEmployees() {
		List<EmployeeBO> listBO = null;
		List<EmployeeDTO> listDTO = new ArrayList<>();
		//use Repo
		listBO = (List<EmployeeBO>) empRepo.findAll();
		//convert listBO to listDTO
		listBO.forEach(bo->{
			EmployeeDTO dto = new EmployeeDTO();
			BeanUtils.copyProperties(bo, dto);
			dto.setSerialNo(listDTO.size()+1);
			dto.setGrossSalary(dto.getSal()+dto.getSal()*0.3f);

	dto.setNetSalary(dto.getGrossSalary()+dto.getGrossSalary()*0.1f);
			dto.setSal(Math.round(dto.getSal()));
			listDTO.add(dto);
		});

		return listDTO;
	}

	@Override
	@Transactional(propagation = Propagation.REQUIRED, readOnly =
false)
	public String registerEmployee(EmployeeDTO dto) {
		EmployeeBO bo = null;
		//convert DTO to bo
		bo = new EmployeeBO();
		BeanUtils.copyProperties(dto, bo);
		//use repo
		bo = empRepo.save(bo);
```

```
                return bo!=null?"Employee Registered":"Employee not
Registered";
        }

        @Override
        @Transactional(propagation = Propagation.REQUIRED, readOnly =
true)
        public List<Integer> fetchALLDeptNo() {
                // use repo
                return empRepo.getAllDeptNos();
        }

        @Override
        @Transactional(propagation = Propagation.REQUIRED, readOnly =
false)
        public void removeEmployeeById(int id) {
                int count = 0;
                //use Repo
                empRepo.deleteById(id);
        }

        @Override
        @Transactional(propagation = Propagation.REQUIRED, readOnly =
true)
        public EmployeeDTO fetchEmployeeById(int id) {
                EmployeeDTO dto = null;
                EmployeeBO bo = null;
                //use Repo
                bo = empRepo.findById(id).get();
                //convert bo to dto
                dto =  new EmployeeDTO();
                BeanUtils.copyProperties(bo, dto);
                return dto;
        }

        @Override
        @Transactional(propagation = Propagation.REQUIRED, readOnly =
false)
        public String ModifyEmployeeById(EmployeeDTO dto) {
                EmployeeBO bo = null;
                int count = 0;
                //convert DTO to BO
                bo = new EmployeeBO();
                BeanUtils.copyProperties(dto, bo);
                //use Repo
                bo = empRepo.save(bo);
                return bo==null?dto.getEmpNo()+" employee details are not
found to update":dto.getEmpNo()+" employee details are found to update";
        }

}
```

EmployeeController.java

```
        @GetMapping("/deleteEmp.htm")
        public String removeEmployee(RedirectAttributes redirect,
@RequestParam int eno) {
                // use service
                service.removeEmployeeById(eno);
                // add result to flash attribute
                redirect.addFlashAttribute("resultMsg", "Employee is
Deleted");
                return "redirect:list_emps.htm";
        }
```

EmployeeBO.java

```
@Data
@Entity
@Table(name = "EMP")

public class EmployeeBO {

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        @Column(name = "EMPNO")
        private Integer empNo;
        private String ename;
        private String job;
        private Float sal;
        @Column(name = "DEPTNO")
        private Integer deptNo;

}
```

## Thymeleaf

- Another UI Technology that is built on the top of html for dynamic web pages.
- Only HTML gives static web page. HTML tags + thymeleaf tags gives dynamic webpages. It is light weightier alternate to heavy weight JSP pages.
- In the execution of JSP page lot of memory and lot of CPU time is required because internally translated. In JSP equivalent servlet component and creates multiple implicit objects (9) if used or not used for all these things lot of memory and CPU time required.
- To overcome the above problems of jsp pages use thymeleaf as lightweight alternate for rendering dynamic webpages.
- We need to write thymeleaf tags in html tags by importing thymeleaf namespace by specifying its namespace URI.

```
<html xmlns:th="https://www.thymeleaf.org">
        .....
</html>
```

- Spring boot gives built-in support of thymeleaf UI by giving,
    - default prefix is <classpath>/templates/
        - (classpath here is src/main/resources folder).
    - default suffix is .html
- To use thymeleaf in Spring boot add this starter to pom.xml.

