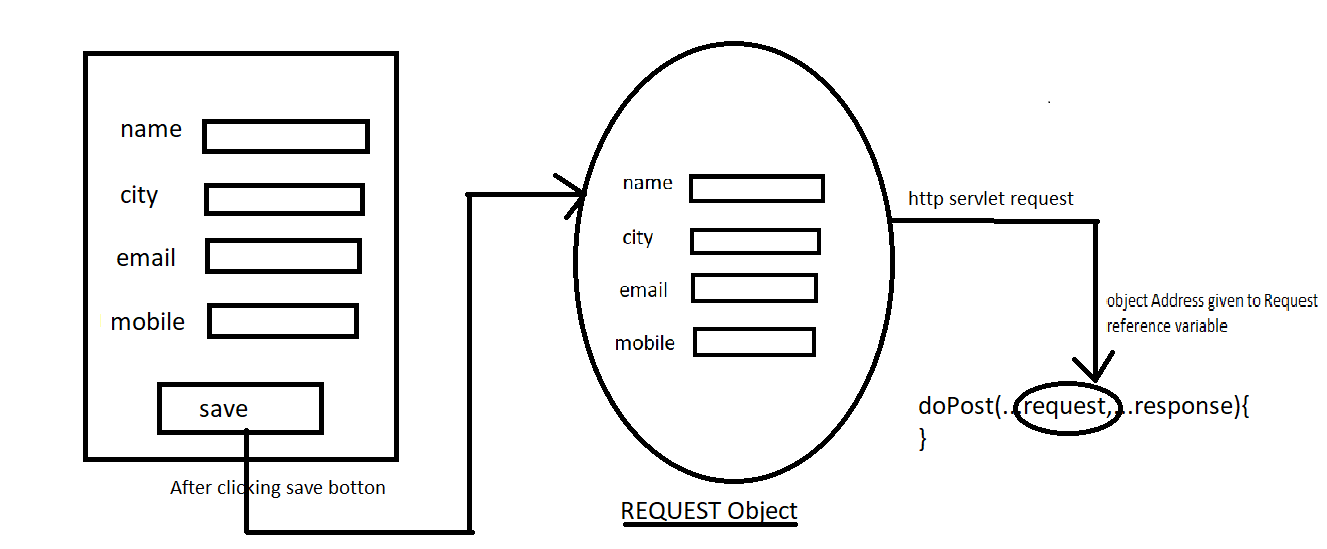
ADVANCE -JAVA-THEORY

ADVANCE JAVA

1. Steps to create web application(without SQL & JDBC):

* Step 1 (installing tom cat):
  + New 🡪 others→ servers → Apache→ tomcat 9
  + Download tomcat → download zip→ unzip
  + Select tomcat on specified version in eclipse→ browse → tomcat → open until bin is found→ select → java jdk 1.8→ finish
  + Right click on server in bottom down section near console and start the server, or just click green play button.
* Step 2(dynamic web project):
  + Right click → dynamic web project instead of java project →name → web\_app\_1→ dynamic web module version (by default 4 if it wont work use 3.1).
  + Always create html files in src/main/webapp → new → other→ html→ file name→ (newRegistration)→ finish.
* Step 3(servlet):
  + Src/main/java → right click → others → servlets→ name→ (newRegistration)→ delete all comments.
  + Note: always use tomcat 9 in 2021 version as there are some problems with ver 10.
  + Note: @WebServlet("/newRegs") this should be same as that of the name which is present in form action.
  + Note: request is a reference variable which points to request object where all the data is stored as mentioned in the form.
  + Note: in html we use name attribute in input tag as it is similar to variables it stores variables and those can be used in backend.
  + Note: request.getParameters(“(city→ as it is written in name attribute in input tag inn html)”)
  + Here we use post method to send data to the sql so write all these code in post method in servelets.
  + Note: constructing login page using tables→ form→ table→ (tr→ td→td)\*4

1. Concept of POST/Get Request:

* Request is a reference variable where it points to request object where all the data is stored (i.e., the name attribute things with input present in html file).
* Serialization implementation is happening in the background.
* Concept:
* 

