right click project → configure build path → left side of project facets
↓
Java ⇒ 1.8

Ques. Capacity of Hashmap, hashtable &
DS of Hashmap &

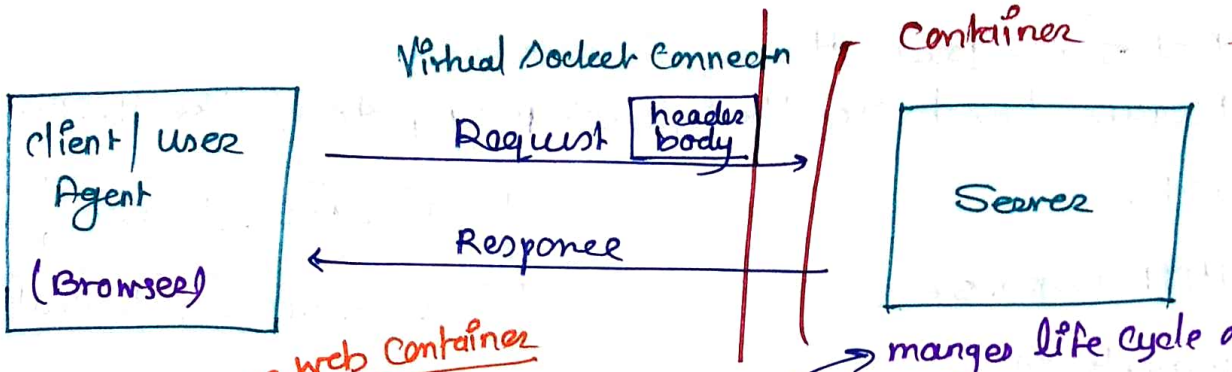
→ window → preferences → Java → Installed JRE → add
↓
directory
↓
given path upto bin
↓
add new JRE

* Servlets *-

* client side technology ⇒ HTML, Javascript, bootstrap.
* Server side technology ⇒ JSP, Servlet (use to create dynamic webpages)
  ↳ (Java Server pages)

→ CGI → (Common Gateway Interface) Ex:- language is made
⇒ process based. Seperate process each and every request.
⇒ Process is heavy weight.

* HTML → markup language → boor → no main method → no required any Compiler → does not req. any Seperate software



* Apache tomcat → web Container ⇒ It is work as a Container as well as Server. → manages life cycle of Servlet.

* HTTP protocol ⇒ Hypertext transfer protocol
⇒ It is Connectionless protocol? Why?

→ Servlet ⇒ Java pgm.
⇒ Thread based prog. language.
⇒ Create Seperate thread for each & every incoming client request.
⇒ Thread is lightweight - it takes less memory & less execution time thus it will improve app's performance.

⇒ URL → Uniform Resource locator (www.gmail.com/index.HTML)

⇒ URI ⇒ www.gmail.com (uniform Resource Identifier)

Apache download ⇒ JDK 8 → apache 8.5 → download → 8.5.96
→ 64bit win zip → lib (Jar files)

* Every time creating dynamicallyweb project we need to add jar files.
* for JSP → JSP Jar file.

* Browser ⇒ Html content.
* Website ⇒ Collection of web pages (text, audio, image).
* Static website ⇒ Common for every end user.
* Dynamic website ⇒ End to end user ~~request~~ view changes.

* Web-appln architecture ⇒

→ Request Response model.
→ client - browser, user agent.
→ www.gmail.com/index.html.
→ Web server locates the request resource.
→ Request travels over the internet using protocol like HTTP - HTTP request.
→ HTTP Response.

Web server ⇒ Tomcat
Web Container ⇒ Tomcat

* Components ⇒

1) Web Server ⇒ physical on which the appln resides called as web server
2) Web Container ⇒ Once that manages life cycle of servlet.

* Website ⇒ Collection of web pages (text, image, audio)
→ first page ⇒ Home page (welcome page)
→ URL (Address)

* Static website ⇒ Common for every end user.
1) client side tech ⇒ HTML, CSS, JS. ---
* Dynamic website ⇒ End to end user view changes.
→ Server side tech ⇒ Servlet, Jsp, php. ---

* HTTP (Hypertext transfer protocol) :- It is a data communication protocol used to establish communication bet client & server.

file → new → create dynamic web projects → project name → Dynamic web module Version (2.5) folder {(3.0) is notation based} → next → src → next.
→ Generate web.xml deployment descriptor [✓] → finish.
(web.xml → configuration file)
→ Add a Jar file servletapi.sar in web-content → lib folder put file.

Project Structure ⇒ Web Content ⇒ web.xml file, HTML, ..
⇒ All resources will be here
⇒ Lib folder ⇒ All Jar files
⇒ open web.xml → click source → delete some tags except html.index.

⇒ Web content → new (right click) → HTML file
next ← index.html ← file name
↳ Resources name should be lower.

{In JS while port in website called as Document}

⇒ <body> Hello world </body>
<title> test </body>
⇒ right click on file → open with → web browser.
{window → web browser → & we can change browser}

{window → show View → Servers
click on link to add Servers ⇒}
↓
→ Define new Server ⇒ select apache tomcat 8.5 ⇒ next
⇒ directory (browse) . file → select ⇒ JRE (1.8) → finish
(till bin don't go inside bin)
↓
{If not then click on Installed JRE.}

{⇒ Server started then last Smt will be Server startup in secms
⇒ ● → start server.
■ → stop Server}

* Double click on Server → We can change port No of Server. ⇒ 8080
⇒ 8081
⇒ 8082.

→ To delete server → delete server folder

→ click on appln → run as server
↓
Same steps.

{ localhost ⇒ server IP Address
8080 ⇒ Port No. }

{ header contain all client info }

* form.html → change in web.xml file also same name at welcome page tag

→ <title> student form </title>

→ View ⇒ HTML, JSP

→ POJO → Java Bean → model ⇒ this carries actual data
(Plain old Java object)

→ Servlet is a Controller. ← for that Controller required.
↓
(Interface or mediator betn View & model)

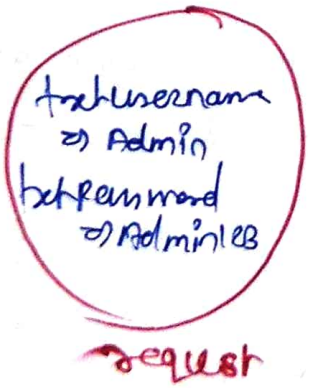
* Java Resource → src → right click → Servlet → add Jar file to lib (Servlet-Api)

configure build path.

* click on appn → Build path → Libraries → add Library → add JAR file external
↓
Select Servletapi.Jar
↓
Apply

→ Referenced libraries → has Jar file.

web.xml will always info about web appln



* url pattern in xml file & action name should be match & from Html file
then servlet will be called from servlet-class tag

→ get method → In URL it will show username & password,
→ localhost:8080/myapp1/serv1?txtusername=admin&txtpass=admin123.
query string

→ Post method → username & password hidden.

```
* Package com.app;
  public class myservlet extends HttpServlet {
  Protected Void doGet(HttpServletRequest request, HttpServletResponse)
                    throws ServletException, IOException {
  {
  →
      responce.setContentType("text/html"); // Multipurpose mail extension type (MIME)
      printWriter pw = responce.getWriter(); // to write something
      // Read username & password.
      String uname = request.getparameter("txtUsername");
      String pwd = request.getparameter("txtPassword");
                  (should same in Html file)

      if(uname.equals("admin") && pwd.equals("admin123"))
          {
            pw.write("login Successfully" + uname);
      else {
            pw.write("Invalid");
          }
  }
```

Que. Why HTTP is said to be stateless protocol & bcoz each request is executed independently, without any knowledge of Requests that were executed before it & which means once the transaction ends, the connection betn the browser & Server is also lost.

form.html ⇒

```
<body>
<h3> Student Registration form </h3>
<form action = "serv1" method = "get">
                  → URL mapping.
<table>
<tr>
 <td> Enter Username : </td>
 <td> <input type = "text" name = "txtUsername"></td>
</tr>
<tr>
<td> Enter password : </td>
```

```
<td> < input type = "Password" name = "txtPassword" > </td>
</tr>
<tr>
<td> </td>
<td> < input type = "Submit" Value = "submit" > </td>
</tr>
</table>
</form>
</body>
</html>
```

Dynamic web module Version ⇒ 3.0

{ status ⇒ 200 okay }

Project Name ⇒ QueryStringDemo

javax.servlet.GenericServlet. → abstract class

→ URL mappings ⇒ /S1 (In create servlet box)

@WebServlet ("/S1")

{ localhost:8080/QueryStringDemo/S1?txtName = rahul & txtAge = 1

→

```
Package com.app;
@WebServlet ("/S1")
public class Servlet extends GenericServlet implements Servlet
    public Void Service (ServletRequest request,
ServletResponce responce) throws ServletException,
IoException {
```

```
response.setContentType("text/Html");
PrintWriter pw = response.getWriter();
String name = request.getParameter("txtName");
// String no = request.getParameter("txtAge");
int age = Integer.parseInt(request.getParameter("txtAge"));
if(age>18) {
    pw.write("Eligible for Voting "+name);
else {
    pw.write("Not eligible for Voting "+name);
}
```

Converting String to Integer.

} Run on Server then ⇒ URL

Server IP Address | Port No | App in Name | URL mapping

→ Query String ⇒ HTTP://localhost:8080/querystringdemo/s1?txtName=rahul&txtAge=10

---

1) Static Response ⇒ Performing any action with or without executing any resource at server machine then

2) Dynamic Response ⇒ generate any response from Server by performing some action or by executing any resource at server machine.

* HTTP port NO ⇒ 80

→ HTTP request Contains URL, HTTP methods (Get, Post, Head ----), optional Parameters.

→ HTTP Response ⇒ Status code :- 200 ok

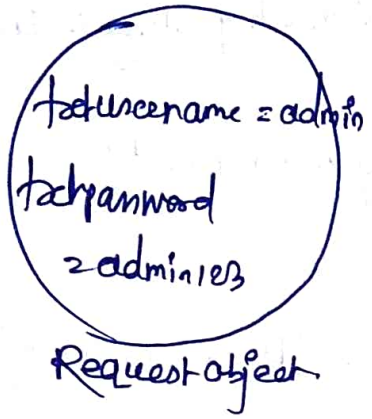
Content-type :- Image, doc, pdf

Content = actual Content.

→ when to get & post ?

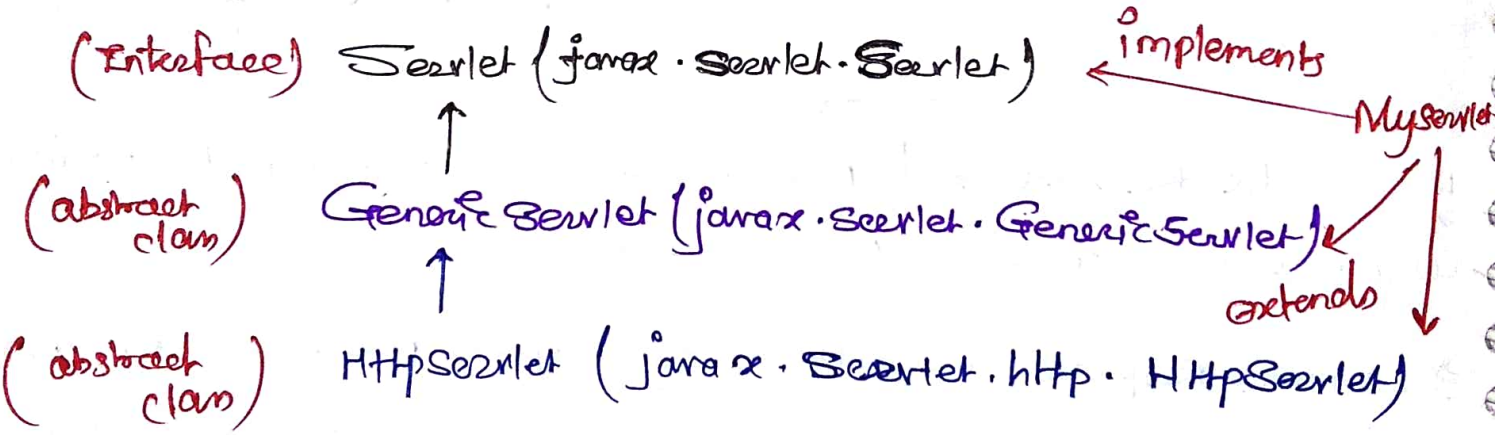
GET → if we want something from server.
POST → if we want give something to server.

→ HTTP is a stateless protocol → Every request treated as new request.



→ Servlet ⇒ Platform independent java classes executes on server

→ Servlet API :: javax.servlet
javax.servlet.http



* Life Cycle of Servlet ⇒

1) loading & Instantiation
2) Initialization
3) Servicing the request.
4) Destroying the Servlet.

→ Data Can be Passed in two ways ⇒

1) HTML form.

2) Query string.

After creating a Servlet next box:- ⇒

URL mapping ⇒ /Servl → This url-mapping in xml file & HTML should match.

1) Name = driverName.
   Value = Com.mysql.jdbc.Driver

2) username → name
   root → value

3) Password
   root

4) intvalue
   7.8

4 Initialization Parameters.
created.
↓
In Web.xml file this will created
<init-param>

In Servlet class ⇒ right click → Source → override implements methods → Goto Generic Servlet.

init() → method from Generic Servlet.

* Servlet initialization ⇒ parameter initialization.

→ Two ways ⇒ 1) ServletConfig ⇒ Interface.
⇒ An object of ServletConfig is created by a web-Container for each Servlet.

⇒ This object is used to get information configuration from web.xml file.

⇒ ServletConfig getServletConfig() of Servlet.

Syntax :-

To provide initialized parameter

<web-app>

<servlet>

— —

<init-param>

<param-name> parameterName </param-name>

<param-value> parameterValue </param-value>

</init-param>

</servlet>

</web-app>

{ web.xml file it we show } →

2) ServletContext ⇒ (Multiple Servlet असतो तेव्हा project मध्ये ते use ServletContext means object Access करता येणारा servlet साठी)

→ object - Container

This is only one object per web application.

ServletContext getServletContext()

outside servlet tag in web.xml file

<context-param>

<param-name> parameterName </param-name>

<param-value> parameterValue </param-value>

</context-param>

* init() ⇒ 1) Read initialization parameters using ServletConfig object.

2) Initialize one time activities such as registering a Db driver, logging service using the ServletContext object.

↓

called once throughout servlet.

{method name start with has always have return type boolean}

→ new project → InitParameter Demo

→ index.html

```
<body>
<a href = "servl"> click here !!! </a>
</body>
</html>
```

"servl" → URL-mapping

→ web.xml

for servlet context

```
<context-param>
    <param-name> commonMsg </param-name>
    <param-value> Hello world !!! </param-value>
</context-param>

<servlet>
    <description> </description>
    <display-name> Servlet1 </display-name>
    <servlet-name> Servlet1 </--->
    <servlet-class> com.app.Servlet1 </--->
    <init-param>
        <description> </--->
        <param-name> driverName </param-name>
        <param-value> com.mysql.jdbc.Driver </--->
    </init-param>
    <init-param>
        <description> </--->
        <param-name> username </--->
        <param-value> root </--->
    </init-param>
```

```
<init-param>
    <description></description>
    <param-name>password</param-name>
    <param-value>root</param-value>
</init-param>
<init-param>
    <description></description>
    <param-name>intrate</param-name>
    <param-value>7.8</param-value>
</init-param>
</servlet>
<servlet-mapping>
    <servlet-name>Servlet1</servlet-name>   (classname)
    <url-pattern>/serv1</url-pattern>
</servlet-mapping>
</web-app>
```

→ Servlet :- (2.5)

```
public class Servlet1 extends HttpServlet {

    private String dname, uname, pwd, msg;
    private double irate;
    private ServletConfig config;

@override
    public void init() throws ServletException {
        super.init();
        System.out("In init() method");
```

- ServletConfig
- ServletContext.

```
Config = getServletConfig();
dname = Config.getInitparameter("driveName");
irate = Double.parseDouble(config.getInitParameter(
                                "intRate"));
ServletContext Context = getServletContext();
msg = Context.getInitparameter("commonMsg");
}

protected void doGet(HttpServletRequest request, HttpServletResponce reparce)
                        throws ServletException, IOException {

    Sysout("In doGet()");
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    // pw.write("<he>Drive name =" + dname + "</he>");
    // pw.write("<br><he> IntRate =" + irate + "</he>");
    pw.write("<br><he> Message =" + msg + "</he>");
    Enumeration<String> e = Config.getInitParameterNames();
    String str = "";
    pw.write("<h1> parameter Initialization in Servlet</h1>");
    While(e.hasMoreElements()) {
        Str = e.nextElement();
        pw.write("<br> parameter name =" + Str);
        pw.write("parameter Value =" + Config.getInitParameter(
                                        Str));
    }
}
```

Single Parameter

for multiple Parameters

→ New project for ServletConfig
InitParameterDemo1 (3.0)

→ ~~Make generic servlet~~ . superclass .

→ @override init method.

```
@WebServlet(
        urlPatterns = {"/Serv1"},
        initParams = {
            @WebInitParam(name="uname", value="root"),
            @WebInitParam(name="password", value="root")
        })
```

→ Run on Server ⇒ In URL ↴
// http://localhost:8080/InitParametersDemo1/Serv1

(Same as previous program in Servlet .)