```
  <!--
https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-thymeleaf -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-thymeleaf</artifactId>
   <version>2.3.4.RELEASE</version>
</dependency>
```

- Standard jsp tags identified with fixed prefix called <jsp:xxx> and similarly thymeleaf tags are identified with <th:xxx> prefix.
- Popular symbols in thymeleaf programming,
    - @ --> To specify Location (/<global path>/<request path>)
    - $ --> To read data from container managed scopes like model attributes
    - * --> To bind/link data to form components (useful only in thymeleaf forms)

a. To read data from model attributes/ container managed scopes
    - for primitive/ wrapper/ String type model attributes th:text="${<attribute name>}".
    - for Object type model attributes th:text="${<ObjectName>}".
    - for getting property values from Object type model attributes th:text="${<ObjectName>.<property name>}".
    - for looping through arrays/collection th:each="<counter variable>:${<collection/array type attribute>}".

b. To display images (To link image file to <img> tag)
    - <img th:src="@{/path}">

c. To Link/map with hyperlink
    - <a th:href="@{/path}"> xxxx </a>

d. To Link/ map CSS file
    - <link rel="stylesheet" th:href="@{/path}">

e. To Link/ map java script file
    - <script type="text/javascript" th:src="@{/path}">……</script>

f. Thymeleaf forms are bi-directional forms i.e. they support DataBinding (writing form data to model class object) and Data Rendering (writing handler methods supplied model attributes/ model class obj data to form components)
    - To specify action URL <form th:action="@{global path/request path}" >
    - For binding model class object data/ model attributes data to form components (for form backing object operation)
        - <form th:object="${model attribute name/object name of model class}"> (alternate to <frm:form modelAttribute="....">)
    - For binding model class object data/ model attributes data to form components
        - <input type="text" th:field="*{<model class property name/model attribute name>}"> (alternate to <frm:input path="….."/>

Q. What is the difference b/w <th:text> and <th:field> tags in thymeleaf?
Ans.
- th:text is given to read data from container from different scopes and to display them on browser like reading and displaying model class object data and model attributes data.
- th:field is given to bind model class object property values/ model attribute values (reference data) to form components.

## Converting Mini Project to Thymeleaf UI based Application:

Step 1: Add Thymeleaf starter to pom.xml file

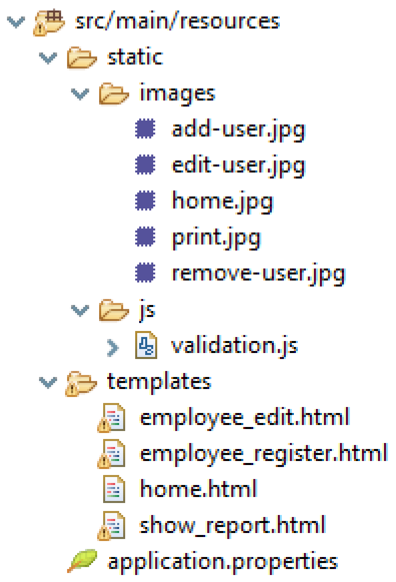
pom.xml

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Step 2: Provide global path to controller class

```
@Controller
@RequestMapping("/employee")
 public class EmployeeController {
              ………………
 }
```

Step 3: Copy js, images folders of webapp/webcontent to "static" folder of src/main/resources folder and WEB-INF/pages folder jsp files to template folder of "src/main/resources" folder, change .jsp extension to .html.



```
src/main/resources
  static
    images
      add-user.jpg
      edit-user.jpg
      home.jpg
      print.jpg
      remove-user.jpg
    js
      validation.js
  templates
    employee_edit.html
    employee_register.html
    home.html
    show_report.html
  application.properties
```

Step 4: Modify "template" folder .html files code to thymeleaf code from HTML code.

Step 5: Run the application.

Note: To execute thymeleaf code thymeleaf engine is required which will be come because of the thymeleaf starter jar file. This engine converts thymeleaf tags to HTML code and sends to browser as response.

## Directory Structure of MVCBootProj06-MiniProject-DataJPA-Thymeleaf:

- Copy past the MVCBootProj05-MiniProject-DataJPA and the name from Web Project Setting to MVCBootProj06-MiniProject-DataJPA-Thymeleaf.
- Add Thymeleaf starter.
- And follow the above step to setup.
- Then write the following code with in their respective file.

index.jsp

```
<jsp:forward page="employee/welcome.htm"/>
```

home.html

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
        <link rel="stylesheet"
th:href="@{https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstra
p.min.css}">
</head>
<body>
        <div class="container p-3 my-3 bg-primary text-white">
                <h1 style="text-align: center">Employee Details</h1>
        </div>
        <div class="container p-3 my-3 border">
                <h3 style="color: cyan; text-align: left">
                        <a th:href="@{/employee/list_emps.htm}">Get all
Employees</a>
                </h3>
        </div>
</body>
</html>
```

show_report.html

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
<link rel="stylesheet"
        th:href="@{https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/b
ootstrap.min.css}">
</head>
<body>
        <div class="container p-3 my-3 bg-primary text-white">
```

```
                    <h1 style="text-align: center">Employee
InformationReport</h1>
          </div>
          <div class="container">
                    <div th:if="${empsInfo!=null}">
                              <table class="table table-dark table-striped table-sm">
                                        <tr>
                                                  <th>Serial No.</th>
                                                  <th>Employee ID</th>
                                                  <th>Employee Name</th>
                                                  <th>Designation</th>
                                                  <th>Salary</th>
                                                  <th>Gross Salary</th>
                                                  <th>Net Salary</th>
                                                  <th>Department No.</th>
                                        </tr>
                                        <tr th:each="dto:${empsInfo}">
                                                  <td th:text="${dto.serialNo}"/>
                                                  <td th:text="${dto.empNo}"/>
                                                  <td th:text="${dto.ename}"/>
                                                  <td th:text="${dto.job}"/>
                                                  <td th:text="${dto.sal}"/>
                                                  <td th:text="${dto.grossSalary}"/>
                                                  <td th:text="${dto.netSalary}"/>
                                                  <td th:text="${dto.deptNo}"/>
                                                  <td>
                                                            <a
th:href="@{/employee/editEmp.htm(eno=${dto.empNo})}"><img
th:src="@{/images/edit-user.jpg}" width="50" height="50"/></a>
                                                            <a
th:href="@{/employee/deleteEmp.htm(eno=${dto.empNo})}"
onclick="return confirm('Are you sure to delete '${dto.ename})"><img
                                                                      th:src="@{/images/remove-
user.jpg}" width="50" height="50"></a>
                                                  </td>
                                        </tr>
                              </table>
                    </div>
                    <span th:unless="${empsInfo!=null}"><h1>Records Not
found</h1></span> <br>
```