1. Steps to create web application(without SQL & JDBC):

* Step 1 (installing tom cat):
  + New 🡪 others→ servers → Apache→ tomcat 9
  + Download tomcat → download zip→ unzip
  + Select tomcat on specified version in eclipse→ browse → tomcat → open until bin is found→ select → java jdk 1.8→ finish
  + Right click on server in bottom down section near console and start the server, or just click green play button.
* Step 2(dynamic web project):
  + Right click → dynamic web project instead of java project →name → web\_app\_1→ dynamic web module version (by default 4 if it wont work use 3.1).
  + Always create html files in src/main/webapp → new → other→ html→ file name→ (newRegistration)→ finish.
* Step 3(SQL Query):
  + Create sql database in MYSQL
  + Create database 4pm\_demo\_db\_1;
  + Use 4pm\_demo\_db\_1;
  + Create table registration(FirstName varchar(20), city varchar(20), email varchar(20), email varchar(20), mobile varchar(10));
  + Select \* from registration;
  + insert into registration values('Pankaj', 'Bangalore', 'pankaj@gmail.com', ‘9496415549’);
* Step 4 (Copying SQL connector):
  + Copy SQL connector file and paste in src/main/webapp/web INF/lib.
  + Note: always write 🡪 Class.forName(“com.mysql.jdbc.Driver”); before establishing connection.
* Step 5(servlet):
  + Src/main/java → right click → others → servlets→ name→ (newRegistration)→ delete all comments.
  + Here use same Connection 🡪 Statement 🡪 request.getParameters(“(\_)”)→ assign it to variable and use ‘”+var name+”’ to give into SQL Query inside stmt.updateQuery(“”);
  + Note: always use tomcat 9 in 2021 version as there are some problems with ver 10.
  + Note: @WebServlet("/newRegs") this should be same as that of the name which is present in form action.
  + Note: request is a reference variable which points to request object where all the data is stored as mentioned in the form.
  + Note: in html we use name attribute in input tag as it is similar to variables it stores variables and those can be used in backend.
  + Note: request.getParameters(“(city→ as it is written in name attribute in input tag inn html)”)
  + Here we use post method to send data to the sql so write all these code in post method in servelets.
  + Note: constructing login page using tables→ form→ table→ (tr→ td→td)\*4

1. Servlets:

* Servlets is a java class.
* It is a subclass of HTTP servlets; it is used to perform Back-end coding of the application.

1. Note: Whenever we run servlet it will always execute get method.
2. Note: whenever we want to retrieve data from database then we should always use updatequery method and 🡪assign to local reference variable 🡪 PrintWriter out = response.getWriter(); 🡪 out.println() to write an html code to display in the front end.
3. Note: during validation of sql data like login page we use SQL Query i.e.,🡪 select \* from login where email=’”+email+”’ and password=’”+password+”’.
4. Note: anything inside WEB-INF folder can only be run by servlet 🡪 request.getRequestDispachther(“location with /”) 🡪 rd.forward(request, response) – if we want to go to given page, if we want to same or backward page use rd.include(request, response);🡪if its run directly it will give 404 error.
5. Note: index.html → is the first page of html always , even if we don’t write index.html in web link also it will point to index.html as it is default start/home page.
6. Inter Servlet Communication:

* ISC: When one servlet is calling another servlet using request dispacther is called as inter servlet communication.
* Request.setAttribute & .get Attribute will work only when request dispatcher concept is used.
* Call back methods🡪 main method, doGet method, doPost method etc. that’s the reason why servlet wont have any main methods as they consists of doGet and doPost methods.
* .setAttribute are just used for short time that is temporary memory, whereas to store values for longer time we use session variable concepts.
* If request dispatcher is called from post method of first servlet then it will call 2nd servlet post method only.
* Session should be used in ecommerce app where logic of add cart etc where items should be stored for longer time.
* As long as history/cookies are not deleted, session will not be deleted.
* Session: once the data is stored in session variable then that data can be remembered across the application.

1. JSP: Java server Page:

Only

HTML

CSS

JavaScript

Html

Java

.html .jsp

1. Script-Let Tag<% %>:

* Advantages:
  + 1)Implicit objects can be created.
  + 2)request
  + 3)response
  + 4)session
  + 5)out Object etc.
* Disadvantages:
  + 1)cannot create variable with access specifiers.
  + 2)Cannot build any methods in it.

1. Declaration Tag<%! %>:

* Advantages:
  + 1)can create variable with access specifiers.
  + 2)Can build any methods in it.
* Disadvantages:
  + 1)Implicit objects cannot be created.
  + 2)request cannot be created.
  + 3)response cannot be created.
  + 4)session cannot be created.
  + 5)out Object cannot be created etc.