→ New project ⇒ JavascriptDemo Javascript Registration form.

```
<head>
<Script type = "text/javascript">
        function Validation() {
            if(document.Contact_form.txtUsername.value == "") {
                alert("Please Enter a username");
                return false;
            }
            if((document.Contact_form.gender[0].checked == false)
            && (document.Contact_form.gender[1].checked == false)) {
                alert("Please select your gender");
                return false;
            }
```

(fn name)

```
if(document.Contact_form.age.selectedIndex == 0) {
        alert("Please select your Age Group");
        return false;
}
if(document.Contact_form.check.checked == false) {
        alert("please agree terms & Cond");
        return false;
}
}
</script>
</head>
<body>
    <h2> Registration form </h2>
    <form action = "Sesv1" method = "get" name = "Contact_form"
                                onsubmit = "return validation()">

    <table border = "1">

        <tr>
          <td> Enter a username: </td>
          <td> : <input type = "text" name = "txtUsername"></td>
          <td> . </td>
        </tr>
        <tr>
          <td> Enter a password: </td> name = "txtPassword"></td>
          </td><input type = "Password" name = "txtPassword"></td>
        </tr>
        <tr>
          <td> Gender: </td>
          <td> Male < input type = "radio" name = "gender" value = "MALE">
               Female < input type = "radio" name = "gender" value = "female">
          </td>
        </tr>
```

```
<tr>
  <td> Enter age: </td>
  <td> <select name = "age">
        <option value = "select" selected = "selected"> select </option>
        <option value = "21-30"> 21-30 </option>
        <option value = "31-40"> 31-40 </option>
        <option value = "41-50"> 41-50 </option>
  </select> </td>
</tr>
<tr>
  <td> <input type = "checkbox" name = "check"> </td>
  <td> Agree terms and Condn !!! </td>
</tr>
<tr>
  <td> <input type = "reset" value = "RESET"> </td>
  <td> <input type = "submit" value = "SUBMIT"> </td>
</tr>
</table>
</form>
</body>
</html>
```

* Servlet Collaboration ⇒ Communication betn two Servlets.

→ projectNew → RequestDispatcherDemo (3.0)

1) RequestDispatcher interface :-

Dispatches the request to another resource.

a) public void forward(ServletRequest request, ServletResponse response);

⇒ forward the request from a Servlet to another resource

b) public void include (ServletRequest request, ServletResponce responce);

⇒ Include the content of resource in the Response.

* RequestDispatcherDemo (8.0)

→ index.html

```
<form action = "serv1"  method = "post">
Student Registration form
Enter username : [          ]
Enter Password : [          ]
[Submit.]
```

→ Servlet1

```
@WebServlet ("/serv1")
public class Servlet1 extends HttpServlet {
protected void doGet (HttpServletRequest request, HttpServletResponce responce)
        throw ServletException, IOException {

    response.setContentType ("text/html");
    PrintWriter pw = response.getWriter();
    //Read username & password
    String uname = request.getParameter ("txtUsername");
    String pwd   = request.getParameter ("txtPassword");
    if (uname.equals ("admin") && pwd.equals ("admin123")) {
        RequestDispatcher rd = request.getRequestDispatcher ("serve");
        rd.forward (request, responce);
    } else {
        pw.write ("Invalid username or Password");
        RequestDispatcher rd = request.getRequestDispatcher ("index.html");
        rd.include (request, responce);
    }
```

→ Servlet

```
@WebServlet("/serv2")
public class Servlet2 extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.write("welcome" + request.getParameter("txtUsername"));
}
```

---

2) sendRedirect(String URL) - HttpServletResponse Interface

→ used to redirect response to another resource.

→ Every time a new request is being sent.

→ client side.

* SendRedirect Demo (3.0)

→ Search.Html

```
<head>
</head>
<body>
<h3> Search Engine Demo </h3>
<form action = "Serv1" method = "post">
<table>
<tr>
<td> Enter Data </td>
<td> input type = "text" name = "str" ></td>
</tr>
<tr>
<td> Select Engine: *</td>
```

```
<td><input type = "radio" name = "Engine" value = "google">Google
<input type = "radio" name = "Engine" value = "yahoo"> YAHOO
<input type = "radio" name = "Engine" value = "Bing"> BING
</td>
<tr>
<td></td>
<td><input type = "submit" value = "submit"></td>
</tr>
</table>
</form>
</body>
</Html>
```

→ myservlet.java

```
@WebServlet("/serv1")
public class myservlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
                      Response) throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        String data = request.getParameter("str");
        String engine = request.getParameter("Engine");
        String url = "";

        try {
            if(data.length() == 0) {
                throw new Exception();
            } else if(engine == null) {
                throw new Exception();
            }
```

```
if( engine.equals("Google")) {
    url = "HTtps://www.google.com/search?q="+data;
}
if( engine.equals("YAHOO")) {
    url = "HTtps://in.search.yahoo.com/search?p="+data;
}
if( engine.equals("Bing")) {
    url = "https://www.bing.com/search?q="+data;
}
responce.sendRedirect(url);
}catch(Exception e) {
responce.sendRedirect("error.html");
}
```

→ error.html

```
<body>
<h3> Error occured !!! </h3>
</body>
```

---

* Servlet filter :-

→ used for preprocessing & postprocessing of data.

```
public void doFilter(HTTPServletRequest request, HTTPServletResponce
                     responce, FilterChain chain);
```

→ FilterDemo (3.0)

right click on base package → new → Filter.

→ FiltDemo1 (e.5)

→ web.xml

```
</servlet-mapping>
<filter>
    <display-name> MyFilter </display-name>
    <filter-name> MyFilter </filter-name>
    <filter-class> com.app.myfilter </filter-class>
</filter>
<filter-mapping>
    <filter-name> MyFilter </>
    <url-pattern> /serv1 </>
</filter-mapping>
</web-app>
```

→ index.html

```
<head>
<body>
<a href = "serv1"> click here !!! </a>
</body>
</html>
```

→ myServlet.java

```
Public class myServlet extends HttpServlet {
  protected void doGet (--, --) throws -- {
      response.setContentType("text/html");
      PrintWriter pw = response.getWriter();
      pw.write("<br> Servlet Invoked <br>");
  }
```

→ myfilter.java

```
Public clas Myfilter implements filter {

pubic Void dofilter(ServletRequest request, ServletResponse response,
                    filterchain chain) throws ---- {
    // place your code here.
    printwriter pw = response.getwriter();
    pw.write("filter Invoked Before");
    // pass the request along filter chain
    chain.dofilter(request, response);
    pw.write("filter Invoked after");
}
```

add → index.html:

```
<body>
<!-- <a href="serv1">click here !!! </a> -->
<form action="serv1" method="get">
```

{ Enter name: [ ] [Submit] }

add → myfilter.java

```
public Void dofilter(---------, ---, filterchain chain)
                    throws ---- {
    // place your code here
    Printwriter pw = response.getwriter();
    pw.write("filter invoked Before");
    String user = request.getParameter("txtName");
    user = user.trim(); // Remove white spaces.
    request.setAttribute("uname", user);
    // Pass the request along the filter chain
```

```
chain.filter(request,response);
pw.write("filter invoked after");
```

add } → Servlet1.java:-

```
protected void doGet(--) throws -- {
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.write("<br> servlet invoked <br>");
    String name = (String) request.getAttribute("uname");
    if(name.equals("RAHUL")){
        pw.write("valid user:"+name);
    }else{
        pw.write("Invalid user");
    }
}
```

Lab work:-
Append
Username
of Rahul
write after
chain.filter()
using response obj

→ filterDemo (3.0)

→ Myfilter.java

```
@WebFilter("serv1")
public class Myfilter implements Filter( ---- ) {
```

* Session handling:-

Block cookies
Add gmail
& check output

→ Particular time interval.

→ A session starts from very 1st user till appln is closed.

Why session tracking?

→ web appln works on HTTP:

HTTP is stateless protocol:-

client state and execution history is not maintained

→ Simulation Assuming trading is absent;
↳ (Demo)

client	Server
→ Access facebook.com	Locate index.html & send it.
→ Login username & password	Identify the user & Validate, respond the users home page.
→ click on wall	Cannot understand which users wall, as no info is provided, respond with an error message & ask user to login again.

* Session-tracking ⇒

Is way to maintain state (data) of a user.

→ 1) HTTP Session :- Interface

Container creates a session-id for each user
— The Container uses this id to identify the user.

→ HTTPServletRequest interface :-

1) public HttpSession getSession() :-

Return the current session associated with request.
If request does not have a session it will create one.

2) public HttpSession getSession(boolean create)

public void invalidate() ⇒ Invalidate the session & unbinds any object bound to it.

— To set attribute in session scope we have setAttribute().

→ HttpSessionDemo (3.0)

index.html →

login.html →

→ index.html

```
<head>
<title> welcome </>
</head>
<body>
<a href = "login.html"> login</a>
<a href = "ProfileServlet"> profile</a>
<a href = "LogoutServlet">Logout</a>
</body>
</html>
```

→ Login.html

Enter username [ ]
Enter password [ ]
[Submit]

→ LoginServlet.java

```
Protected Void doGet( ——— ) throws ——— {
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();

request.getRequestDispatcher("index.html").include(request,response);
    String uname = request.getparameter("txtUsername");
    String pwd = request.getParameter("txtPassword");
```

```
if(uname.equals("admin") && pwd.equals("adminses")) {
    HttpSession Session = request.getSession();
        Session.setAttribute("user", uname);
        ——————("pass", pwd);
        pw.write("Successfully Logged In");
} else {
    pw.write("Invalid username & password !!!");
request.getRequestDispatcher("Login.html").include(request, response);
}
```

→ profileServlet.java.

```
doGet method {
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    request.getRequestDispatcher("index.html").include(request,
                                                response);
HttpSession Session = request.getSession(false);
    if(Session!=null) {
        pw.write("Profile:");
        pw.write("username = "+Session.getAttribute("user"));
        pw.write("Password =" + Session.getAttribute("pass"));
    } else {
        pw.write("Session expired");
    request.getRequestDispatcher("Login.html").include
                                        (request, response);
}
```

(if Session is already there it will not create new one)

→ Logout.java.

```
doGet method {
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    HttpSession session = request.getSession();
    session.invalidate(); // unbinded all objects that is annotated
    pw.write("Logout!!!");
}
```

Cookies

1) Persistent cookies → chrom चालू केला email login browser बंद केला तरी email login राहिल.
2) Non-persistant cookies → chrom चालू केला email login browser बंद केला तर gmail Logout होईल.

- Textual info → text & value.
- Stored at client side (browser)

cookies tracks user References

javax.servlet.http.Cookie class

1) Cookie() → Construct a cookie.
2) Cookie(String name, String value) - with name & value.
   (name → key, value → value)

- public void setMaxAge(int expiry) → Sets the max. Age of cookie in seconds
- public void setName(String name)
- public String getName()
- public void setValue(String value)
- public String getValue().
- Public void addCookie(cookie ck) → HttpServletResponse: used to add cookie to response object

How many Cookies Per application allowed?
→ Can support Cookies as Large as 4096 bytes in size

public Cookie[] getCookies() → HttpServletRequest – return all cookies from browser

→ CookieDemo (3.0)

→ index.html

```
<form action = "serv1" method = "get">
        Enter firstName [          ]
            [Enter]
```

→ Servlet1.java

```
doGet method {
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
String fname = request.getParameter("txtfirstName");
// create a cookie
Cookie c = new Cookie("fname", fname);
// set age of cookie
  c.setMaxAge(30);
// Add Cookie to response.
  response.addCookie(c);
pw.write("<center><form action='serve' method='get'>");
pw.write("Enter a lastname : <input type = 'text' name =
                                'txtlastName'>");
pw.write("<br><br><input type = 'Submit' Value= 'GO'>");
pw.write("</form></center>");
}
```

→ Servlet2.java

```
doGet method {
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
String lname = request.getParameter("txtlastname");
Cookie[] cookie = request.getCookies();
```

```
String fnm = "";
if(cookie != null){
    fnm = cookie[0].getValue();
    pw.write("<centre><h1>firstName = "+fnm);
    pw.write("<br>Lastname = "+lname);
    pw.write("</h1></centre>");
}else{
    pw.write("<centre><h1>Session Expired");
    pw.write("</h1></centre>");
}
```

create university Result Appln:-
Subjects
PRN Name S1 S2 S3
on display page → Total marks Percentage
Grade
(Assignment monday submission)

mysql 8.0
mysql-connecter-java-8.0.28.jar

→ Login Registration Demo (3.0)
⇒ 2 Jar files ⇒ lib ⇒ mysql-connecter-java.jar
Add ⇒ Servlet-Api.jar

It is also known as type IV driver

Com.app.controller → LoginServlet. Registration
Com.app.dao → EmployeeDao
Com.app.model → Employee.java

* Connection → Interface → java.sql.Connection,
→ Connecting to DB ⇒
1) Register the driver → com.mysql.jdbc.Driver
2) Establish a connection → "jdbc:mysql://localhost:3306/db", "root", "root";
3) Create a statement.

Port No DB, Database Number

executeUpdate() → will fire query of database.

→ Regiske.html

```
<body><hL> Employee Registration form </>
<form action = "RegistrationServlet" method = "post">
<table>
<tr>
<td> Enter Employee Name: </td>
<td><input type = "text" name = "empname"></>
</tr>
<tr>
<td> Enter a password : </>
<td><input type = "password" name = "emppwd"></>
</tr>
<tr>
<td> Enter email : </>
<td> < input type = "email" name = "empemail"></>
</tr>
<tr>
<td> Enter a phoneNo : </>
<td> <input type = "number" name = "empphNo" required = "required"></>
</tr>
<tr>
<td><input type = "submit" value = "Register"></>
</tr>
</table>
</body>
</html>
```

→ index.html

```
<title> Login </>
<script type = "text/javascript">
    function callServlet() {
        var servletName = document.empform.register.value;
        if( servletName == "" ) {
            alert("No value ...");
```

(servletName: variable name; document.empform: blank page in JavaScript; register: name)

```
        return false;
    }else{
        alert("Servlet Name = " + ServletName);
        document.location.href = ServletName + ".html";
        return false;
    }
}
</script>
</head>
<body>
<h1>Employee login form</h>
<form action = "LoginServlet" method = "post" name = "empform">
<table>
<tr>
<td> Enter username : </td>
<td> <input type = "text" name = "txtUsername"> </td>
</tr>
<tr>
<td> Enter password : </td>
<td> input type = "text" name = "txtPassword"></td>
</tr>
<td> <input type = "button" value = "register" onclick = "return callServlet();"
name = "register"></td>
<td> <input type = "submit" value = "login"> </td>
</tr>
</table>
</form>
</body>
```

→ Employee.java (com.app.model)

```
public class Employee implements Serializable {
    private int empId;
    private String empName, empPwd, empEmail, empPhno;
    // Generate getters & Setters.
```

→ com.app.dao EmployeeDao.java

```
import java.sql.*;
import com.app.model.Employee;
public class EmployeeDao {
    public static Connection getConnection() throws SQLException {
        Connection con = Null;
        try {
            // 1) Register the driver
            Class.forName("com.mysql.jdbc.Driver");
            // 2) Establish a Connection.
            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/db","root","Ameya10k");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return con;
    }
    public static int empAdd(Employee e) {
        int i = 0;
        try {
            Connection con = EmployeeDao.getConnection();
            3) create a Stmt.
        preparedStatement stmt = con.preparedStatement("insert into
            emp1 (empName, empPwd, empEmail, empPhno)" +
                "Values (?,?,?,?)");
```

class → Final class
implements
java.io.Serializable

Connection → Interface
DriverManager → Class
Connection → interface

```
            Stmt.setString(1, e.getEmpName());
            Stmt.setString(2, e.getEmpPwd());
            Stmt.setString(3, e.getEmpEmail());
            Stmt.setString(4, e.getEmpPhno());
            i = Stmt.executeUpdate(); // will fire query for DB.
        } catch (SQLException el) {
            el.printStackTrace;
        }
        return i; // { 1 row affected / 2 rows affected } ला count येईल i मध्ये
    }
    public static boolean Validate(Employee e) throws SQLException {
        Connection Con = EmployeeDao.getConnection();
        PrepareStatement Stmt = Con.prepareStatement("select * from
            empl where empName = ? and empPwd = ?");
        Stmt.setString(1, e.getEmpName());
        Stmt.setString(2, e.getEmpPwd());
        Resultset rs = Stmt.executeQuery();
        boolean s = rs.next();
        return s;
    }
}
```

→ LoginServlet.java

```
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    doGet method {
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
```

```
String name = request.getparameter("txtUsername");
String pwd = request.getparameter("txtPassword");
Employee e = new Employee();
e.setEmpName(name);
e.setEmpPwd(pwd);
boolean status = false;
try {
    status = EmployeeDao.Validate(e);
} catch (SQLException el) {
    el.printStackTrace();
} if(status == true) {
    pw.write("Login Successfull!!!");
    request.getRequestDispatcher("index.html").include(request,
                                                response);
} else {
    pw.write("Invalid Username or password!!");
    request.getRequestDispatcher("index.html").include(
                                    request, response);
}
}
```

→ RegistrationServlet.java

```
@WebServlet("/RegistrationServlet")
public class RegistrationServlet extends HttpServlet {
    doGet method {
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
```

```
String name = request.getParameter("empName");
String pwd = request.getparameter("empPwd");
String email = request.getParameter("empEmail");
String phno = request.getParameter("empphno");

Employee e = new Employee();
e.setEmpName(name);
e.setEmpPwd(pwd);
e.SetEmpEmail(email);
e.SetEmpphno(phno);
int ans = 0;
ans = EmployeeDao.empAdd(e);
if(ans>0){
  pw.write("Registration Done!!!");
  request.getRequestDispatcher("index.html").include(request,response);
}else{
  pw.write("Registration failed!!!");
  request.getRequestDispatcher("registe.html").include(request,response);
}
}
```

* JSP (Java Server Page) :- (Internally Converted to Servlet
→ .jsp Extension → Implicit objects.