```
                    <span th:if="${resultMsg ne null}">
                              <marquee direction="right">
                                        <h2 class="text-primary font-weight-bold"
th:text="${resultMsg}"></h2>
                              </marquee>
                    </span>
                    <br> <a th:href="@{/employee/saveEmp.htm}"><img
th:src="@{/images/add-user.jpg}"
                              width="100" height="100" /></a>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; <a
                              th:href="@{/employee/welcome.htm}"><img
th:src="@{/images/home.jpg}" width="100"
                              height="70" /></a> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
                              <a href="JavaScript:doPrint()"><img
th:src="@{/images/print.jpg}"
                              width="70" height="70">
          </div>
          <script lang="JavaScript">
                    function doPrint() {
                              frames.focus();
                              frames.print();
                    }
          </script>
</body>
</html>
```

employee_register.html

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
<link rel="stylesheet"
          th:href="@{https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/b
ootstrap.min.css}">
</head>
<body>
          <div class="container p-3 my-3 bg-primary text-white">
                    <h1 style="text-align: center">Employee Registration</h1>
          </div>
          <div class="container">
                    <script language="JavaScript" th:src="@{/js/validation.js}">
                    </script>
                    <form  th:object="${empFrm}"  onsubmit="return
validate(this)" method="POST">
                              <table class="table table-dark table-striped table-sm">
                                        <tr>
                                                  <th>Employee Name</th>
                                                  <th><input type="text"
th:field="*{ename}"/>
                                                  <span class="text-danger"
th:errors="*{ename}">
                                                  </span><span id="enameId"></span></th>
                                        </tr>
                                        <tr>
                                                  <th>Employee Designation</th>
                                                  <th><input type="text" th:field="*{job}"/>
                                                  <span class="text-danger"
th:errors="*{job}"></span><span id="jobId"></span></th>
                                        </tr>
                                        <tr>
                                                  <th>Employee Salary</th>
                                                  <th><input type="text" th:field="*{sal}"/>
                                                  <span class="text-danger"
th:errors="*{sal}"></span><span id="salId"></span></th>
                                        </tr>
                                        <tr>
                                                  <th>Department Number</th>
                                                  <th><select th:field="*{deptNo}">
                                                                      <option
th:each="dno:${deptsInfo}" th:value="${dno}" th:text="${dno}"/>
                                                            </select></th>
                                        </tr>
                                        <tr>
                                                  <td colspan="2"><input type="submit"
value="Register" /></td>
                                        </tr>
                              </table>
                              <input type="hidden" th:field="*{vflag}"/>
                    </form>
          </div>
</body>
</html>
```

employee_edit.html

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
<link rel="stylesheet"
        th:href="@{https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/b
ootstrap.min.css}">
</head>
<body>
        <div class="container p-3 my-3 bg-primary text-white">
                <h1 style="text-align: center">Employee Edit Information</h1>
        </div>
        <div class="container">
                <script language="JavaScript" th:src="/js/validation.js">
                </script>
                <form th:object="${employee}" onsubmit="return
validate(this)" method="POST">
                        <table class="table table-dark table-striped table-sm">
                                <tr>
                                        <th>Employee Name</th>
                                        <th><input type="text"
th:field="*{ename}"/>
                                        <span class="text-danger"
th:errors="*{ename}">
                                        </span><span id="enameId"></span></th>
                                </tr>
                                <tr>
                                        <th>Employee Designation</th>
                                        <th><input type="text" th:field="*{job}"/>
                                        <span class="text-danger"
th:errors="*{job}"></span>
                                        <span id="jobId"></span></th>
                                </tr>
                                <tr>
                                        <th>Employee Salary</th>
                                        <th><input type="text" th:field="*{sal}"/>
                                        <span class="text-danger"
th:errors="*{sal}"></span>
                                                <span id="salId"></span></th>
                                </tr>
                                <tr>
```