1. Expression Tag<%= %>:

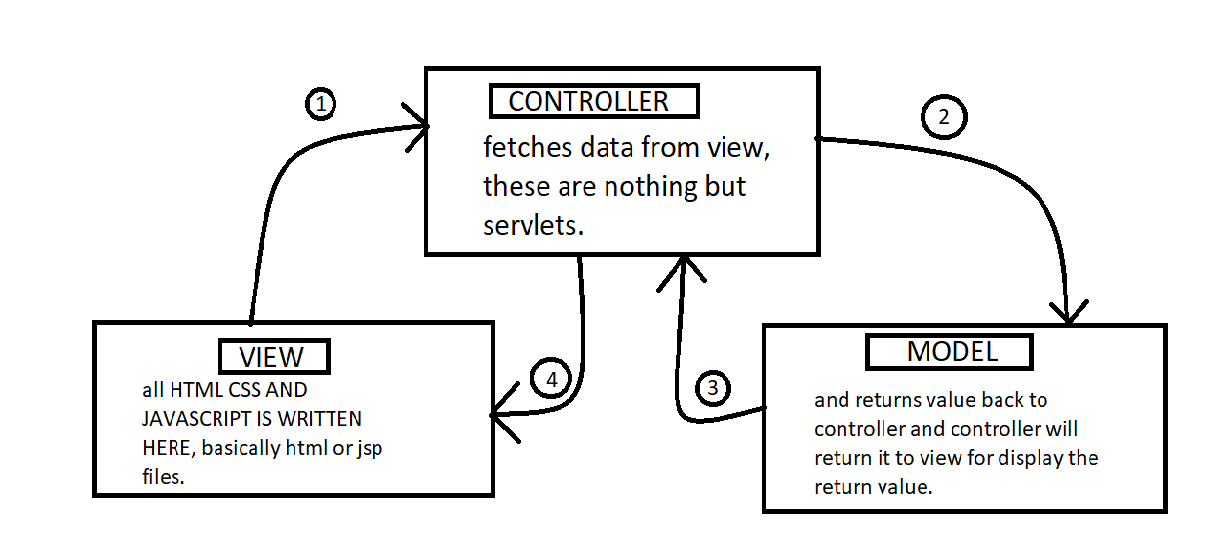
* It is generally like out.println() implicit jsp objects.
* We can write only one line of code in it.
* We don’t have to use any semicolon inside.

1. Directive Tag<%@ %>:

* 2 Types:
  + Page Directive Tag<%@page import=”” %>:
    - Basically, to write all imports like, file class, package imports, scanner class imports all these should be written in page directive tags.
  + Include Directive Tag<%@ include file=” any .jsp or .html or any txt files or etc.” %>:
    - Basically, it is used to import or include other or same extension files into jsp.

1. MVC Architecture:

* Flow chart:



* VIEW:
  + All HTML, CSS & JS are written inside view, either in html file or in jsp file.
* CONTROLLER:
  + Fetches data from view and will give to MODEL.
  + These are nothing but servlets.
* MODEL:
  + Will perform business operation and returns value back to controller and controller will return back top view to display it in front end.
  + All databases code will also be written here, including business logic.