→ Java technology.
→ that generate dynamic web pages on HTML.

{JSTt → JSP Server tag language.}

{ Why underscore at
-jspService() }

→ Java code embedded in HTML code.
→ Separate presentation and Business Logic.
→ 80% HTML & 20% Java code
→ Expression language & JSTL.
(EL)

* JSP API ⇒ 1) javax.servlet.jsp
2) javax.servlet.jsp.tagext

* JSP Lifecycle ⇒ hello.jsp

* 7 phases ⇒ 1) Translation → HelloServlet.java.
2) Compilation → helloServlet.class.
3) Loading.
4) ~~Instantiation~~
5) Initialization → jspInit() ⇒ called once throughout Lifecycle of Servlet.
6) Service → _jspService()
7) Destruction → jspDestroy()

Assignment.
ques) use HttpSession in Air ticket Reservation System
Refer Blackbook.
ques) Library management system Using Servlets & deployement to tomcat.

* Three types of predefined tags :-
  1) Directives.
  2) Scripting element.
  3) Actions.

* JSP Elements :-

  <!---------> Html Comment.

  <%--------%> JSP Comment.

* JSP Scripting tag :- Embedded Java Code in JSP.
  or elements :-
  1) Scriplet Tag :- To write business Logic in service()

     <%-----%> Internally goes to

  2) Expression Tag :- To print on Browser.

     <%= ----- %>

  3) Declaration tag :- Declare global variables & methods.

     <%! ------ %>

→ JSPAPP (3.0)

⇒ 2 Jar files ⇒ jsp-api.jar
⇒ Servlet-Api.jar

create jsp ⇒ web-Content → new → create JSP → index.jsp

{ jsp name & welcome file tag in web.xml Should be same }

Date class present In which packages?
2 Date classes in 2 diff package?
1) java.sql.Date()
2) java.util.Date()
& one more Simple Date class for formatting & parsing.

→ index.jsp

```
<title> first </>
<body>
<h3> Hello welcome to jsp !!!!</h3>
<br>
Time on server is : <%= new java.util.Date() %>
</body>
</html>
```

→ first.jsp

```
<body>
<%!
    int count = 0;
%>
<%
    count++; %>
<%= "Visitor count = " + count %>
</body>
```

→ form.html

Student Registration form

Enter UserName [ ]

Enter Password [ ]

[Submit]

```
<form action = "second.jsp" method = "get">
```

→ second.jsp

```
<body>
<%!
    int num = 100;
    public String toLower(String str) {
        return str.toLowerCase();
    }
%>
```

```
<%
    String name = request.getParameter("txtUsername");
    String pwd  = request.getParameter("txtPassword");
    out.write("Name = " + name + " password = " + pwd);
%>
```

{implicit objects are directly we can use in JSP}

```
<%-- <%= "ToLoweeCase = " + toLower("Hello") %> --%>
</body>
</html>
```

→ table.jsp

```
<body>
<form action = "third.jsp" method = "get">
<table>
<tr>
<td> Enter any Number : </td>
<td> input type = "number" name = "txtNum" > </>
</tr>
<tr>
<td><input type = "submit" value = "SUBMIT" ></>
</>
</table>
</form>
</html>
```

→ third.jsp

```
<body>
<%
  int num = Integer.parseInt(request.getParameter("txtNum"));
%>
<table border = "1">
<%
   for (int i=1; i<=10; i++){
%>
<tr>
<td><%= (num*i) %></td>
</tr>
```

```
<%
 }
 %>
</table>
</body>
</html>
```

→ Expression-test.jsp

```
<Html> Test </title>
</head>
<body>
Converting string to uppercase: <%= new String("rahul").toUpperCase() %>
<br>
15 multiplied by 4 is: <% (15*4) %>
<br>
67 greater than 78: <% (67>78) %>
</body>
</html>
```

* JSP Implicit objects:-

object	type	Purpose
1) out	JspWriter	Outputstream
2) request	HttpServletRequest	Access the details of request.
3) response	HttpServletresponse	Access the details of Response.
4) session	Httpsession	To handle Http Session.
5) Applin	ServletContext	Reffers to web-app
6) Config	ServletConfig	ServletConfig Information.
7) page	object	the page itself
8) Pagecontext	Pagecontext	Reffers to page environment
8) Exception	Throwable	Useful for error handling

→ ImplicitDemo (3.0)

Language used by browser is en_US.

→ index.jsp

```
<body>
Request-user Agent : <%= request.getHeader("User-Agent") %>
<br>
Request Language : <%= request.getLocale()  %>
</body>
```

→ first.jsp

Enter Num1 : [ ]

Enter Num2 : [ ]

DIVISION

```
<form action = "second.jsp" method = "get">
```

→ second.jsp

```
<%@ page ------ errorPage = "error.jsp" %>
```

Page directive

```
<body>
<%
  int n1 = Integer.ParseInt(request.getParameter("txtNum1");
  int n2 = Integer.Parseint(request.getParameter("txtNum2");
  int c = n1/n2;
  Out.write("Division = " + c);
%>
</body>
</html>
```

→ error.jsp

<%@ Page ——— ——— isErrorPage = "true" %>

↑ {By Default it is false.}

<body>
<h1> Exception occured !!!
<%= Exception %>
</h1>
</body>
</html>

→ JSPApp1 (3.0)

What is include directive & include action tag?

include directive → Static data

Diff. asked In Interviews

include action tag → dynamic page request.

→ Action Tags :

1) jsp: forward → forward request and response object to another resource.

2) jsp: include → include the content of another resource at the time of page request.

3) jsp: param → sets the parameter value

→ ActionTagDemo (3.0)

→ index.jsp

```
<body>
<jsp: forward page = "first.jsp">
<jsp: param value = "Rahul" name = "name" />
</jsp: forward>
</body>
</html>
```

→ header.jsp

```
<body>
<h1 align = "centre">
JSP tutorial !!!
</h1>
```

→ footer.jsp

```
<%@ page ------ import = "java.util.*" %>
<body>
<h1 align = "centre">
Last updated : <%= new Date() %>
</h1>
```

→ first.jsp

```
<body>
<%@ include file = "header.jsp" %>
Name = <%= request.getParameter("name") %>
<br>
<jsp:include page = "footer.jsp" />
</body>
</html>
```

Static import JSP tutorial msg change होत नाही म्हणून include directive use केला

dynamic import Date() change होणार time पण change होणार म्हणून Runtime वर Hence we used include action tag.

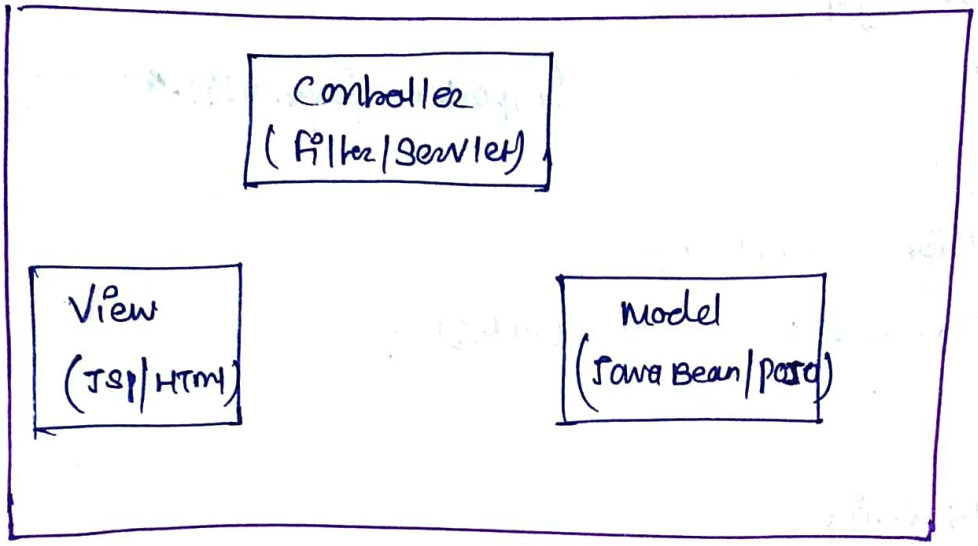
include directive	include action tag
1) <%@ include file = footer.jsp" %>	1) <jsp: include page = "footer.jsp" />
2) Static binding	2) Dynamic binding
3) known as file include	3) known as page include.
4) Prefered when include page does not change often.	4) Prefered when include page changes often
5) It is a static import.	5) It is dynamic import.
6) loaded at Compilertime.	6) loaded at Runtime.
7) faster	7) Slower

* Why layered appln Design?

→ Seperate the business logic and presentation.

→ changes in business logic should not affect the presentation layer and vv.

Soln ⇒ MVC (Model View Controller)



Model → Container

{ Java Bean
Java class }

Rules ⇒ 1) No-arg constructor.
2) It should be Serializable.
3) getters and setters.

<jsp: useBean id = "instanceName"
scope = "page/request/session/application"
class = "fully qualified name"></jsp: useBean>

* jsp: useBean action Tag ⇒ use to locate & instantiate bean class.

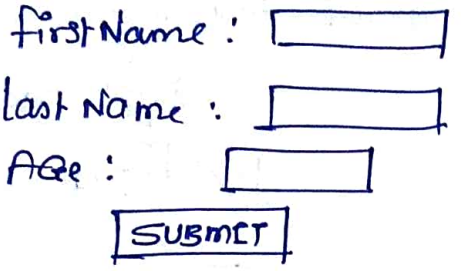
* jsp: setProperty : sets a property value or values in bean using the setter method.

* jsp: getProperty : returns the value of the property.

→ JSPMVC Demo (3.0)

right click on User
→ copy qualified name
it will show like after
pasting com.app.User.

→ index.jsp




```
<form action = "welcome.jsp" method = "get">
```

→ User.java

```
Package com.app;
import java.io.Serializable;
public class User implements Serializable {
    private String firstName, lastName, age;
    public User() { }
  // Generate getter & Setter.
```

→ welcome.jsp

```
<body>
<jsp:usebean id = "u" class = "Com.app.User"/>
<!-- User u = new User(); -->
<jsp:setProperty property = "*" name = "u"/>
```

(* → सगळ्या fields set कुलयात)

User Details are as follows :-

```
FirstName : <jsp:getProperty property = "firstName" name = "u"/>
<br>
LastName : <jsp:getProperty property = "LastName" name = "u"/>
<br>
Age : <jsp.getproperty = "Age" name = "u"/>
</body>
</html>
```

* Web-Inf → right click → file → data.tld or message.tld extension must be .tld.

→ TaglibDemo (3.0)

→ To create a Custom tag we have to extend a class SimpleTagSupport.

* right click on Source class → Source → override implements methods

SimpleTagSupport.

V doTag()

{ manually write → in web.xml
file < welcome-file >
three tagdemo.jsp . }

toString is method of object class not a String class.

<%@ page ——— <%@ taglib prefix = " " uri = " " %>

* { copy tld file qualified name & paste in URI }

* Taglib directive ⇒

Custom tag
3 Components

1) *.jsp (Tag library with jsp page) ⇒ which needs functionality

2) *.java (Tag Handler class file) ⇒ which implements the functionality required in JSP.

3) *.tld (Tag library Descriptor) ⇒ which acts as a link bet JSP and Java Code.

{ 1) create a tld file
2) tld file specify tld version & jsp version
3) Implementation → }

→ message.tld

```
<taglib>
  <tlib-version> 1.0 </tlib.version>
  <jsp-version> 2.0 </jsp-version>
  <short-name> My Custom tag </short-name>
  <tag>
  <name> MyMsg </name>
  <tag-class> mypack.Details </tag-class>
  <body-content> Empty </body-content>
  </tag>
  <tag>
  <name> upper </name>
  <tag-class> mypack.Convert </tag-class>
  <body-content> Scriptless </body-content>
  </tag>
</taglib>
```

short Name given to access tags.

→ tagdemo.jsp

```
<%@ taglib uri = "WEB-INF/message.tld" prefix = "myprefix" %>
</head>
<body>
<myprefix : MyMsg />          → tag
<myprefix : upper> Mayuri Java trainer </ myprefix : upper>
                                → JSP Body
</body>
```

→ Details.java

```
package mypack;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class Details extends SimpleTagSupport {
    @override
    public void dotag() throws JspException, IOException {
```

```
        super.doTag();
        JspWriter out = getJspContext().getOut();
        out.println("this is my custom Tag!!!");
    }
}
```

→ Convert.java

```
package mypack;
public class Convert extends SimpleTagSupport {
    StringWriter sw = new StringWriter();
    @Override
    public void doTag() throws JspException, IOException {
        super.doTag();
        getJspBody().invoke(sw);
        JspWriter out = getJspContext().getOut();
        out.println(sw.toString().toUpperCase());
    }
}
```

Can main method be overloaded?
→ yes.

Why main method is static?
Diff. final, finally, finalize

Why String immutable?
Why preparedstatmt, connection all are interface & not class?

3 String parameters in getConnection why?

notify, wait, notifyall used in multi-threading but they are methods of object class. Why?

* JSTL ⇒

{ mvnRepository.com
To download JSTL Jarfile.
JSTL Jar 1.2 }

→ JSTLDemo (3.0)

for URI ⇒ Go to Java Resources → libraries → web app libraries
↓
expand ← META-INF ← expand it ← JSTL 1.2 Jar
↓
c.tld → copy URI from this file <URI> "http://java.---
↓ ↓ ---"</URI> line No. 12
core taglib directive

→ index.jsp

```
<%@ ------ import = "java.util.*" %>
<%@ taglib URI = "http://java.sun.com/jsp/jsH/core" prefix = "c" %>
<body>
<c:out Value = "${'welcome to infoway'}"></c:out>
<br> <br>
<c:set Var = "date" Value = "<%= new Date() %>"></c:set>
Time on Server : ${date}
</body>
```

→ Student.java

```
public class Student {
    private String firstName;
    private String lastName;
    private Boolean goldStudent;
    // Generate getters & Setters
    // Generate to String,
    // Generate para.Constructor.
```

→ Students-info.jsp

```
<%@ page import = "java.util.*, com.app.Student" %>
<%@ taglib uri = "http://java.sun.com/jsp/jsH/core" prefix = "c" %>
```

```
<%
    List<Student> data = new ArrayList<Student>();
      data.add(new Student("Rahul", "Paneer", false));
      data.add(new Student("Amaylak", "Lak", false));
      data.add(new Student("Yashi", "patel", True);
    pageContext.setAttribute("myStudents", data); // Refer page env.
%>
<body>
<table border = "1">
<tr>
<th> firstName </th>
<th> LastName </>
<th> GoldStudent </>
</tr>

<c:forEach Var = "temp" items = "${myStudents}" >
<tr>
 <td> ${temp.firstName} </>
<td> ${ temp.LastName } </>
     <td>

<c:if test = "${temp.goldstudent}" >
Gold Student.
</c:if>

<c:if test = "${not temp.goldStudent}">
-
</c:if>
 </td>
 </tr>
</c:forEach>
</table>
</body>
</Html>
```

(There is no condn else in JSTL)

→ student-info1.jsp

Everything same as student-info.jsp.

```
<td>
<c:choose>
<c:when test="${temp.gold student}">
Gold student.
</c:when>
<c:otherwise>
NA
</c:otherwise>
</c:choose>
</td>
</tr>
</c:forEach>
</table></body></html>
```

replace if-else with this.