```
                                    <th>Department Number</th>
                                    <th><select th:field="*{deptNo}">
                                                <option
th:each="dno:${deptsInfo}" th:value="${dno}" th:text="${dno}"/>
                                          </select></th>
                              </tr>
                              <tr>
                                    <td colspan="2"><input type="submit"
value="Update Employee" /></td>
                              </tr>
                        </table>
                        <input type="hidden" th:field="*{vflag}"/>
                  </form>
            </div>
</body>
</html>
```

## Pagination

### Using Spring boot MVC + spring Data JPA:

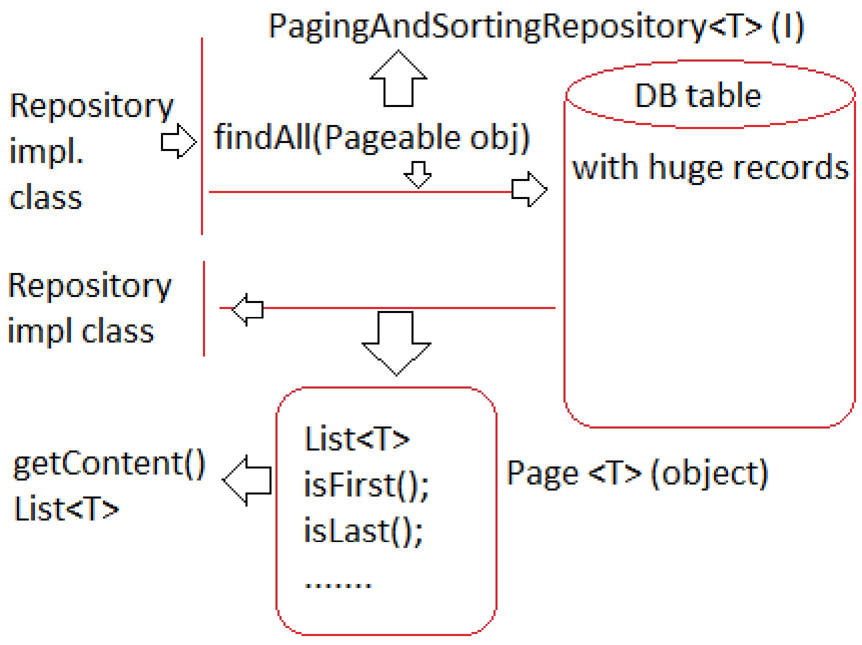
- The process of displaying huge number of records pages by page having page numbers as hyperlinks with next, previous, first, last links called pagination.
- Spring Boot MVC and Spring Data gives built-in support for pagination.

### To do pagination:

a. Take Spring data JPA repository extending from PagingAndSortingRepository <T> which gives
    a. Page<T> findAll (Pageable pageable)

```
                      |->carries pageNo, pageSize as inputs
        List<T> getContent()
        isFirst(), isLast()
        getTotalRecords()
        getTotalPages()
        getCurrentPage()
```

Note: In Pageable Object Page No input is 0 based index.



b. We should make sure request URL is always coming with query String having page, size as the req param values.

*http://localhost:3030/MVCProj28/report_paging.htm?page=... &size=...*

Because Spring MVC DispatcherServlet can create Pageable object dynamically by gathering pagesize from "size" and pageNo from "page" fixed request parameter.

```
@GetMapping("/report_paging.htm")
public String showReportWithPagination(Map<String,Object> map,
            @PageableDefault (page=0, size=5) Pageable pageable) {
            //use Repo
            Page<Employee> page=empRepo.findAll(pageable);
            //keep page object in model attrbute
            map.put("page", page);
            //return Inv
            return "emp_report";
}
```

Note: @PageableDefault makes DispatcherServlet to take given values as the default values for pageNumber, pageSize while constructing Pageable object, if the request is not given having page, size request params.

c. Use JSTL tags in jsp page or thymeleaf tags to read page Object data and to display report having page number and other hyperlinks like next, previous and etc.

## Directory Structure of MVCBootProj07-MiniProject-DataJPA-Pagination:

- Copy past the MVCBootProj05-MiniProject-DataJPA and the name from Web Project Setting to MVCBootProj07-MiniProject-DataJPA-Pagination.
- Then write the following code with in their respective file.

### home.jsp

```
            <h3 style="color: cyan; text-align: left">
                  <a href="report_paging.htm">Get all Employees through
Pagination</a>
            </h3>
```

### IEmployeeMgmtService.java

```
      public Page<EmployeeDTO> getPageData(Pageable pageable);
```

### EmployeeMgmtServiceImpl.java

```
      @Override
      public Page<EmployeeDTO> getPageData(Pageable pageable) {
            //use Repo
            Page<EmployeeBO> pageBO = empRepo.findAll(pageable);
            //Get List collection
            List<EmployeeBO> listBO = pageBO.getContent();
            //convert listBO to listDTO
            List<EmployeeDTO> listDTO = new ArrayList<>();
            listBO.forEach(bo->{
                  EmployeeDTO dto = new EmployeeDTO();
                  BeanUtils.copyProperties(bo, dto);
                  dto.setSerialNo(listDTO.size()+1);
                  dto.setGrossSalary(dto.getSal()+dto.getSal()*0.3f);

      dto.setNetSalary(dto.getGrossSalary()+dto.getGrossSalary()*0.1f);
                  dto.setSal(Math.round(dto.getSal()));
                  listDTO.add(dto);
            });
            //convert listDTO to pageDTO
            Page<EmployeeDTO> pageDTO = new PageImpl<>(listDTO,
      , pageBO.getPageable(), pageBO.getTotalElements());

            return pageDTO;
      }
```

EmployeeRepository.java

```
public interface EmployeeRepository extends
PagingAndSortingRepository<EmployeeBO, Integer> {
```

EmployeeController.java