→ JSPCRUDDemo.jsp

→ create table emp(id integer auto-increment,
name Varchar(30),
password Varchar(50),
email Varchar(50), gender Varchar(30),
Country Varchar(30), primary key (id));

→ Jar files req → jsp-api.jar
→ JSH-1.2.jar
→ Servlet-api.jar
→ mysql-connector.jar

employee model field name & emp form.htm names should be same

→ index.jsp

```
<%@ page ------
<body>
<!- CREATE READ UPDATE DELETE ->
<h1> CRUD </h1>
<hr>
<a href="addEmployee.jsp"> Add Employee </a>
<a href="ViewEmployee.jsp"> View Employe </a>
</body>
</html>
```

→ addEmployee.jsp

```
<body>
<jsp:include page="employeeform.html"></jsp:include>
</body>
</html>
```

→ employeeform.html

```
<body>
<a href="ViewEmployees.jsp"> View employees </a>
<hr>
<form action="addEmp.jsp" method="post">
<table>
<tr>
```

Enter Name : [ ]

Enter password : [ ]

Enter Email-Id : [ ]

Select Gender : Omale Ofemale

Select Country : [India ▽]

[add Employee]

→ addEmp.jsp

```
<%@ page import = "com.app.dao.EmployeeDao" %>
<body>
<jsp:useBean id = "emp" class = "com.app.bean.Employee"></>
<jsp:setProperty property = "*" name = "emp"/>
<%
    int i = EmployeeDao.saveEmp(emp);
    if(i>0)
        response.sendRedirect("success.jsp");
    else
        response.sendRedirect("error.jsp");
%>
</body></html>
```

→ Employee.java

```
package com.app.bean;
public class Employee implements Serializable {
    private int id;
    private String name, password, email, gender, country;
    // Generate getter & setter, toString
}
```

→ Success.jsp

```
<body>
Employee Record Saved Successfully !!!
<jsp:include page = "Employeeform.html"></>
</body>
</html>
```

→ ViewEmployees.jsp

```
<%@page import="com.app.dao.EmployeeDao, com.app.bean.Employee
,java.util.*"%>
<%@taglib uri="http://java.sun.com/jsp/jsH/core" prefix="c"%>
<h1> Employees</h1>
<hr>
<% List<Employee> list = EmployeeDao.getAllEmployees();
request.setAttribute("list", list);
%>
<table border="1" width="100%">
<tr>
<th> ID</th><th> Name </th><th> Password</th><th>Email</th>
<th>Gender</th><th> Country</th><th> Edit </th><th> Delete </th>
</tr>
<c:forEach var="e" items="${list}">
<tr>
<td> ${e.getId()}</td>
<td> ${e.getName()}</>
<td>${e.getPassword()}</>
<td> ${e.getEmail()}</>
<td> ${e.getGender()}</td>
<td> ${e.getCountry()}</td>
<td> <a href="editEmployee.jsp?id=${e.getId()}">Edit</a></td>
<td> <a heef="deleteEmployee.jsp?id=${e.getId()}"> Delete</a></td>
</tr>
</c:forEach>
</table>
</body>
```

→ editEmployee.jsp

```
<%@ page import = "com.app.dao.EmployeeDao, com.app.bean.Employee" %>
<%@ taglib uri = "http://java.sun.com/jsp/jsH/core" prefix = "c" %>
<body>
<% int id = Integer.parseInt(request.getParameter("id"));
   Employee e = EmployeeDao.getEmployeeId(id);
%>
<h1> Edit Employee </h1>
<hr>
<form action = "editEmp.jsp" method = "post">
<input type = "hidden" name = "id" value = "<%= e.getId() %>">
<table>
<tr>
<td> Name : </td>
<td> <input type = "text" name = "name" value = "<%= e.getName() %>
</td>

</tr>
<tr>
<td> Password : </td>
<td> <input type = "password" name = "password" value = "<%=
                                    e.getPassword() %>">.
</td>
</tr>
<tr>
<td> Email </td>
<td> <input type = "email" name = "email" value = <%= e.getEmail() %>"
</td>
</tr>
<tr>
<td> Gender : </td>
<td>
<%
 if(e.getGender().equals("male"))
 {
%>
```

```
<input type="radio" name="gender" value="male" CHECKED>Male
<%
}
else
{
%>
<input type="radio" name="gender" value="male">male
<%
}
%>
<%
if(e.getGender().equals("female"))
{
%>
<input type="radio" name="gender" value="female" CHECKED>female
<%
}
else
{
%>
<input type="radio" name="gender" value="female">female
<%
}
%>
</td>
</tr>
<tr>
<td>Country:</td>
<td>
<Select name="Country" Selected=${e.getCountry()}>
<%
if(e.getCountry().equals("India"))
{
%>
```

```
<option selected = "selected" > India </option>
<%
}
else
{
%>
<option > India </option>
<%
}
%>
<%
if(e.getCountry().equals("USA")
{%>
<option selected = "selected" > Usa </option
<%
}
else
{%>
<option > USA </option >
<%
}
%>
<%
if(e.getCountry().equals("France"))
{%>
<option selected = "selected" > France </option >
<%
}else {
%>
<option > france </option >
<% } %>
```

```
<%
if(e.getCountry().equals("Germany"))
{%>
<option selected="selected">Germany</option>
<%
}else{
%>
<option>Germany</>
<%
}
%>
<option>Other</>
</select>
</td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Edit"></td>
</tr>
</table>
</form>
</body></html>
```

→ editEmp.jsp

```
<%@ page="Comm.app.dao.empDao" %>
<body>
<jsp:useBean id="emp" class="Com.app.bean.Employee"></>
<jsp:setProperty property="*" name="emp"/>
<%
    int i = EmployeeDao.updateEmployee(emp);
    if(i>0){
        response.sendRedirect("viewEmployees.jsp");
    else
        response.sendRedirect("error.jsp");
%></body></html>
```

→ EmployeeDao.jspro

```
package com.app.dao;
// write getconnection method
public static int saveEmp(Employee e) throws SqlException {
    Connection conn = getConnection();
PreparedStatement stmt = Conn.prepareStatement("insert into "+
        " emp(name, password, email, gender, Country)" +
            " Value (?,?,?,?,?)");

    Stmt.setString(1, e.getName());
    Stmt.setString(2, e.getPassword());
    Stmt.setString(3, e.getEmail());
    Stmt.setString(4, e.getGender());
    ————————(5, e.getCountry());
    int status = Stmt.executeUpdate();
    return Status;
} public static List<Employee> getAllEmployees() throw ——— {
        List<Employee> l = new ArrayList<Employee>();
    Connection conn = getConnection();
    PreparedStatement Stmt = Con.prepareStatement("select * from
                                                    emp");
    ResultSet rs = Stmt.executeQuery();
    While(rs.next()) {
        Employee e = new Employee();
        e.setId(rs.getInt(1));
        e.setName(rs.getString(2));
```

```
        e.setPassword(rs.getString(3));
        e.setEmail(rs.getString(4));
        e.setGender(rs.getString(5));
        e.setCountry(rs.getString(6));
        l.add(e);
    }
    return l;
}
public static Employee getEmployeeId(int id) throws SQLException {
        Employee e = null;
        Connection con = getConnection();
        PreparedStatement stmt = con.prepareStatement("select * from emp
                                 where id = ?");
        stmt.setInt(1, id);
        ResultSet rs = stmt.executeQuery();
        while(rs.next()) {
            e=new Employee();
            e.setId(rs.getInt(1));
            e.setName(rs.getString(2));
            e.setPassword(rs.getString(3));
            e.setEmail(rs.getString(4));
            e.setGender(rs.getString(5));
            e.setCountry(rs.getString(6));
        }
    return e;
}
```

```
Public static int updateEmployee(Employee e) throws ------{
    Connection con = getConnection();
PreparedStatement stmt = con.prepareStatement("update emp set
    name=?, password=?," + "email=?, gender=?,
    country=? where id=?");

    stmt.setString(1, e.getName());
------(2, e.getPassword());
------(3, e.getEmail());
------(4, e.getGender());
------(5, e.getCountry());
~~stmt.setInt~~(6, e.getId());
    int i = stmt.executeUpdate();
    return i;
}

public static int delete(Employee e) throws ------{
    // Lab work
```

→ error.jsp

```
<body>
FAILED!!!
<jsp:include page = "employeeform.html"></>
</body></html>
```

→ deleteEmployee.jsp

```
<body>
<!--
    get ID from request object
    parse it
    call deleteEmployee from DAO.
    Test if i>0 → View Employees.jsp
    else → error.jsp
-->
</body></html>
```

→ JSTL (Jsp std. tag library) :-

Set a tags Simplifies the JSP developement.
fast developement.
No use of Scriplet tag

* Types of JSTL tags :-
  1) Core tags.
  2) function Tags
  3) formatting Tags

} & many more

JSTL.jar

→ todo-demo.jsp

```
<%@page import = "java.util.*"
    session = "true" autoflush = "true" %>

<body>
<!-- 1. Create a Html form -->
<form action = "todo-demo.jsp">
Add new Item: <input type = "text" name = "theItem">
<input type = "Sumit" value = "SUBMIT">
<!-- 2. Add a new Item to TODO list -->
<%
  //Get the todo items from the session
List<String> items = (List<String>) session.
                getAttribute("myTODOList");

//If todo items doesn't exist create a new one
  if (items == null) {
     items = new ArrayList<String>();
    session.setAttribute("myTODOList", items);
  }
```

```
// See if there is form data to add
String theItem = request.getParameter("
if(theItem != "" && theItem != null){
        items.add(theItem);
}
%>
<!--Steps: Display all the TODO items from session
<hr>
<b> To List items</b><br>
<ol>
<%
    for(String temp : items){
        out.println("<li>"+temp+"</li>");
    }
%>
</ol>
</form></body></html>
```

* Hibernate *

→ Hibernate.Org Latest Version ⇒ 6.4
↓
Version 5.4 down load for jdk=8 for Java 11, 17, 21

→ MVNRepository.com
↓
for dependancies down load

→ File → new → MAVEN Project → next
→ Catalog select (Internal) → maven-archtyp-quickstart Version 1.1 select
{configuration file ⇒ xml file (Pom.xml)}

GroupId → Com.app → package Name .
ArtifactId → projectName → Hibernate

→ search on MVNRepository.com i.e. hibernate
Core dependancy → hibernate core → 5.5.4 final
Relocation

{ mvn Repository folder → M2 folder } → C:\user>admn>.me

→ search → mysql connector/J → 8.2.0
. Put this dependancy in dependencies tag.

→ src/main/java → right click → new → other
→ xml file → next → hibernate.cfg
↓
Configuration

Maven Depencies → expand hibernate core
org.hibernate → hibernate-configuration 3.0.
expand

→ copy line 10 to 12 & paste in hibernate.cfg

{ hbm2ddl → Hibernate to data defination lang
create → create a new table
update → create a table & then update

{ dialect → to map your data type to table types
fname String ⟶ fname Varchar.
age int ⟶ age integer.

→ Hibernate.Demo

{ xml file → property → show_sql it will show all fired queries on console }

→ student.java

```
import javax.persistence.Column;
@Entity
@Table(name = "Student1es")
public class Student{
    @Id → primary key given
    @GeneratedValue(Strategy = GenerationType.AUTO)
    private int id;
    @Column(name = "fname")
    private String firstName;
```

2 table will create 1) student1es 2) hibernate sequence

→ App.java

```
pubife elan APP {
    main method {
1) // Create SessionFactory
SessionFactory factory = new Configuration().
                Configure("hibeernate.cfg.xml").
                addAnnotatedclass (student.class).
                buildsessionfactory();

2) // Create Session
   Session session = factory.opensession();

3) Begin transaction
     session.beginTransaction();

4) Peform Operations
   // create student object
     Student s = new student();
      s.setFirstName("Pratik");
      s.setLastName("patel");
      s.setAge(25);
      session.save(s);
```

Get Student - Id

```
int id=5;
Student s = Session.get(Student.class,id);
Sysout(s);
```

// update student

```
int id=3;
Student s2 Session.get(Student.class,id);
s.setFirstName("Mansi");
s.setLastName("Rajput");
```

// Delete Student

```
int id=2;
Student s2 session.get(Student.class.id);
session.delete(s);
```

// List of Students

```
Query query = Session.createQuery("from Student");
List<Student> l = query.list();
for(Student s:l){
    Sysout(s);
}
```

// HQL - Hibernate Query Language

```
List<Student> l = session.createQuery("from
    s where s.lastname='patel'").getRes

    for(Student s : l) {
        Sysout(s);
    }
    Sysout("Object Persisted !!!");
    Session.getTransaction.Commit();
    Session.closed();
    factory.close();
}
}
```

---

* If we use GenerationType.AUTO we can use property hbm2DDL.auto file create or update. create will create a table & update will create table if not exist or if exist it will update & 2 tables will create in DB one more hibernatesequence will also create will show next id to be insert & In query fired by hibernate id is not auto-increment it will just take one sequence.

If we use GenerationType.IDENTITY we can
use property hbm2DDL.auto as create or
update. create will create table & update will
just update the values it will not create a table
this is diff. & Here hibernate_Sequence table
will not create & here query fired by hibernate
the id will be auto-incremented.

File → New Java project

↓

create single Java project.

↓

right click on project.

↓

configure

↓

convert to maven project.

→ Jar files → Java Archieves.

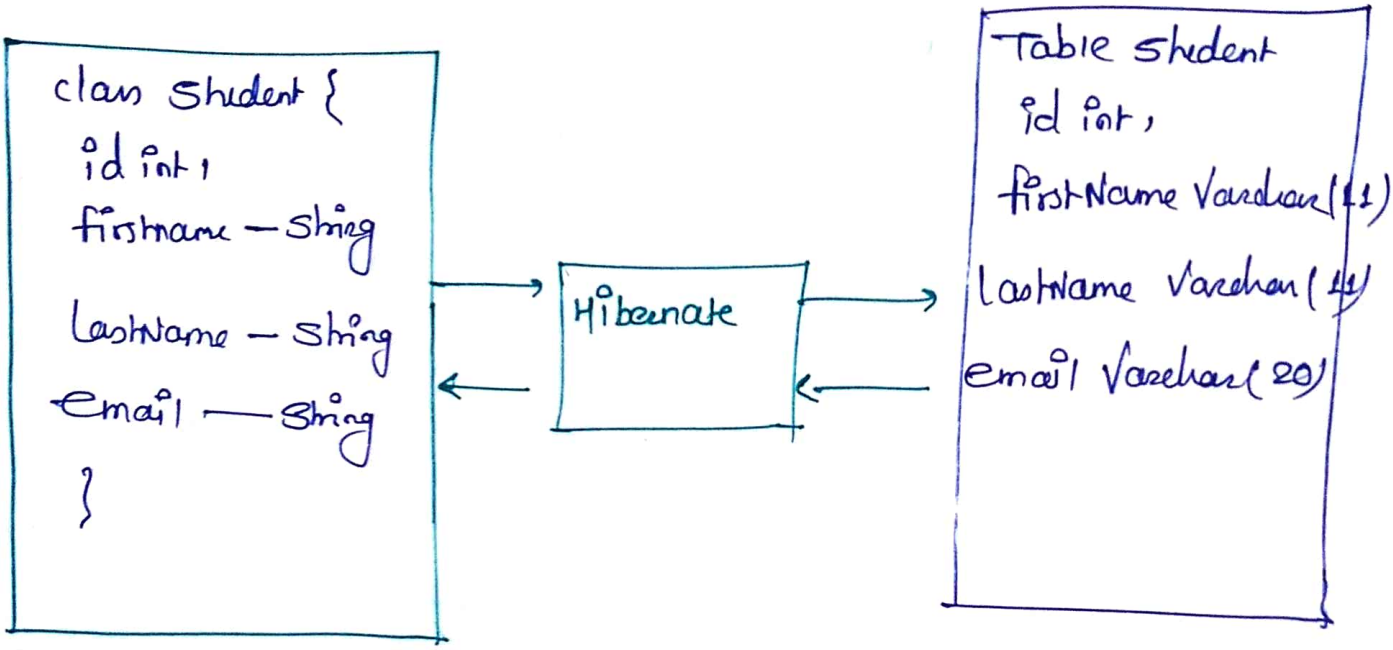
↓

to add libraries.

Which has packages & all.

Hibernate :-

→ framework for persisting java objects in a DB.
→ Minimize the amount of JDBC code.
→ Handles all low level SQL.
→ Internally uses JDBC API.
→ ORM (Object Relational mapping)
→ Map java class and DB Table.



JPA (Java Peesistence APIs) :-

→ Std. Specification.
→ Hibernate is an implementation of JPA specification.

{ javax.persistence package → JPA interfaces & classes }

→ hibernateWebApp (3.0)
create dynamic web project.
then convert it to maven project.

dependencies to add { Jar → mysql connector.
→ hibernate core ~~reallocation~~ 5.5.4
→ JSP API → 2.0

dao layer → DB related code

→ index.jsp

```
<body>
<form action = "register.jsp" method = "post">
<table>
<tr>
<td> Enter Username : </td>
<td> <input type = "text" name = "name" ></td>
```

Enter name [ ]

Enter password [ ] Enter email [ ]

Submit

→ Employee.java

```
package com.app.bean;
import java.io.Serializable;
@Entity
@Table(name ="emp456")
public class Employee implements Serializable {
    private String name, password, email;
    // Generate getters & setters, toString.
    @Id
    @GeneratedValue(Strategy.Generated.Identity)
    private int id;
    @Column(name = "EmpName")
    private String name;
    @Column(nam = "empPassword")
    private String password;
    @Column(name = "Empemail")
    private String email;
```

→ hibernate.cfg.xml.

```
<mapping class = "com.app.bean.Employee" />
```

→ register.jsp

```
<%@ page import = "com.app.dao.EmployeeDAO" %>
  <body>
  <jsp:useBean id = "emp" class = "com.app.beans.Employee" />
  <jsp:setProperty property = "*" name = "emp" />
  <%
     int i = EmployeeDAO.registerEmp(emp);
     if(i>0) {
        response.sendRedirect("success.jsp");
     }else {
        response.sendRedirect("error.jsp");
     }
%>
```

→ EmployeeDAO.java

```
package com.app.dao;
public class EmployeeDAO {
    public static int registerEmp (Employee e) {
    SessionFactory factory = new Configuration().
                             Configure("hibernate.cfg.xml").
                             addAnnotatedClass(Employee.class).
                             buildSessionFactory();
    Session session = factory.openSession();
     session.beginTransaction();
        int i = (int) session.save(e); // queries will fire
        Session.getTransaction().Commit(); // Data will Insert
                                              In DB.
        session.close();
        factory.close();
       return i;
}
```

→ Succes.jsp
```
<body>
 registered
</body.
```

→ error.jsp
```
<body>
 error occured failed
</body>
```

Que 1 Develope a web application online book shop using hibernate Persistance. (Saturday evening.)

→ www.hibernate.org/docs
Documentation 5.4

* OnetoOne Mapping

HibernateOnetoOne (3.0)

→ Instructor.java

```
package com.app.entity;
@Entity
@Table(name = "instructor")
public class Instructor{
                    implements Serializable.
 @Id
 @GeneratedValue(Strategy = GenerationType.AUTO)
 @Column(name = "id")
 Private int id;
 @Column(name = "first-name")
 private String firstName;
 @Column(name = "last_name")
 private String lastName;
 @Column(name = "email")
 private String email;
 @OnetoOne(cascade = CascadeType.ALL)
 @JoinColumn(name = "instructor_detail_id").
 private InstructorDetail instructorDetail;

 // Generate getters & setters, Constructor, toString
```

→ InstructorDetail.java :

```
package com.app.entity;
import java.io.Serializable;
@Entity
@Table(name = "instructor_detail")
public class InstructorDetail implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private int id;
    @Column(name = "youtube_channel")
    private String youtube_channel;
    @Column(name = "hobby")
    private String hobby;
    // Generate getters & setters, toString, Constructor.
```

→ hibernate.cfg.xml

```
<!-- <property name = "current_session_context_class">thread</property> -->
<mapping class = "com.app.entity.Instructor"/>
<mapping class = "com.app.entity.InstructorDetail"/>
```

→ App.java

```
package com.app.dao;
public class App {
    public static void main(String[] args) {
        SessionFactory factory = new Configuration()
            .configure("hibernate.cfg.xml")
            .addAnnotatedClass(Instructor.class)
            .addAnnotatedClass(InstructorDetail.class)
            .buildSessionFactory();
```

```
Session session = factory.openSession();
Instructor instructor = new Instructor("anney", "Lok", "anney@gmail.com");
InstructorDetail instructorDetail = new InstructorDetail("http://www.mno.com/
                                    youtube", "singing");

  // associate the objects.
instructor.setInstructorDetail(instructorDetail);
  Session.  : BeginTransaction();
  Session.save(instructor);
  Session.getTransaction().Commit();
  Session.close();
  }
}
```

→ DeleteDemo.java

```
package com.app.dao;
public class DeleteDemo {
        main method {
        SessionFactory factory = new Configuration()
                                  .
                                  .
                                  .
                                  .
        Session session = factory.openSession();
        try {
          session.beginTransaction();
          int theId = 1;
Instructor tempInstructor = session.get(Instructor.class, theId);
System.out.println("foundInstructor: " + tempInstructor);
  // delete the          Instructor
        if(tempInstructor != null) {
              System.out.println("Deleting ---" + tempInstructor);
              Session.delete(tempInstructor);
        }
```

```
        Session.getTransaction().commit();
        Sysout("success");
        }finally{
            session.close();
            factory.close();
        }
    }
```

→ HQL (Hibernate query language) ⇒ Instead of table name we use class name.

⇒ Db independant query language

⇒ Query Interface :-

⇒ createQuery() → Session interface.

Methods of Query interface :-

1) int executeUpdate()
2) List list()

// to get all Records ⇒

```
Query query = session.createQuery("from Employee");
List list = query.list();
```

// to update employee Details ⇒

```
Query query = session.createQuery("update employee set
        fname=:n Where id=:i");
    q.setParameter("n","mayuri");
    q.setParameter("i",111);
    int status = query.executeUpdate();
    tx.commit;
```

→ HQL with aggregate fn :-

// get total salary of all Employees.

```
Query query = session.createQuery("select sum(salary)
                from Employee");
```

// maxm salary of Employee :-

```
Query query = session.createQuery("select max(salary)
                from Employee");
```

→ Multilingual

→ i18n - messages - test.jsp

```
<%@ page language = "java" content types = "text/html; charset = UTF-8
    page Encoding = "UTF-8" %>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/ core" prefix = "c" %>
<%@ taglib uri = "  ------  /jstl/ fmt" prefix = "fmt" %>
<HiHe>  </HiHe>
<c:set Var = "Thelcale"
Value = "${not empty param.thelocale ? param.thelocale :
pageContext.request.locale}" scope = "session" />
<!-- param.theloeale - Read the local param sent by a link
pageContent.request.locale - Read the local sent by browses HTTP
request -->
<fmt:setlocale Value = "${ Thelocale }" />
<fmt:setBundle basename = "com.app.jsp.tagdemo.i18n.resources.
                           Base Package Name
mylabels" />
</head>
<body>
<a href = "i18n-messages-test.jsp?Thelocale = en_US"> English (US)</a>
|
<a href = ------ = es_ES">Spanish (ES)</a>
|
<a href = ------ = de_DE"> German (DE)</a>
|
<a href = ------ = hi_IN"> Hindi (IN)</a>
|
<a href = ------ = mr_IN"> Marathi (IN)</a>
<hr>
```

```
<fmt:message key="label.greeting"/><br><br>
<fmt:message key="label.firstname"/><i>Ameylake</i><br><br>
<fmt:message key="label.lastname"/><i>Oaklake</i><br><br>
<fmt:message key="label.welcome"/><br><br>
<hr>
Selected Locale : ${theLocale}
</body>
</html>
```

<!-- com.app.jsp.tagdemo.i18n.resources.mylabels.nn_IN.properties →

catalog - internal

A file → New → Maven project → maven-archetype-webapp (1.0)
→ creating a Web app in → add dependencies → 3 → mysql connector.
→ JSP
→ hibernate core
→ update the project.
↓
right click project → maven → update project
↓
☑ Force update

→ src → main → rightclick → folder → Java folder will get.

* Hibernate Named query: -

→ way to use any query by providing meaningfull Name.

@NamedQueries - defined multiple named queries.
@NameQuery - defined single name query.

Que. 1) Can we traverse ↑ reverse using Iterator? NO
2) Can we perform remove operation using iterator? NO
3) Is a iterator a universal cursor??
4) Can iterator apply to map interface?

3 Cursors in Java
→ Iterator
→ ListIterator
→ Enumeration.

HibernateNamedquery (5.0)

→ Employee.java

```
package com.app;
@Namedqueries(
    {
     @Namedquery(
        name = "findEmployeeName",
        query = "from Employee e Where e.name = :name"
                  ↳ Entity Name
     ),

     /*@Namedquery(
        name = "",
        query = "") */

    }
)
@Entity
@Table(name = "emp123")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.Auto)
    int id;
    String name;
    int salary;
    String job;
    // Generate getters & setters, constructor, tostring
}
```

→ App.java

```
package com.app;
public class App {
    main method {
        SessionFactory factory = new Configuration()
            .configure("hibernate.cfg.xml")
            .addAnnotatedClass(Employee.class)
            .buildSessionFactory();
```

```
Session session = factory.openSession();
    // Insert Data first.
    // hibernate Named query
TypedQuery<Employee> query = session.getNamedQuery("findEmployeeByName");
query.setParameter("name", "prachi");
List<Employee> employees = query.getResultList();
Iterator<Employee> itr = employees.iterator();
  While(itr.hasNext()) {
        Employee e = itr.next();
          Sysout(e);
  }
    // session.getTransaction.Commit();
     Session.close();
  }
}
```

→ Hibernate.cfg.xml

---

* Spring *

→ Optimal to J2EE.
↘ or alternative

→ change perspective from J2EE to Java.

→ Helper classes ---- makes thing easier.

→ Rod Johnson 2003

* IOCDemo → IOC (Invesion of Control) :- the approach of Outsaureing the Construction and management of objects.

Que 1) DS for Stack, Vector &
2) Diff. ArrayList & Vector &
3) Diff. Hashmap & Hashtable.
4) Capacity of Hashtable, Hashmap &
5) Why Notify, NotifyAll are methods of object class & not
6) What is Reentrant Lock in multithreading &
7) Concurrent Hashmap &
8) Difference betn Join & yeild &
9) Why there are two ways to implementing Runnable & extending Thread class &
10) Thread-scheduler &
11) Two Threads Running in bagreound & ⇒ garbage calleetor.
⇒ main-Thread.
12) What is executor framework &
→ also known as thread pool.
13) Setpriority, maxpriority not worked at Software level but Hardware level so it will not worked.
14) Automatic Resource management & (ARM)
15) Inner class & → Method local &
16) Why need of annonymous inner class &
17) Diff. throw & throws.
18) Write a program to get Runtime information about class & (Reflection)
19) Why equals & Hashcode come together &

21) Diff. betn Comparable & Comparator?
→ String & all wrapper classes implements Comparable.
22) Converting date to string & & string to Date & formatting & parsing &
23) What is cloning?
↓ → must implement cloanable.
method of object class -
24) What is shallow clone & Deep clone?
25) Diff. betn checked & Unchecked Exception?

Spring

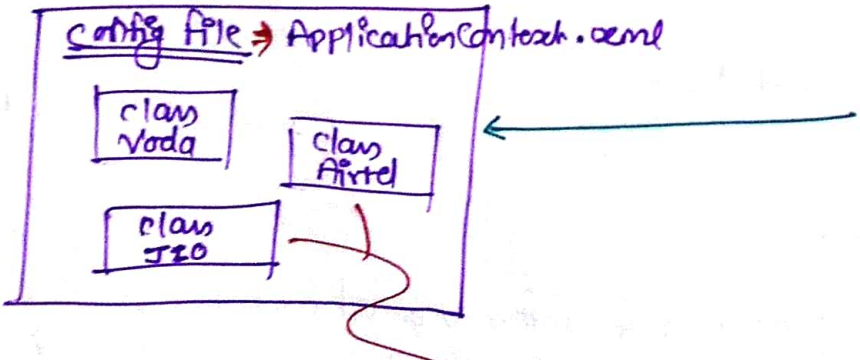
1) change perspective to Java,
2) Create Java project.
3) right click on project → Build path → configurebuildpath
→ goto libraries → add External Jars → Spring beans
→ Spring Context
→ Spring core
→ Spring expression
all release
→ Commons-logging.jar

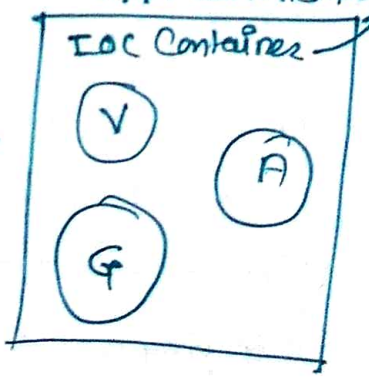
{ Goto browse → Search → xml Schemabased configuration
→ 40. xml schema Configuration → 10.2.1 Referencing Schema →
Copy from box second equivalent file.

{ ~~Appln context.xml~~ xxx.
Appln Context will read Configuration file & all beans & create object of
bean & i.e. Spring bean.

→ How to make app configurable :-
→ Spring framework
→ I can create objects for you.
→ I Can manage objects for you,

IOC Container → ApplicationContext





Spring beans

getBean("v");

getBean("name of Bean"); // method of

→ 2 types of IOC Container :-

1) BeanFactory - Interface - legacy

2) ApplicationContext - Interface - advanced

→ sim.java

```
package com.app
public interface sim {
        public void calling();
        public void data();
}
```

→ Vodafone.java

```
public class Vodafone implements Sim {
    public Vodafone() {
        System.out.println("Vodafone constructor called ----- ");
    }
    @override
    public void calling() {
        System.out.println("Calling using Vodafone");
    }
    @override
    public void data() {
        System.out.println("Browsing data using Vodafone!");
    }
}
```

→ Jio.java package com.app
// same as Vodafone class.

→ appMobile.java
import org.springframework.context.ApplicationContext
import org.springframework.context.support.
ClassPathXmlApplicationCon
package com.app
public class Mobile {
main method {
/* Vodafone Voda = new Vodafone();
Voda.calling();
Voda.data(); */

/* Jio jio = new Jio();
jio.Calling();
jio.data(); */

/* ApplicationContext Context = new ClassPathXmlApplicationContext
("applicationContext.xml"); */

/* Vodafone V = (Vodafone) Context.getBean("Voda");
V.calling();
V.data(); */

ApplicationContext Context = new ClassPathXmlApplication("applicationContext

Sim s = Context.getBean("sim", Sim.class);
S.calling();
S.data();
→ Type casting is not needed now.

}

→ applicationContext.xml
<beans> (copy xmls configuration schema)
<!-- bean definations here -->
<bean name = "sim" class = "com.app.Jio"></bean>
<!-- Jio sim = new Jio(); -->
→ changing com.app.Vodafone also
</beans>

* Dependency Injection *

⟸ Simply means injecting values for your dependency.

1) family
2) Job
3) car
4) Hometown (passport)
5) Name 6) Home NO.

⟶ In Java, they all are objects.

```
class Me {
    String name = "Ameerpet";
    int home_no = 123;
    family f = new family();
    Job j = new Job();
    ArrayList<Integer> impNos = new ArrayList<Integer>();
    impNos.add(31224352);
    impNos.add(31224353);
}
```

Type parameter

⇒ No hard coding required any more
⇒ Spring will inject values for your dependency.

By using :- 1) Setter injection.
2) Constructor injection.

⟶

```
class me {
    String name;
    int home_no;        } dependency in form of literals.
    family f;
    Job j;              } Dependency in form of objects.
    ArrayList<Integer> impNo;  } Dependancy in form of collection.
}
```

→ DependancyInjection → Java project → click right → configure

→ Convert to maven project → packaging → Jar (Java SO) → finish.

update project right click → maven update after this

→ Dependency → Spring beans → 5.0.10 RELEASE
→ Spring Core → 5.0.10 RELEASE
→ Spring Context → 5.0.10 RELEASE
→ ~~Spr~~ Apache commons Loggin → 1.2

→ DIDemo

→ Human.java

```
package com.app;
public class Human {                    Heart heart;
        Sysout("Human Constructor");
}
public Human(Heart heart){
        Sysout("I'm in Constructor");
        this.heart = heart;
}
public void setHeart(Heart heart){
        Sysout("I'm in setter.....");
        this.heart = heart;
}
public void pumping(){
        if(heart != null){
                heart.pump();
        }else{
                Sysout("you are dead");
        }
}
```

→ Heart.java

```
public class Heart {
        public void pump() {
                System.out.println("you are alive");
                System.out.println("your heart is pumping");
        }
}
```

→ beans.xml

(copy xmlns schema)

```
<bean id="human" class="com.app.human">
<!-- <constructor-arg ref="heart"></constructor-arg> -->
        → By constructor injection
<property name="heart" ref="heartobject"></property>
        → by setter injection.
</bean>
<!-- Human human = new Human(heart); -->
        → This should be match with <constructor-arg> ref=heart
<!-- <bean id="heart" class="com.app.heart"></bean> -->
<bean id="heartobject" class="com.app.heart"></bean>
        → match with setter injection.
<!-- Heart heart = new Heart(); -->

<!-- Human human = new Human(); -->

<!-- Heart heartobject = new Heart(); -->

<!-- human.setHeart(heartobject); -->
```

→ App.java

Com.app

```
public class App {
        main method {
ApplicationContext context = new ClassPathXmlApplicationContext
                                ("beans.xml");
Human h = context.getBean("human", Human.class);
h.pumping();
} }
```

* Autowiring :-

byName :- id name & ref variable name ↑ (In bean class) should be same,
↕ setter
byType :- dependency type and class attribute type ↑ (In bean class) should be same
byConstructor :-

beans.xml →

<!-- <bean id = "human" class = "com.app.Human" autowire = "constructor"/>
<bean id = "heart" class = "com.app.Heart"/>

<!-- <bean id = "human" class = "com.app.Human" autowire = "byName"/>
<bean id = "heart" class = "com.app.Heart"/> -->
→ Human class ref variable should match.

<bean id = "human" class = "com.app.Human" autowire = "byType"/>
<bean id = "heart" class = "com.app.Heart"/> (ref var type)
→ Human class
dependancy type & class attribute type should match.

* Autowiring :- Injecting value for object dependency implicitly
Cannot be applied for String & primitives.
@Autowired.

→ AutowiringDemo (Java project → Maven)

{ Search → spring-context on maven Repository site on browser
40.2.8 the context schema → copy → paste beans.xml file }

→ beans.xml → To activate @autowired tag

<context:annotation-config/>
<bean id = "human" class = "com.app.Human"></bean>
<bean id = "heart" class = "com.app.heart"></bean>

→ Human.java.

```
public class Human {          → field level.
    @Autowired
    Heart heart;

    public Human() {
        Sysout(".. Human Constructor");
    }

    // @Autowired
    public Human(Heart heart) {
        Sysout("In Constructor---");
        this.heart = heart;
    }

    public void pumping() {
        if(heart != null)

    //@Autowired
    public void setHeart(Heart heart) {
        Sysout("In Setter---");
        this.heart = heart;
    }

    // pumping method from previous ex:-

}
```

@Qualifier ⇒ Cannot work with Constructor.

→ Required when 2 beans of same class

→ always comes with @Autowired.

→ Human.java

```
public class Human {
    //@Autowired
    //@Qualifier("OctopusHeart")
    Heart heart;

    public Human() {
        System.out.println("Human constructor");
    }
}
```

change it to humanheart also & run

```
//@Autowired
//@Qualifier("HumanHeart")
public Human(Heart heart) {
    System.out.println("In para constructor");
    this.heart = heart;
}
```

✗ Qualifier does not applied to Constructor

Qualifier always comes with @Autowired & works for setter injection & field level only

```
@Autowired
@Qualifier("HumanHeart")
public void setHeart(Heart heart) {
    System.out.println("In setter");
    this.heart = heart;
}
```

→ heart.java

```
public class Heart {
    private String nameOfAnimal;
    private int noOfHeart;

    // Generate getters & setters, default constructor
}
```

→ beans.xml

(copy context-schema)

<context:annotation-config/> → to activate @Autowired.

<bean id="human" class="com.app.Human"></bean>

<bean id="humanHeart" class="com.app.Heart">

<property name="nameofAnimal" value="Human"></property>

<property name="noofHeads" value="1"></property>

</bean>

<bean id="octopusHeart" class="com.app.Heart">

<property name="nameofAnimal" value="Octupus"></property>

<property name="noofHeart" value="3"></property>

</bean>

---

* PropertiesDemo (Java project → maven) packaging → Jar
update project.

→ Student.java

public class Student { → This value will go in name in displayStudentInfo() method.