```
        @GetMapping("/report_paging.htm")
        public String showPageData(Map<String, Object> map,
@PageableDefault(page = 0, size = 3, sort = "ename", direction =
Direction.DESC)Pageable pageable) {
                //use service
                Page<EmployeeDTO> pageDTO =
service.getPageData(pageable);
                //keep pageDTO object as model attribute
                map.put("page", pageDTO);
                //return LVN
                return "show_page_report";
        }
```

Show_page_report.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1" isELIgnored="false"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstr
ap.min.css">
</head>
<body>
        <div class="container p-3 my-3 bg-primary text-white">
                <h1 style="text-align: center">Employee
InformationReport</h1>
        </div>
        <div class="container">
        <c:choose>
                    <c:when test="${page ne null && !empty page}">
                         <table class="table table-dark table-striped table-
sm">
                              <tr>
                                   <th>Serial No.</th>
                                   <th>Employee ID</th>
                                   <th>Employee Name</th>
                                   <th>Designation</th>
                                   <th>Salary</th>
                                   <th>Gross Salary</th>
                                   <th>Net Salary</th>
                                   <th>Department No.</th>
                              </tr>
                              <c:forEach var="dto"
items="${page.getContent()}">
                                   <tr>
                                        <td>${dto.serialNo}</td>
                                        <td>${dto.empNo}</td>
                                        <td>${dto.ename}</td>
                                        <td>${dto.job}</td>
                                        <td>${dto.sal}</td>
                                        <td>${dto.grossSalary}</td>
                                        <td>${dto.netSalary}</td>
                                        <td>${dto.deptNo}</td>
                              </c:forEach>
                         </table>
                         <br>
                         <div style="text-align: center;">
                              <a href="report_paging.htm?page=0">First
Page</a>&nbsp;&nbsp;&nbsp;
                              <c:if test="${!page.isFirst()}">
                                   <a
href="report_paging.htm?page=${page.getNumber()-1}">Previous</a>
                              </c:if>&nbsp;&nbsp;&nbsp;
                              <c:forEach var="i" begin="0"
end="${page.getTotalPages()-1}" step="1">
                                   [<a
href="report_paging.htm?page=${i}">${i+1}</a>]&nbsp;&nbsp;&nbsp;
                              </c:forEach>&nbsp;&nbsp;&nbsp;
```

```
                                        <c:if test="${!page.isLast()}">
                                                <a
href="report_paging.htm?page=${page.getNumber()+1}">Next</a>
                                        </c:if>&nbsp;&nbsp;&nbsp;
                                        <a
href="report_paging.htm?page=${page.getTotalPages()-1}">Last Page</a>
                                </div>
                        </c:when>
                </c:choose>
        </div>
</body>
</html>
```

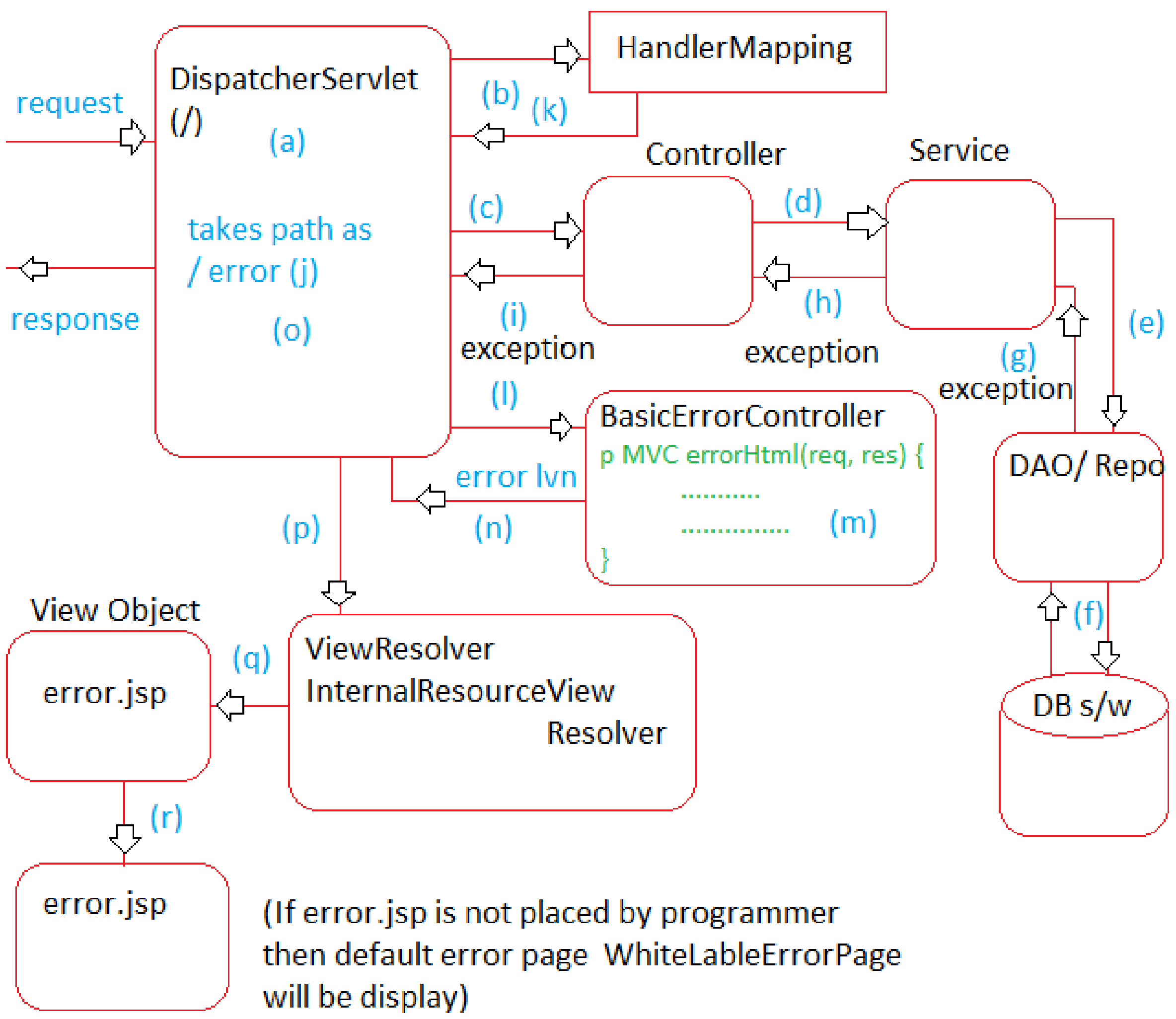
## Exception Handling in Spring Boot MVC

- Spring Boot MVC gives built-in support for Exception handling and also default error pages to display the if these error messages are not use-friendly, we can customize them by taking our error messages in our own error pages.
- In Spring MVC/ Spring Boot MVC the exceptions raised in any layer will come to Dispatcher Servlet through exception propagation and DS internally uses the controller called BasicErrorController (implementation class of ErrorController (I)) to render error pages with default error messages on to the browser (with support of errorHtml() or error() handler methods) having request path "/error".

### Flow of execution:

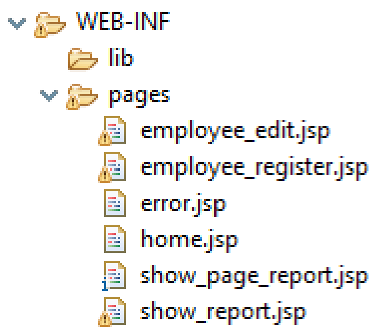
- DS calls handler method of our controller class through Handler Mapping.
- DS gets problem (Exception) from Controller (controller's own exception or propagated from DAO, Service class)
- DS maps the request to BasicErrorController through HandlerMapping having request "/error" and errorHtml () or error () handler method take the request and render default error page.

Note: All the operations in DS (DispatcherServlet) takes place from doDispatch (-, -) that is called through doService(-,-) method.



## To configure custom error page for better user-friendly UI:

- Add error.jsp in the place your view resolver pointing to (like prefix value of ViewResolver configuration).



## Directory Structure of MVCBootProj08-MiniProject-ExceptionHandling:

- Copy past the MVCBootProj07-MiniProject-DataJPA-Pagination and the name from Web Project Setting to MVCBootProj07-MiniProject-ExceptionHandling.
- Then write the following code with in their respective file.

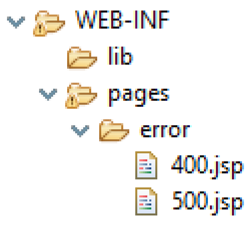
error.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<html>
<head>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min
.css">
</head>
<body>
        <div class="container p-3 my-3 bg-primary text-white">
                <h1 style="text-align: center">Error Page</h1>
        </div>
        <div class="container p-3 my-3 border">
                <h2>Internal Problem: Try after some time</h2>
                <h3>Status: ${status}</h3>
                <a href="welcome.htm"><img src="images/home.jpg"
width="100" height="70"/></a>
        </div>
</body>
</html>
```

HTTP Status Codes:
100 -199: Information (1XX)
200-299: Success (2XX)
300-399: Redirection (3XX)
400-499: Incomplete (Client-side problem) (4XX)
500-599: Server-side error (5XX)

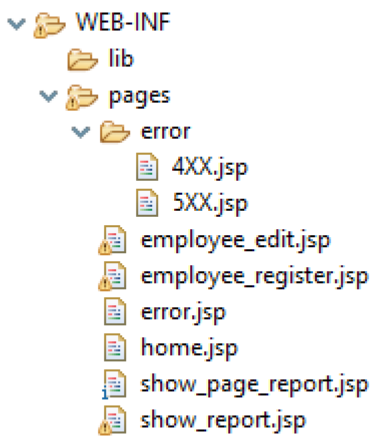
Note: 4XX, 5XX are error status code (400-599).

- Instead taking same error page for all error codes we can design specific page for specific error code having <error code>.jsp as the file name is /error folder.