@Value("Ameylok")

private String name;

private String interestedCourse;

@Value("${student.hobby}") → Expression language Reading from properties file.

private String hobby;

public Student() { }

```
public void setName(String name){
        this.name = name;
}
@Value("SDM")
public void SetInterstedCourse(String interestedCourse){
    this.interestedCourse = interestedCourse;
}
public void setHobby(String hobby){
        this.hobby = hobby;
}
public void displayStudentInfo(){
        Sysout("Student name = " + name);
        Sysout("Intersted Course = " + interestedCourse);
        Sysout("Student hobby = " + hobby);
}
```

create file

→ Student-info.properties (parallel to package)

```
1 Student.name = Pradhi
2 Student.interestedCourse = SAG
3 Student.hobby = Singing
```

→ beans.xml

(copy Context-schema)

```
<Context: annotation-config/>
<Context: propertyPlaceholder location = "classpath:
                                    Student-info.properties"/>
```

write manually

Hard coded values.

```
<!-- <bean id = "student" class = "com.app.Student">
< property name = "name" Value = "Amey Lok">
< property name = "interestedCourse" Value = "Java"> </property>
< property name = "hobby" Value = "football"></property>
</bean> -->
```

// Reading from properties file

<!-- <bean id = "student" class = "com.app.Student">

<property name = "name" value = "${student.name}" ></property>

<property name = "interestedCourse" value = "${student.interestedCourse}" ></property>

<property name = "hobby" value = "${student.hobby}"></property>

</bean> -->

// @value → used in bean class

<bean id = "student" class = "com.app.Student"></property>

→ App.java

```
package com.app;
public class App{
        main method {
ApplicationContext context = new classpathxmlApplicationContext
                                ("beans.xml");
Student s = context.getBean("student", student.class);
s.displayStudentInfo();
```

* Spring MVC *

→ Spring MVC (3.1) ⇒ Dynamic web project ⇒ DO not generate xml file
⇒ Add Jars ⇒ All release.jar files from folder.

→ web-INF ⇒ Spring-mvc-demo-servlet.xml → for component scanning & View Resolver.
2) web.xml (from folder)

web.xml → <display-name> Springmvc </display-name>

dispatcher Servlet → goto Java Resources → libraries → webapp libraries
→ Spring-WebMVC → Expand → 1st package → org.springframework.web.servlet.
→ Expand → DispatcherServlet.class → rightclick → copy qualified name
→ goto web.xml → paste in <servlet-class> tag → and remove .class from it.

→ mention base package name in Steps servlet xml at <context: component-scan
→ for internal ResourceViewResolver class ⇒ in web.xml web-app library → Springweb-mvc
→ expand → org.springframework.web.servlet.View → expand → InternalResourceViewResolver.class → right click → copy qualified name

→ WebINF → Views (create folder) → main-menu → .jsp

Controller → View
&
View → controller
} model will do it is Controller

Springmvc (3.1)

→ ~~webmvc~~

~~webapp~~

→ Model – Container – Actual data – String, Objects,
Collection – so on –

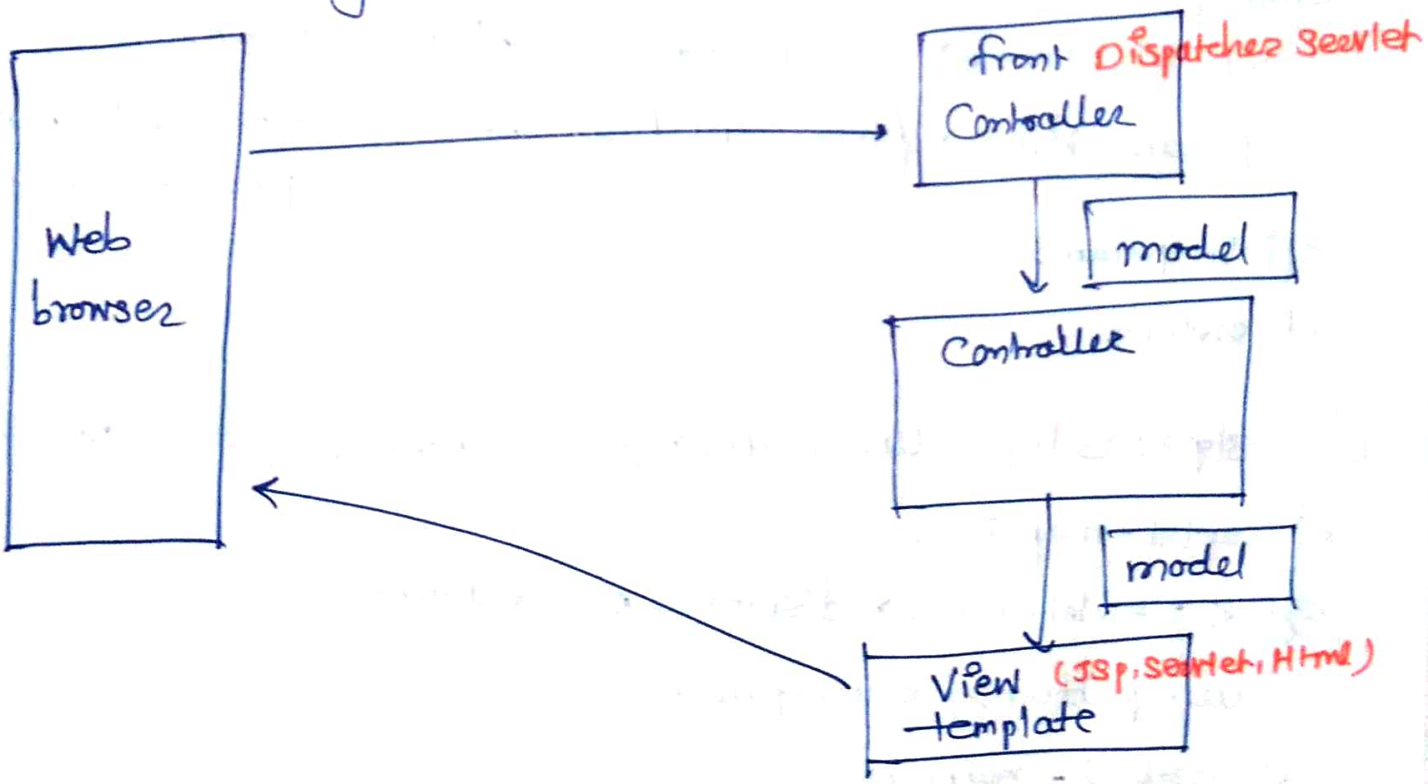
View – JSP, HTML

Controller → Business logic.
↳ {to add data to model & give to View}

→ framework for building web-app in Java.

→ Based MVC design pattern.

→ Leverages core features of Spring DI, IOC.



* Spring mvc configuration :-

Part: 1) Spring mvc Configuration process.
– add Configuration to the file : WEB-INF/web.xml

1) Configure Spring mvc DispatcherServlet.
2) Setup URL mapping to Spring mvc DispatcherServlet.

pack8) Add Configuration to the file: web-INF/spring-mvc-demo-Servlet.xml

→ Add Support for Spring Component scanning.
→ Add Support for Conversion, Validation, formatting.
→ Configure Spring mvc View Resolver.

→ web.xml

<!-- Step1: Configure Spring mvc DispatcherServlet -->
<Servlet>
<Servlet-name> dispatcher </Servlet-name>
<Servlet-class> org.springframework.web.Servlet.DispatcherServlet </Servlet-class>
→ copy

<init-param>
<param-name> ContextConfigLocation </param-name>
<param-value> /WEB-INF/spring-mvc-demo-Servlet.xml </param-value>
</init-param>
</Servlet>

<!-- Step2: Setup a URL mapping for Spring mvc DispatcherServlet -->
<Servlet-mapping>
<Servlet-name> dispatcher </Servlet-name>
<url-pattern>/</url-pattern>
</Servlet-mapping>

→ Spring-mvc-demo-Servlet.xml

<Step3: Add Support for Component Scanning -->
<context:Component-scan base package = "com.app"/>

<!-- Step 4: - Add Support for Conversion, formatting & Validation.
<mvc:annotation-driven/>

<!-- Step 5: Configure mvc View Resolver -->
<bean class = "org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name = "prefix" value = "/WEB-INF/views/"></property>

<property name = "suffix" value = ".jsp"></property>

</bean>

</beans>

→ HomeController.java

package com.app.controller;

@Controller
public class HomeController {

@RequestMapping("/")
public String showMyPage() {

return "main-menu"; // WEB-INF/view/main-menu.jsp

}

}

→ main-menu.jsp

<body>
Welcome to Spring mvc!!!

 click here !!
</body> </html>

→ StudentController.java

```
package com.app.controller;

@Controller
@RequestMapping("/student")
public class StudentController {
    @RequestMapping("/hello")
    public String showpage() {
        return "student-form";
    }

    @RequestMapping("/processform")
    public String processStudentform(HttpServletRequest request,
                                     Model model) {
        String studentName = request.getparameter("txtName");
        StudentName = studentName.toUpperCase();
        String result = "Hey" + StudentName + "Welcome !!!";
        model.addAttribute("message", result);
        return "student-Confirmation";
    }
}
```

→ Studentform.jsp

```
<body>
<form action = "processform" method = "post">
Enter a Student Name : <input type = "text" name = "txtName">
<br>
<input type = "submit" value = "GO">
</form> </body> </html>
```

→ Student-Confirmation.jsp

```
<body>
Student confirmed:
<hr>
Student Name : ${ message }
</body>
</html>
```

edit Student Controller.java → OR

there is parameter in request obj. Instead of Http. Servlet & get Parameter 2 lines we done with 1 annotation

```
@RequestMapping("/processform")
public String processStudentform (@RequestParam("txtName")
        String StudentName, Model model) {

    StudentName = StudentName.toUpperCase();
    String result = "Hey" + StudentName + " welcome to Interay";
    model.addAttribute("message", result);
    return "student-Confirmation";
}
```

Go to Src → libraries → web app libraries → goto Spring web mvc → meta-Inf at last → expand → Spring-form.tld → open
To use spring form tag.

#Spring-mvc

→ Student-form.jsp

```
<%@taglib uri="http://www.springframework.org/tags/form"
          Prefix="form" %>
<body>
<form:form action="processform" modelAttribute="student">
Enter FirstName : <form:input path="firstName" />
<br><br>
Enter LastName : <form:input path="LastName" />
<br><br>
```

- Select Country:

```
<form:select path="country">
<form:options items="${student.countryoptions}"/>
</form:select>
<br><br>
Enter language:
Java <form:radiobutton path="Language" value="Java"/>
python <form:radiobutton path="Language" value="python"/>
Claud computing <form:radiobutton path="Language" value="claud computing"/>
<br><br>
Select Operating-Systems:-
Mac <form:checkbox path="operatingsystems" value="Mac"/>
Windows <form:checkbox path="operatingsystems" value="windows"/>
Linux <form:checkbox path="operatingSystems" value="Linux"/>
<br><br>
<%--<form:button type="submit" value="submit"></form:button>-->
<input type="submit" value="submit">
<form:form></body></html>
```

→ Student.java

```
package com.app.model;
public class Student{
    private String firstName;
    private String lastName;
    private String country;
    private Map<String,String> countryoptions;
    private String[] operatingsystems;
```

```
public Student() {
    countryOptions = new HashMap<String, String>();
    countryOptions.put("EN", "India");
    countryOptions.put("DE", "Germany");
    countryOptions.put("ES", "Spanish");
    countryOptions.put("FR", "France");
}
// Generate toString, getters & setters.
```

→ Student-Controller.java

```
package com.app.Controller;
@Controller
@RequestMapping("/student")
public class StudentController {
    @RequestMapping("/hello")
    public String showPage(Model model) {
        //create a student object.
        Student s = new Student();
        // add a Student to Container
        model.addAttribute("student", s);
        return "student-form";
    }
    @RequestMapping("/processform")
    public String processStudentform(@ModelAttribute("student")
                                     Student student) {
        System.out(theStudent);
        return "Student-confirmation";
    }
}
```

→ Student-Confirmation.jsp

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

<body>
Student confirmed : ${student.firstName} ${student.lastName}
<br><br>
Student country : ${student.country}
<br><br>
Student language : ${student.language}
<br><br>
Operating System :
<ul>
<c:forEach var="temp" items="${student.operatingSystems}">
${temp}
</c:forEach>
</ul> </body>
```

* Spring MVC form tags :-

Form tags to generate HTML for you :-

form tag	Description
form:form	main from container
form:input	Textfield
form:textarea	Multi-line text field
form:checkbox	checkbox
form:radiobutton	Radio buttons.
form:select	Drop down list.

Data binding :-

- Spring mvc form tags can make use of data binding.
- Automatically setting/retrieving data from a java object/bean.

```
<%@taglib prefix="form" uri=" ">
```

AOP (Aspect Oriented programming) :-

{ mvn repository → AspectJ (search) (1.9.7) → File => Jar (2.0MB)
weaver

→ SpringAopDemo (Java project)

{ Goto build path & add all Jar files RELEASE.jar from Springframework folder. }

{ add aspectJweaver.jar & Common-logging jar also }

{@Before (ex)} → pointcut expression

→ New Requirement :-
→ Need to add logging to our DAO methods.
→ Add some logging statements before the start of the method.

DAO → add logging code

```
public void addAccount(Account theAccount, String userId){
        //add code for logging
        // add code for security check
    Session session = SessionFactory.getCurrentSession();
    Session.save(theAccount);
}
```

If 10L layers, add Logging Security to each layer will now become is complex.

AOP ⇒ Based on a Concept Aspect.

"Cross-cutting concern"

Concern ⟶ Logic & functionality.

* Aspect can be re-used at multiple locations.

⟶ Benefits of AOP :-

1) Code for aspect is defined in a single class.
2) Configureable.

AOP Terminology

* Aspect AOP terminology :-

Aspect :- module of code for cross cutting concerns.

Aspects can be reused at multiple locations.

Advice :- What action is taken & When it should be applied.

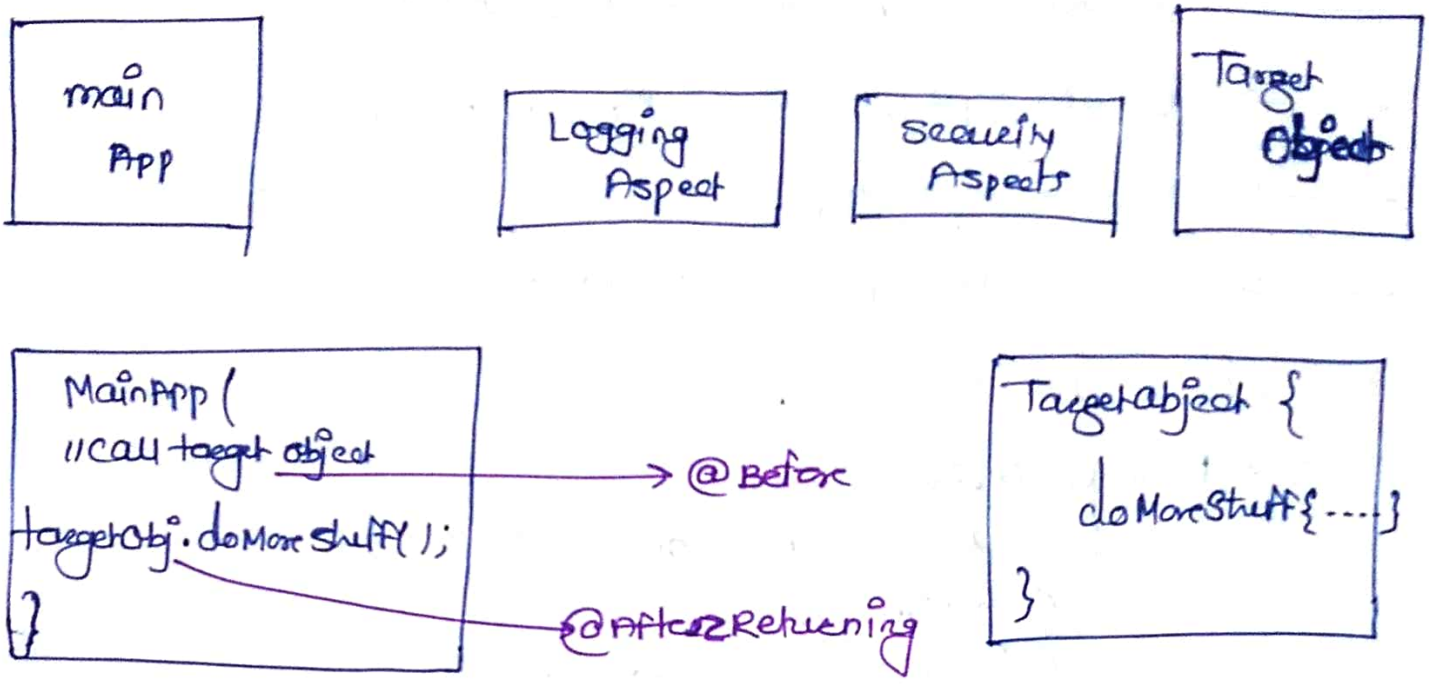
Jointpoint : When to apply code during program execution.

Pointcut :- A predicate expression for where advice should be applie

Advice Type ⇒

1) Before advice ⇒ run code before the method.
2) After finally advice ⇒ run code after the method.
3) After returning advice.
4) After throwing advice ⇒ run after the method (Exception is thrown)
5) Around advice ⇒ run before & after method.

AOP @Before Advice Overview



Pointcut Expression Language:-

→ execution (modifiers-pattern? return-type-pattern declaring-type-pattern? method-name-pattern (param-pattern) throws-pattern?)

? - optional

modifiers-pattern? - public or *(matches on everything)

declaring-type-pattern - class-name

method-name-pattern - method name to match

param-pattern - param to match

throws-pattern? - Exception type to match.

AOP - match on Parameters:

For param pattern -

() - matches a method with no arg.

(*) - matches a method with one arg of any type

(..) - matches 1 ——— with() or more args of any type.

* Steps to Create project :-

→ Create a target object = AccountDAO

→ Create Spring Java Config class (No XML)

→ Create main app

→ Create an Aspect with @Before advice.

* Spring Aop Demo :

→ DemoConfig.java

```
Package com.app.aopdemo;
@Configuration
@EnableAspectJAutoProxy
@ComponentScan("com.app.aopdemo")
public class DemoConfig{
    @Bean(name="bean1")
    public AccountDAO accDAO(){
        return new AccountDAO();
    }
    @Bean(name="bean2")
    public MembershipDAO memDAO(){
        return new MembershipDAO();
    }
}
```

→ AccountDAO.java