```
WEB-INF
  lib
  pages
    error
      400.jsp
      500.jsp
```

- If specific error code page is not found in "error" folder then it goes to common error page (error.jsp)
- We can design specific error code page having names like 4XX.jsp, 5XX.jsp to same error pages for all 4xx errors and 5xx errors.


```
WEB-INF
    lib
    pages
        error
            4XX.jsp
            5XX.jsp
        employee_edit.jsp
        employee_register.jsp
        error.jsp
        home.jsp
        show_page_report.jsp
        show_report.jsp
```

4XX.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<html>
<head>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min
.css">
</head>
<body>
        <div class="container p-3 my-3 bg-primary text-white">
                <h1 style="text-align: center">Error Page</h1>
        </div>
        <div class="container p-3 my-3 border">
                <h2>Internal Problem - Clint Side error</h2>
                <h3>Status: ${status}</h3>
                <a href="welcome.htm"><img src="images/home.jpg"
width="100" height="70"/></a>
        </div>
</body>
</html>
```

5XX.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<html>
<head>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min
.css">
</head>
<body>
	<div class="container p-3 my-3 bg-primary text-white">
		<h1 style="text-align: center">Error Page</h1>
	</div>
	<div class="container p-3 my-3 border">
		<h2>Server Internal Problem</h2>
		<h3>Status: ${status}</h3>
		<a href="welcome.htm"><img src="images/home.jpg"
width="100" height="70"/></a>
	</div>
</body>
</html>
```

- We can create Custom Exception class linking with our choice error response status code and we can use that exception in our application as shown below.

EmployeeNotFoundException.java

```
package com.nt.exception;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;
@ResponseStatus(code = HttpStatus.SERVICE_UNAVAILABLE)
public class EmployeeNotFoundException extends RuntimeException {
	public EmployeeNotFoundException() {
		super();
	}
	public EmployeeNotFoundException(String msg) {
		super(msg);
	}
}
```

## Embedded Servers in Spring Boot MVC

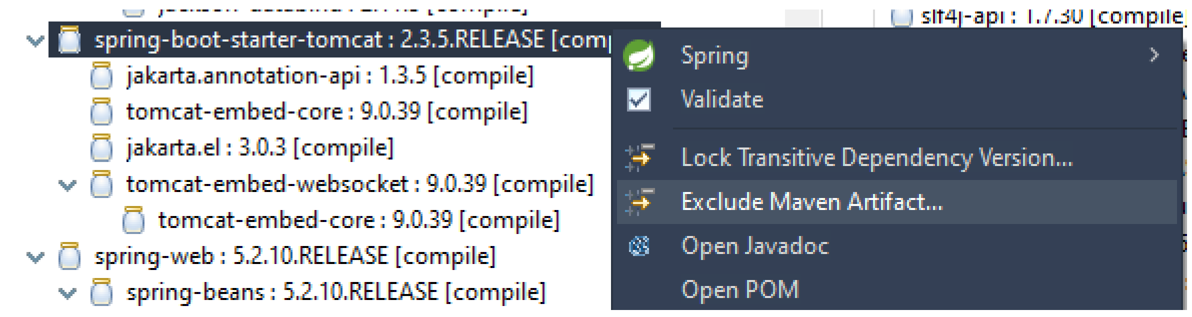
- As of now Spring Boot MVC is supporting only 3 embedded and multiple (all) external server.
    a. Apache Tomcat (default)
    b. Eclipse Jetty
    c. JBoss Undertow

- In development and testing of Spring Boot MVC applications "Embedded" servers will be used, in staging (UAT) and production environment "External Servers" will be used.

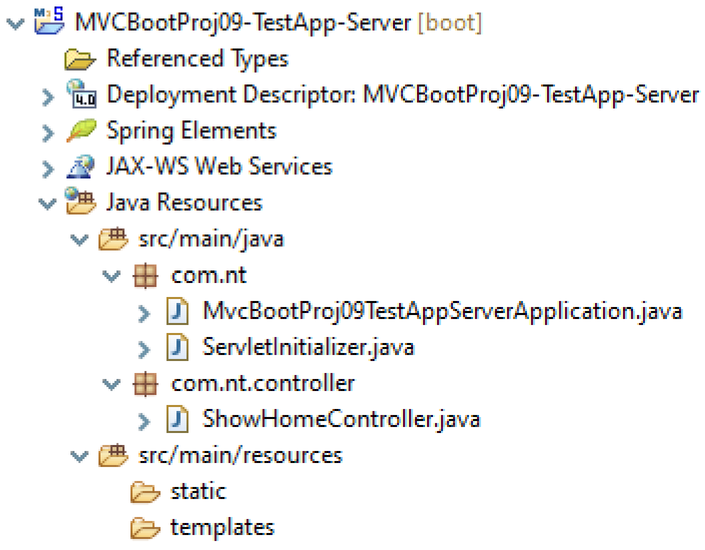
To configure Jetty server with Spring Boot MVC application:
Step 1: Add spring-boot-starter-jetty and apache-jsp dependency

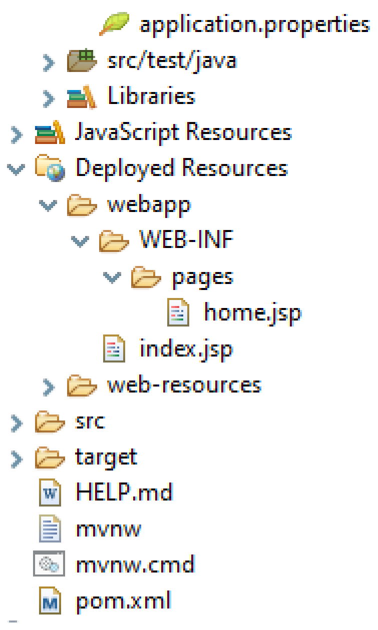
Step 2: Remove all entries related tomcat exclude from spring-boot-starter-web, better to exclude spring-boot-starter-tomcat.

Note: To exclude go to Dependency Hierarchy tab of pom.xml then go to spring-boot-starter-tomcat jar file, right click then click on Exclude Maven Artifact.