```
Package Com.app.aopdemo.dao;
@Component
public class AccountDAO {
    public void addAccount(){
        System.out.println("Doing my DB work: adding
                                an account");
    }
```

```
public void doWork(int a, int b) {
        System.out.println("Executing");
}
```

→ MembershipDao.java

```
package com.app.aopdemo.dao;
@Component
public class MembershipDao {
        public void addAccount() {
                System.out.println("Doing my MemDB work: Adding an account");
        }
        public void addSilly() {
                System.out.println("In Silly method");
        }
}
```

→ MyDemoLoggingAspect.java

```
package com.app.aopdemo.aspect;
@Aspect
@Component
public class MyDemoLoggingAspect {
/*  @Before("Execution(public void com.app.aopdemo.dao.
                AccountDAO.addAccount())")
    public void beforeAddAccountAdvice(){
        System.out.println("Executing @Before advice on addAccount()");
    } */
/* // match method starting with add in any class
    @Before("execution(public void add*())")   → all add methods
    public void beforeAddAccountAdvice() {     → pointcut expression
        System.out.println("Executing @Before on addAccount()");
    } */
```

```
// @Before("execution(* * methodName())")
@Before("execution(public * doWork(..))")        Return type
public void beforeAddAccountAdvice(){
    Sysout("Executing @Before advice on doWork()");
}
```

→ mainDemo.java

```
package com.app.aopdemo;
public class maindemo {
    main method{
    // Read spring config class
AnnotationConfigApplicationContext Context =
    new AnnotationConfigApplicationContext(DemoConfig.class);
    // get the bean from the Spring Container
AccountDAO theDAO = Context.getBean("bean1", AccountDAO.class);
MembershipDAO memDAO = Context.getBean("bean2", MembershipDAO.class);
    theDAO.addAccount();
    memDAO.addAccount();
    memDAO.addSilly();
    theDAO.doWork(10,20);
    // close the Context
    Context.close();
}
```

---

1) Diff. bet Aggregation & Composition?
2) What is garbage collector? Can we force garbage collector in java?
3) What is Singleton Design pattern?
4) What is the order of Constructor call in case of inheritance.
5) Interface must be public Why?
6) Can abstract class can have constructer?
7) Why Arrays are called as first-class objects?

8) What is Volitile keyword in Java?

9) What is usage of Regular expression (Reg-ex) in Java?

10) Can I write main method in class?

11) What is diff bet Exception & error in java?

12) ——— " ——— checked & unchecked exception?

13) Can I explicitly call finalize method.

14) What is Reflection & where it is used?

15) What is base class for all exceptions & error in Java?

16) Can I write try block without catch?

17) Can I write multiple catch block for single try block?

18) What is diff. bet assertion & Exception?

19) How are the user-defined Exceptions created in Java?

---

* Problem Stmt:

How can we reuse a pointcut expression?
Want to apply to multiple locations?

Soln: create a pointcut declaration once.
Apply it to multiple devices.

→ SpringAopDemo1

→ MyDemoLoggingAspect.java

* right click on project → build path → configure build path
→ add library
→ User library
→ next
→ User libraries
→ New
→ give name, SpringAop2
→ ok
→ folder → add External Jar's
→

```
package com.app.aopdemo.aspect;
@Aspect
@Component
public class MyDemoLoggingAspect {
    @pointcut("execution(* com.app.aopdemo.dao.*.*(..))")
    private void forDaoPackage() {}
```

any class, any method, any parameter

Jars → All Jars RELEASE & AspectJWeaver & commons logging

```
@Before ("forDaoPackage()")
public void beforAddAccountAdvice() {
    System.out("Executing @Before advice on
                addAccount()");
}

@After ("forDaoPackage()")
public void performAPIAnalatics() {
    System.out("\n ----> Performing API Analytics");
    return false;
}
```

→ AccountDao.java

```
package com.aopdemo.dao;
@Component
public class AccountDemo {
    public void addAccount(Account theAccount,
                           boolean VipFlag) {
        system(getclass()+ "Doing my DB work:
                            Adding an account");
    }

    public boolean doWork() {
        System.out(getclass()+ "DO Work");
        return false;
    }
}
```

→ MembershipDao.java

```
public String addSilly() {
    return getclass()+
        "Doing my MemDB work: Adding an
         membership Account";
}
```

```
Public void goToSleep() {
    System.out.println(getClass() + " I am going to sleep now");
}
```

→ Account.java

```
package com.app.aopdemo;
public class Account {
    private String name;
    private String level;
    public // Generate getters & setters.
```

→ DemoConfig.java.

Same as previous program.

→ MainDemo.java

```
// Read Spring config class
// get the bean from the Spring Container
```

← copy from previous Example

```
Account myAccount = new Account();
theDAO.addAccount(myAccount, true);
theDAO.doWork();
String s = memDAO.addSilly();
System.out.println(s);
memDAO.goToSleep();
// close the context
context.close();
}
}
```

@ After Returning :-

⇒ run after the method (success execution (no exception))

post processing Data ⇒

post process the data before returning to callee format the data or to enrich the data.

to access the return value

post processing data

it executes after a joinpoint completes normally

Joinpoint ⇒

Is a point of execution of the program, such as the execution of method or handling an ~~expression~~ exception.

→ Spring Aop Demo.

→ Account.java

```
// previous program
// Generate parameterised Constructor.
```

→ AccountDao.java

```
package com.app.aopdemo.dao;
@Component
public class AccountDao{
    private String name;
    private String serviceCode;
public List<Account> FindAccounts(boolean tripWire){
    if (tripWire) {
        throw new RuntimeException("No sauce for you!!!");
    }
List<Account> myAccounts = new ArrayList<Account>();
    // Create a sample Accounts
```

```
Account a1 = new Account("ajinkya", "bhosale");
Account a2 = new Account("shubham", "patil");
Account a3 = new Account("pratik", "jadhav");
    myAccounts.add(a1);
    myAccounts.add(a2);
    myAccounts.add(a3);
        return myAccounts;
}
```

// Generate getters & setters.

// Add Account & toget same method from priviaupragram

→ MyDemoLoggingAspects.java

```
package com.app.aopdemo.aspect
@Aspect
@Component
public class MyDemoLoggingAspects {          (any modifier)
/*@AfterReturning(pointcut = "execution(* com.app.aopdemo.dao.
                    AccountDAO.findAccounts(..))",
                    returning = "result")
    public void afterReturningfindAccountAdvice(JoinPoint theJoinPoint,
                    List<Account> result) {
        String method = theJoinPoint.getSignature().toString();
        System.out("\n ===> Executing @AfterReturning on method:"
                    + method);
        System.out("\n ===> Result is:" + result);
} */
```

```
@AfterThrowing(
pointcut = "execution(* com.app.aopdemo.dao.AccountDAO." +
                    "findAccounts(..))",
                    throwing = "theExec")

public void afterThrowingFindAccountAdvice(JoinPoint theJoinPoint,
                                           Throwable theExec){

String method = theJoinPoint.getSignature().toShortString();

System.out("\n ===> Executing @AfterThrowing on method:" + " " + method);

System.out("\n ===> the exception is:" + theExec);

}
```

→ AfterReturningAdviceDemo.java

```
package com.app.aopdemo;

public class AfterReturningAdviceDemo {

    main method {

    //Read spring config class
```

Previous Demo

```
    //get the bean from the Spring Container
```

make it false for @AfterReturning after

```
List<Account> theAccounts = theDAO.findAccounts(true);
```

↳ for @AfterThrowing

```
    //display the Accounts.

    System.out("\n\n Main program : AfterReturningAdvice: ");
    System.out("-----");
    System.out(theAccounts);
    System.out("\n");
      Context.close();
}
```

→ Advice: AfterThrowingAdviceDemo.java

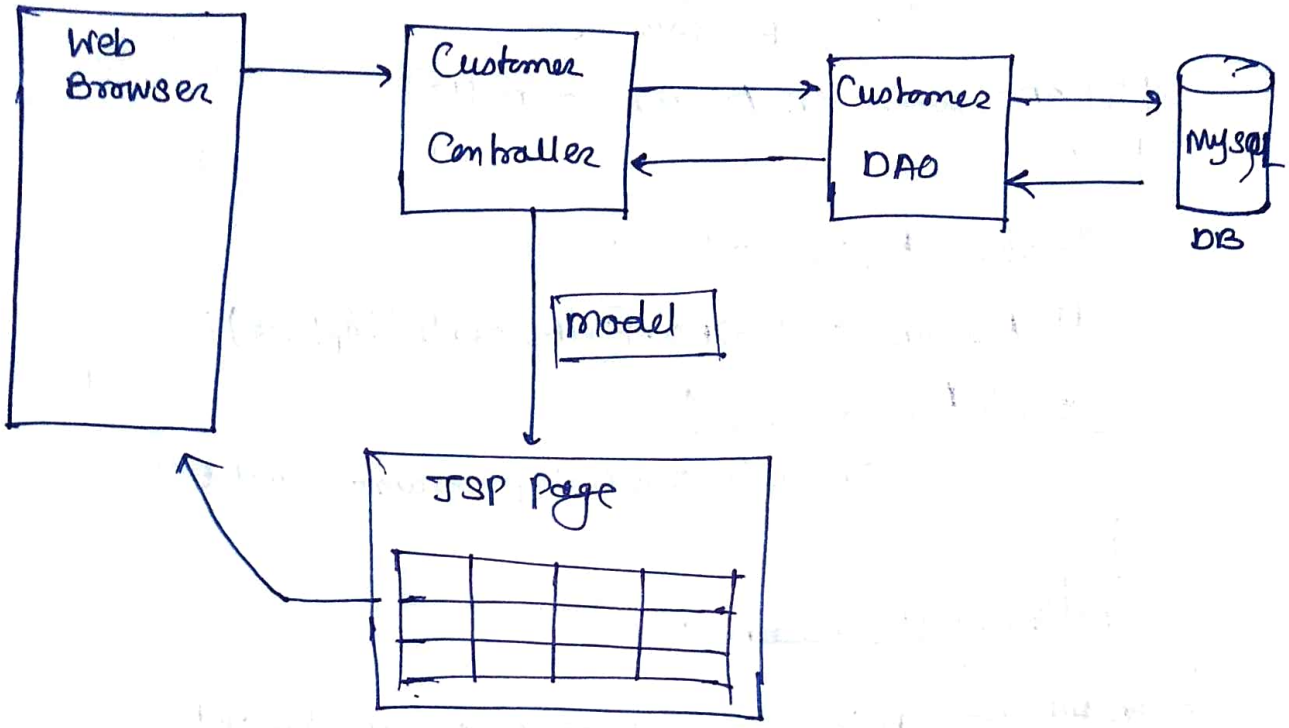
```
package com.app.aopdemo;

public class AfterThrowingAdviceDemo {
        main method {
        // from previous Se.

    List<Account> theAccounts = null;

    try {
        boolean tripWire = true;
        theAccounts = theDAO.findAccounts(tripWire);
    } catch (Exception e) {
            System.out("\n\n Main program ---" + e);
    }
    // display the accounts.

System.out("\n\n Main program : AfterThrowingAdvice :");
System.out("-----");
System.out(theAccounts);
System.out("\n");
Context.close();
}
```

→ spring-demo-aop-account

CRM :- Spring + Hibernate

(Customer Relationship Management).

Get List of Customers.



Refactor → Add a Service layer.

@Service

Intermediate layer for Custom bussiness logic.

→ change perspective to JavaEE.

→ CRM (3.1) do not generate web.xml
dynamic web project.

→ web-INF → ~~Web~~ lib → add all Jars → mysql Connector → spring Jars

→ hibernate Jar → JSP & JSTL Jar → Servlet Jar

<context:Component-scan ——> to find controller package

Goto Java Resources → libraries → web app libraries → CSPO.jar → expand → com.mchange.v2.CBpo → package → expand → CombopooledData-Source.class → copy qualified name → paste it in servlet-xml

Step 6 → class = " "

In JDBCURL → after DB name remove that next line ?useSSL---
If we get error then only write it.
?useSSL=false&serverTimeZone=UTC"

@Transactional ⇒ Uses Aop behind the scene where it will begin & Commit transaction → step 8

Add CSS file ⇒
Web-Content → rightclick → new folder → resources → create subfolder & named it css.

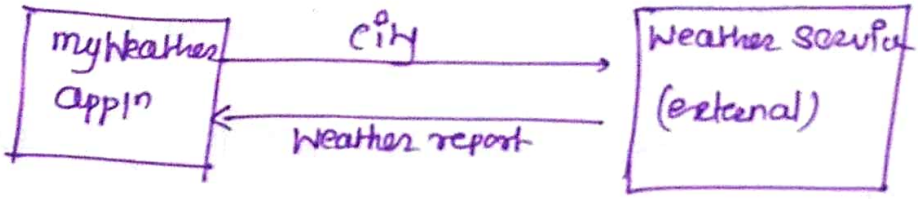
→ resources → rightclick → new folder for Images there
→ new folder for Javascript also

GetMapping ⇒ If we want to get entity or list of entity we use Getmapping.

PostMapping ⇒ If we want to save an entity we use postmapping.

* Rest APIs ⇒ Rest web Services :-

Business problem :-
Wheather report for city.



REST ⇒ Representational state transfer

→ Lightweight approach for Communication betn appln.
→ REST is language Independent.
→ REST appln. Can use any data format.
→ XML and JSON (Javascript object Notation)

possible soln ⇒ openweather.org

JSON ⇒ Light-weight data format for storing and exchanging data.

JSON-Ex ⇒

```
{
"id": 14
"firstName": "Ameya"
"active" : true
```

JSON Value :- Numbers : no quotes.
String : Double quotes
boolean : true/false
Nested JSON Object.
Array
Null.

```
{
"id : 14,
"firstName" : "Prashi",
"active" : true
"address" : {
"street" : "xyz",
"city" : "pune",
----.
}
```

"language" : ["Marathi", "English", "Hindi"]

}

* Java JSON Data binding :-

→ Data binding is a process of converting JSON data to a Java POJO.

* JSON Data binding with Jackson :-

Spring uses Jackson project.

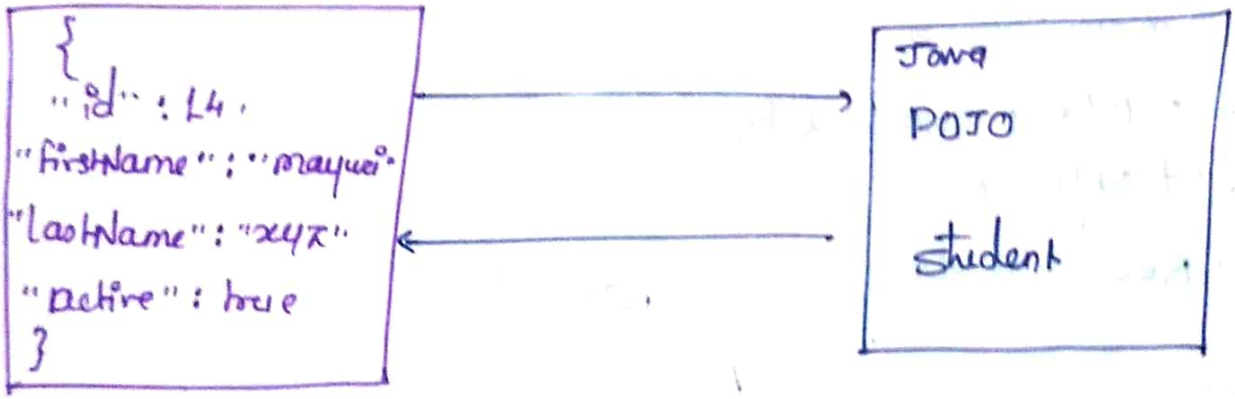
Jackson handles data binding bet^n JSON and POJO

Details :-

http://github.com/FasterXML/jackson-databind

Jackson supports XML and JSON.

By default, Jackson will call appropriate getter & setter method



* JSON to POJO → Call Setter method on POJO
* POJO to JSON → Call getter method on POJO

→ Rest over HTTP

→ use of Rest over HTTP,
HTTP methods for CRUD appln.

	HTTP methods	CRUD operations
1)	POST	Create a new entity.
2)	Get	Read a list of entities or single entity.
3)	put	Update an existing entity.
4)	Delete	Delete an existing entity

* HTTP Response - Status code :-

Code Range	Description
100 - 199	Informational
200 - 299	Successfull
300 - 399	Redirection
400 - 499	client error
500 - 599	Server error.
401	Authentication required
404	file not found,
500	Internal Server error.

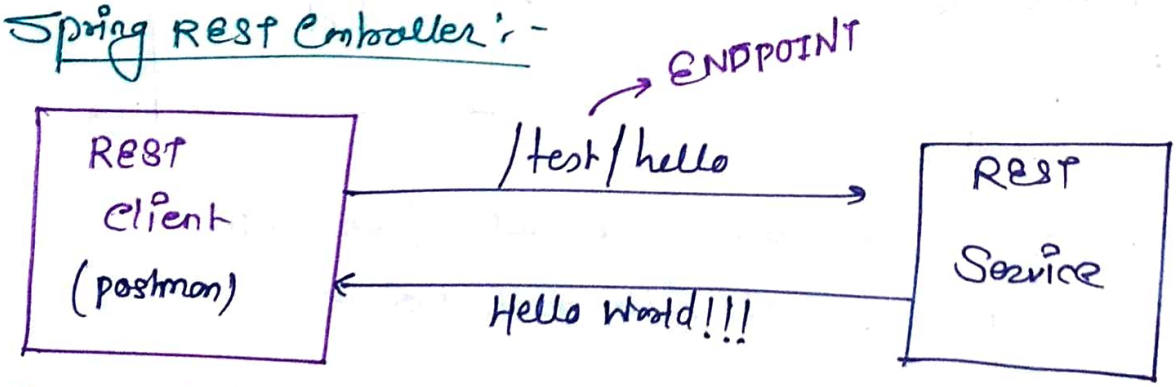
* Client tool :-

Send Http request to REST web API.
We need a client tool - Postman.

MIME - Content Types :-

Basic Syntax : type/sub-type
Ex : text/html

* Spring REST Controller :-



* REST Controller :

→ @RestController → Handles REST requests and Responses.
→ Spring web mvc provide Support for Spring REST.
→ Spring Rest will automatically Convert JAVA POJO's to JSON.
→ As long as the Jackson Project is on the classpath or pom.xml

1.e Modules spring ⇒ http://docs.spring.io
(check diagram)

→ RestControllerDemo1

DemoAppConfig.java

```
package com.app.springdemo.config;

@Configuration
@EnableWebMvc
@ComponentScan
```

Same as previous program from laptop ⇒ RestAPI

→ Student.java

```
package com.app.springdemo.entity;
public class Student {
    public String firstName;
    ——— lastName;
//Generate all.
```

→ StudentRestController.java

```
package com.app.springdemo.rest;
@RestController
@RequestMapping("/api")
public class StudentRestController {
    List<Student> theStudents;
    public StudentRestController() {
        theStudents = new ArrayList<Student>();
    }
```

@PostConstruct public void loadData() {
theStudents = new ArrayList<Student>();
theStudents.add(new Student("rahul", "pawar"));
theStudents.add(new student("aish", "pati"));
the.

@GetMapping("/students")
public List<Student> getStudents() {
theStudents.add(new Student("rahul", "pawar"));
——"——(new student("Amey", "Lok"));
——"——(new Student("xyz", "pqr"));
return theStudents;
}

@GetMapping("/students/{studentID}")
public Student getStudent(@PathVariable int studentID) {
return theStudents.get(studentID);
}

// http://localhost:8080/RestControllerDemo/api/students/1

* Spring Boot *

Java 8 → javax package

After JDK8 → Jakarta package

→ Goto SpringInitializer on browser ⇒
Project ⇒ maven
Spring boot ⇒ 3.1.6
packaging ⇒ Jar
Java ⇒ 17
Add ⇒ dependency ⇒ Spring web
Generate → zip file → extract all

→ Eclipse → maven → expand → existing maven project → add
☑ Projects to working set

@SpringBootAplication ⇒ Combination of 1) @AutoConfiguration → to enable springboot auto config.
2) @Configuration.
3) @ComponentScan
To register extra beans in project (@Bean)
packages to Scan for Components.

→ src/main/resources ⇒
→ Static folder ⇒ Html/css/JS files --etc.
→ templates folder ⇒ thymeleaf → templating engine.

→ src/test/java ⇒ In test folder → create all test classes by using @test annotation to perform unit & integrating testing

→ target folder ⇒ Jar files will be generated

Springboot Comes with many starter project one of is:-
→ Spring-boot-Starter-parent Version 3.1.6 so dependency add to our project will take its parent Version 3.1.6

→ There is property tag where you can change Java Version ⇒ 17

→ Open → eclipse →

→ Add dependency Spring boot DevTools always so we don't need to run a project again & again after committing source code.

→ src/main/ resources

→ Application.properties

→ Add dependency actuator.

→ Springboot 2 project.

* DemoRestController.java

```
package com.java.rest;

@RestController
public class DemoRestController {
    @Value("${trainee-name}")
    private String traineeName;
    @Value("${trainer-subject}")
    private String trainerSubject;

    @GetMapping("/hello")          → Endpoint
    public String sayHello(){
        return "Time on server = " + LocalDateTime.
                                     now();
    }

    @GetMapping("/trainer.info")   → Endpoint.
    Public String trainer
                    Info(){
        return "trainer = [" + traineeName + " " +
                               trainerSubject + "]";
```

→ application.properties

```
server.port = 8081
trainer-name = Amey
trainer-Subject = Java.
server.servlet.context-path = /app
```

→ To see context path.

Run on Server

Enter URL on browser ⇒ http://localhost:8081/app/hello

* Spring Boot =>

provides =>

1) Spring Initializr => http://start.spring.io

Quickly create a starter project

Select your dependancies.

Create a maven project.

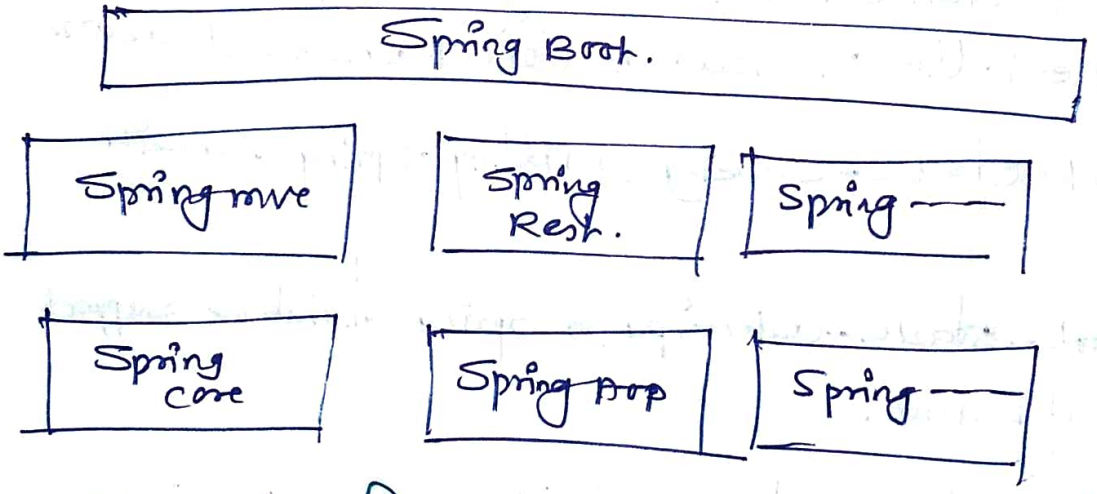
Import the project into your IDE.

2) Spring Boot Embedded Server.

3) Running Springboot Appln :-

Standalone.

4) Deploying SpringBoot Appln :-



* Maven Pom file :-

Spring Boot Starter (A collection of maven dependancies)

(Compatible Version) ⟶ Spring web, Spring web mvc.....

Import =>

Org.springframework.boot.autoConfigure.SpringBootApplicatio

```
@SpringBootApplication
    public class MyDemoApplication {
        main method {
SpringApplication.run(MyDemoApplication.class, args);
    }
}
```

@SpringBootApplicate (scanBasePackages = {})
to add package name need to read other than Base package.

→ There are 30+ Spring Boot starters :-

1) SpringBoot-starter-web ⇒ Building web Apps, includes Validation, REST. Uses Tomcat as default embedded server.
2) Spring-boot-starter-Security ⇒ Adding Spring Security Support.
3) Spring-boot-starter-data-jpa ⇒ Spring database support with JPA & Hibernate.

* Spring Boot Actuator :- Add dependency
  Exposes endpoints to monitor & manage your appln.

* Spring-boot-starter-actuator
  Endpoints are prefixed with : /actuator
  1) /health ⇒ Health info about your appln
  2) /info ⇒ Info. about your project

put in web/browser: http://localhost:8080/actuator/info
" " " /beans
" " " /health

→ application.properties :-

1. management.endpoints.web.exposure.include = *
2. info.app.name = MySpringBootApplication.
3. info.app.description = Actuator Demo
4. info.app.version = 2.7.12
5. management.info.env.enabled = true

} Springboot 3

* to get endpoints & dependencies created.

1) Add Spring-boot-Starter-actuator.
2) Add Spring-boot-Started-web.
3) Add Spring-boot-devtools.

Help to prevent project info. from Evade.

→ Springboot 4. 1) actuator 2) web 3) devtools 4) Spring Security
→ add these 4 dependencies.

<Version>2.7.12</version>
<java.Version>1.8</java.Version.
<maven-jar-plugin-version>3.1.1</maven-jar-plugin-Versin

{ By Default username is ⇒ user
& password is present in Console }

http://localhost:8080/Login

→ application.properties

1
2
3
4 } Same as prev. project Springboot 3
5 Spring.Security.user.name = admin
6 " " user.password = admin123

} When we don't want auto generated username & password

right click on project → build path → configure BP → remove JRE → add library → select system JRE library → next → Installed JRE's → select current one & remove it → add → standard Vm → next → goto directory → program files → Java → JDK → bin → select folder → Apply & close → finish.

→ To change JRE to JDK from build path.

Or

right click on project → run as → 3. maven build → Goals : package write there → Run → msg on Console Build Success → then refresh project → goto target folder → expand → Jar files are generated.

right click on project → properties → left side Resource → Location of project there is 1 button hit on that button → Show in System explorer we get location → select on path → type cmd → Enter.

In Cmd → cd SpringBoot2

→ cd target

→ java -jar SpringBoot002-0.0.1-SNAPSHOT.jar

put name of Jar present in target folder

http://localhost:8081/app/hello

* What is Rest Endpoint in Spring?

→ The Rest Endpoint makes it possible for a web appln to respond to HTTP Rest Requests. Therefore, if a user goes to the browser & Enters a URL, we want to tell Spring to run a specific method & send the return value to user.

* Spring data JPA : - (Java persistence API)

→ The problem ;-

- We Saw how to create dao for Employee.
- What if we need to create a Dao for another entity? Student, Customer, Product - - -
- Do we have to repeat, all the Same code again?

```
@Override
public Employee findById(int theId){
  Employee theEmployee = entityManager.find(Employee.class, theId);

  return Employee;
}
```

Entity type — Primary key

My wish @

→ I Could tell Spring
→ Create a dao for me.
→ Plug in my Entity type and primary key.
→ Give all the basics CRUD features for free.

Solution @

- Spring data JPA (https://spring.io/projects/spring-data-jpa)
- Create a dao for me..
- Plug in my entity type & primary key.
- Give all the basic CRUD features for free.
- More than 70% of reduction of code.

JPARepository - Interface

Entity type — Primary key ID type

JPARepository<Student, Integer>

Developement process ;-

→ Extends JPARepository.
→ Use your repository in your app - No need for implementation class.

* Spring data JPA ⇒ CRUD methods free No Restpoints free.

→ maven
→ 3.1.6
→ Artifact-name ⇒ demo
→ Package ⇒ .
→ dependencies ⇒ Web
⇒ devtools
⇒ Data-JPA
⇒ MySQL-Driver

} Spring Initializer or STS

→ Application.properties :-

Spring.datasource.url = jdbc:mysql://localhost:3306/employee-directory.
?useSSL=false&serverTimezone=UTC
↑ DB Name

Spring.datasource.username = root

Spring.datasource.password = Amey@1ok

→ We are saving a Customer ⇒

Select post :- → create a new entity
http://localhost:8080/api/employees
Select Body → raw → JSON

} IN Postman

@RequestBody ↑ Annotation used in addEmployee method in Controller

{
"firstName" : "Rahul"
"lastName" : "Pawar"
"email" : "Rahul@gmail.com"
}
Press Send button.

{ JSON → JavaScript object Notation }

Id Specified → update
Id not Specified → save (New Entry created)

for customise query we have to use @Query annotation

→ to get a Single Entity **Get**

http://localhost:8080/api/employees/1 → path Variable.

→ Body → raw → JSON

2 → for Multiple

2 → for only one record.

→ to update a employee/record by ~~Id~~

~~Put~~

http://localhost:8080/api/employees

```
{
  "id": 3,
  "firstname": "Amey",
  "lastname": "Dahake",
  "email" : "Amey@gmail.com"
}
```

→ to Delete a record by id **DELETE**

http://localhost:8080/api/employees/3

→ **GET** To Count a records

http://localhost:8080/api/employees/count

Project En Laptop →

* Spring Data REST :- CRUD & Restpoints both one free.

Problem :

We saw to create REST API for Employee (EmployeeRestController. @RestController)

Need to create REST API for another Entity.

Do I have to repeat the Code

→ Create REST APIs for me,

→ Use my existing JPARepository (Entity type, primary key)

Give me all the basics REST API CRUD features for free.

Soln :- Spring data REST

→ Spring data REST will expose endpoints for free

	HTTP Method	Endpoint	CRUD Action
1)	POST	/employees	Create a new Employee
2)	GET	/employees	Get list of all employees
3)	GET	/employees/{empID}	Read a single entity
4)	PUT	/employees	Update existing entity
5)	delete	/employees/{empID}	Delete existing entity.

* Spring data Rest will Scan your project for JPA Repository.

→ RestEndpoints By default, Spring data REST will create endpoints based to entity type.

Simple pluralized form

→ employee /employees student /students.

→ pom.xml

add dependency ⇒ Spring-boot-starter-data-rest → all free endpoints.
→ all free CRUD methods.

your entity : Employee

⇒ EmployeeRepository that extends JPARepository.
⇒ Add this dependency. → Interface

{ Present in dependency data-JPA }

* cruddemo ⇒

→ dependency to add ⇒ 1) Rest Repositories
2) web
3) Devtools
4) Data JPA
5) MySQL Driver.

→ application.properties :-

1 → url
2 → username
3 → password

} same as previous demo

4 // Spring data REST properties
5 Spring.data.rest.base.path = /api

{ 201: created — request has been fulfilled & record has been created
201: ok }

{ 204: No Content — Empty }

→ @Repository on EmployeeRepository then ⇒

Get ⇒ all records ⇒
http://localhost:8080/api/employees → fixed form

Get - Single record (Entity) ⇒
http://localhost:8080/api/employees/1

→ Post ⇒ add an Entity

http: localhost : 8080/api/ employees

Request Body :-

```
{
  "firstName": "priya",
  "lastName": "Deshmukh",
  "email" : "priyanka@gmail.com"
}
```

put – update an Entity

http: localhost: 8080/api/employees/1

Request Body :-

```
{ "firstName": "priyanka",
  ——————————————
  ——————————————
}
```

same as previous.

delete – delete an Entity

http: // localhost: 8080/api/ employees/1

---

* @RepositoryRestResource(path = "members") { Whereever there is Employee it will replace with members

on EmployLayee Repository then

→ Get → all records

http :// localhost: 8080/api/members

Get- single record (Entity)

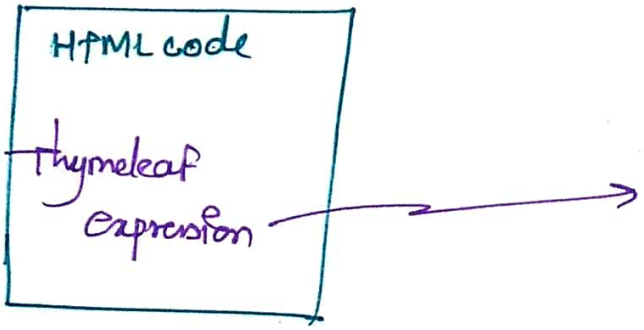
http: // localhost : 8080/api/ members employees/1

* Springboot Thymleaf ⇒ used for both web & non web environment.

→ Java templating engine.

→ Generate HTML Views for web apps.

www.thymleaf.org



→ Can access java code, objects, Spring beans.

* In a web app, Thymleaf is processed on a server. Results included in HTML returned to browser.

Thymeleafdemo ⇒

→ add dependency ⇒ 1) Thymeleaf
2) Web
3) Devtools

→ Employee.java

```
Package com.app.springboot.Thymeleafdemo.model;
public class Employee {
    private int id;
    Private String firstName;
    ← lastname
    ← email;
```

// Generate all.

→ DemoController.java

```
package com.app.springboot.thymeleafdemo.controller;

@Controller
public class DemoController {
    @GetMapping("/")
    public String sayHello(Model theModel) {
        theModel.addAttribute("theDate", new java.util.Date());
        return "helloworld";
    }
}
```

→ helloworld.html

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>Thymeleaf Demo</title>
</head>
<body>
    <p th:text="'Time on Server is ' + ${theDate}" />
</body></html>
```

* spring-mvc-form-tag-validation ⇒ with hibernate validator

dynamic web project

Jar files to add

⇒ hibernate validator 6.2.5 final → dist → copy 3 jar files & paste.

⇒ dist → lib → required → copy all 4 files & paste it.

⇒ JSP
⇒ JSTL
⇒ Servlet.
⇒ all Spring Jar files RELEASE.

⟶ dependency to add ⇒

Hibernate Validator Engine Relocation Artifact ⇒ 6.2.3 final

⟶ dependency add ⇒ Apache Commons IO ⇒ 1.3 version

Jar files can also download { ⇒ Apache Commons file upload ⇒ 1.3 Version

⇒ Jackson. Databind ⇒ 2.7.6 Version

⇒ Context ⇒ web ⇒ expression

⇒ core ⇒ beans ⇒ Servlet ⇒ JSP

add jar files this.

In database

✱ JDBC → right click on Stored procedure → create Stored procedure

Que. Do In and out works with function ? yes

⟶ right click on fn and create function →

⟶ new_procedure

In DB ⇒ CREATE PROCEDURE 'new_procedure'(IN id integer, IN name Varchar(30), IN salary float)
BEGIN
insert into employee values (id, name, salary);
END;

⟶ procedure 1

```
CREATE PROCEDURE 'procedure1'
(IN eid Integer, OUT ename Varchar(30),
OUT esalary float)
BEGIN
Select name, salary into name, esalary
from employee where id = eid;
END
```

→ new_function

```
CREATE FUNCTION 'new_function'
(n1 integer, n2 integer) RETURN int
BEGIN
 declare temp integer;
 set temp = n1+n2;
 RETURN temp;
END
```