```
spring-boot-starter-tomcat : 2.3.5.RELEASE [comp
  jakarta.annotation-api : 1.3.5 [compile]
  tomcat-embed-core : 9.0.39 [compile]
  jakarta.el : 3.0.3 [compile]
  tomcat-embed-websocket : 9.0.39 [compile]
    tomcat-embed-core : 9.0.39 [compile]
spring-web : 5.2.10.RELEASE [compile]
  spring-beans : 5.2.10.RELEASE [compile]
```
slf4j-api : 1.7.30 [compile
Spring
Validate
Lock Transitive Dependency Version...
Exclude Maven Artifact...
Open Javadoc
Open POM

Directory Structure of MVCBootProj09-TestApp-Server:


```
MVCBootProj09-TestApp-Server [boot]
  Referenced Types
  Deployment Descriptor: MVCBootProj09-TestApp-Server
  Spring Elements
  JAX-WS Web Services
  Java Resources
    src/main/java
      com.nt
        MvcBootProj09TestAppServerApplication.java
        ServletInitializer.java
      com.nt.controller
        ShowHomeController.java
    src/main/resources
      static
      templates
```



```
            application.properties
        > src/test/java
        > Libraries
    > JavaScript Resources
    v Deployed Resources
        v webapp
            v WEB-INF
                v pages
                    home.jsp
                index.jsp
        > web-resources
    > src
    > target
      HELP.md
      mvnw
      mvnw.cmd
      pom.xml
```

- Develop the above directory structure using Spring Starter Project option and package and classes also.
- During this choose the following starter dependency
    - Spring Web
- Many jars dependencies will come automatically in pom.xml, and as per our required we can add also in <dependencies> tag.
- Then write the following code with in their respective file.

ShowHomeController.java

```java
package com.nt.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class ShowHomeController {

	@GetMapping("/welcome")
	public String showHomePage() {
		return "home";
	}
}
```

index.jsp

```jsp
<jsp:forward page="welcome"/>
```

home.jsp

```
	<h1 style="color: red; text-align: center;">Welcome to Spring 
Boot</h1>
```

application.properties

```
#View resolver
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp

#server port
server.port=4040
```

pom.xml

```
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
			<exclusions>
				<exclusion>
					<groupId>org.junit.vintage</groupId>
					<artifactId>junit-vintage-engine</artifactId>
				</exclusion>
			</exclusions>
		</dependency>
		<!-- Jetty server related jars -->
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-jetty</artifactId>
		</dependency>
		<dependency>
			<groupId>org.eclipse.jetty</groupId>
			<artifactId>apache-jsp</artifactId>
		</dependency>
	</dependencies>
```

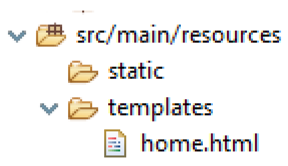
Configuring Undertow server with Spring Boot MVC application:
Step 1: Add spring-boot-starter-undertow dependency in pom.xml
Step 2: Same as jetty server.

Note: As of taking jsp components as view comp in undertow is very complex i.e., we need to change so many configurations. So, it recommended alternate view components like thymeleaf for that we need to the following dependencies pom.xml

pom.xml

```
<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-test</artifactId>
		<scope>test</scope>
		<exclusions>
			<exclusion>
				<groupId>org.junit.vintage</groupId>
				<artifactId>junit-vintage-engine</artifactId>
			</exclusion>
		</exclusions>
	</dependency>
	<!-- Undertow server related jars -->
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-undertow</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-thymeleaf</artifactId>
	</dependency>
</dependencies>
```

- Use thymeleaf stuff take a home.html file in src/main/resources/templates folder and write the following code in home.html



home.html

```
<!DOCTYPE html>
<html xmlns:th="https://www.thymeleaf.org/">
<head>
<meta charset="ISO-8859-1">
</head>
<body>
	<h1 style="color: red; text-align: center;">Welcome to Spring
Boot</h1>
</body>
</html>
```

Note: In both above configuration run the application as Spring Boot App or Java Application because of embedded server. Open the main starter class right click on that/ Right click on the application -> Run As -> Spring Boot App/ Java Application.

Then open the browser as type the below URL for lunching the home page

http://localhost:4040/welcome

---------------------------------------------- The END ----------------------------------------------