1. Mini-Project:

* Day-1:
  + Use mvc Architecture to create project- i.e., create all visual part in view and all connections in controller and all business and database logic in model part i.e., by using industry standards such as inheritance polymorphism Runtime polymorphism and much more by using interface then connecting to class in model layer.
* SQL:
* Create databases studnet\_web\_app\_4pm and create 2 tables login and registration with login consisting of 2 columns email and password and registration table of 4 columns name, city, email, password.
* View:
* Index.jsp:
  + Create form containing email and password and put a submit button with value login. And form action as “loginController” and post method.
* Controller:
* Login Controller.java:
  + Create a package:
  + @Webservlet(“/loginController”)
  + In dopost()🡪 take values from jsp file using request.getParameter(“email”); etc and assign it to the values and create a DAO.java interface.
  + Create class upcasting of both DAO and DAOImpl and call connectDB() and verifyLogin(with email and password).
  + Then got to Model i.e., DAO.java.
  + After taking return value from verifyLogin() assign it to a variable and write a logic with if and else condition to print welcome and in else part invalid.
* Model:
* DAO.Java:
  + This is an interface, i.e., all logical blueprint is written here.
  + Create a method to connect database connectDB();
  + Create a method to verifyLogin(String email, String password);
* DAOImpl.java:
  + First create private initialization of Statement stmt and Connection con;
  + Here override the interface incomplete methods.
  + In connectDB(){} write a logic to get a connection of sql i.e., till stmt=……….;
  + In verifyLogin() method write a logic using sql to verify weather email and password are matching as registered, and make return type as Boolean, go back to controller.
* Day-2:
  + Use mvc Architecture to create project- i.e., create all visual part in view and all connections in controller and all business and database logic in model part i.e., by using industry standards such as inheritance polymorphism Runtime polymorphism and much more by using interface then connecting to class in model layer.
* SQL:
* Create databases studnet\_web\_app\_4pm and create 2 tables login and registration with login consisting of 2 columns email and password and registration table of 4 columns name, city, email, password.
* View:
* Index.jsp:
  + Create form containing email and password and put a submit button with value login. And form action as “loginController” and post method.
  + Here use scriptlet tag to get value set in else part to print the message.
  + i.e., put that logic in if condition so that it wont print null in front-end.
* newRegistration.jsp:
  + Here we create a form having name, city , email, mobile and submit button with register value in it.
  + And after value is stored make it to print “Data registered successfully…” using request dispatcher and include.
  + And add a menu at the top with new registration and show registration options i.e., by using directive tag – include tag of file=menu.jsp.
* Menu.jsp:
  + Write an <a href=”#”>New Registration</a>
  + Write an <a href=”#”> Show Registration</a>
* Controller:
* Login Controller.java:
  + Create a package:
  + @Webservlet(“/loginController”)
  + In dopost()🡪 take values from jsp file using request.getParameter(“email”); etc and assign it to the values and create a DAO.java interface.
  + Create class upcasting of both DAO and DAOImpl and call connectDB() and verifyLogin(with email and password).
  + Then got to Model i.e., DAO.java.
  + After taking return value from verifyLogin() assign it to a variable and write a logic with if and else condition where inside if that is if its true then we have to use request dispatcher to make it to forward to newRegistration.jsp file.
  + And in else part write a set.Attribute() value and write and message and make it to go to index.jsp.
* Registrationcontroller.java:
  + Create a package:
  + @Webservlet(“/loginController”)
  + In dopost()🡪 take values from jsp file using request.getParameter(“name”); etc and assign it to the values and create a DAO.java interface.
  + Then got to Model i.e., DAO.java.
  + Create an Class upcasting object of both DAO and DAOImpl
  + After that put if condition where name.length()!=0 &&…..all 4 parameters and call connectDB method using objects reference variable and then create an registerData(with all 4 parameters);
  + And put request dispatcher to send registration done succesfull message using include method.
  + In else method create another request dispatcher with entered invalid values so that no null values are stored in database.
  + Go to DAO.java interface.
* Model:
* DAO.Java:
  + This is an interface, i.e., all logical blueprint is written here.
  + Create a method to connect database connectDB();
  + Create a method to verifyLogin(String email, String password);
  + Create a method to register values into sql of page newRegistration.jsp, with method registerData(with a 4 parameters);
* DAOImpl.java:
  + First create private initialization of Statement stmt and Connection con;
  + Here override the interface incomplete methods.
  + In connectDB(){} write a logic to get a connection of sql i.e., till stmt=……….;
  + In verifyLogin() method write a logic using sql to verify weather email and password are matching as registered, and make return type as Boolean, go back to controller.
  + In registerData();🡪 write sql to update value into sql DB using executeUpdate method with return type void.
* Day-3:
  + Use mvc Architecture to create project- i.e., create all visual part in view and all connections in controller and all business and database logic in model part i.e., by using industry standards such as inheritance polymorphism Runtime polymorphism and much more by using interface then connecting to class in model layer.
* SQL:
* Create databases studnet\_web\_app\_4pm and create 2 tables login and registration with login consisting of 2 columns email and password and registration table of 4 columns name, city, email, password.
* View:
* Index.jsp:
  + Create form containing email and password and put a submit button with value login. And form action as “loginController” and post method.
  + Here use scriptlet tag to get value set in else part to print the message.
  + i.e., put that logic in if condition so that it wont print null in front-end.
* newRegistration.jsp:
  + Here we create a form having name, city , email, mobile and submit button with register value in it.
  + And after value is stored make it to print “Data registered successfully…” using request dispatcher and include.
  + And add a menu at the top with new registration and show registration options i.e., by using directive tag – include tag of file=menu.jsp.
  + In doGet() method write requestDispatcher to go to newRegistration.jsp using include method.
* Menu.jsp:
  + Write an <a href=”newRegs”>New Registration</a>
  + Write an <a href=”showRegs”> Show Registration</a>

showRegistration.jsp

* + here we import result set from sql package.
  + Import menu.jsp
  + Here we create table and use 4 columns.
  + Then scriptlet tag we use to take value from the showRegistration and display it using while loop.
  + Write another column in both header and in in table delete cell with link to delete using a- attribute.--> delete?emailId=<%=res.getString(3)%>
  + Same way we can write update.--> Update?emailId=<%=res.getString(3)%> & emailId=<%=res.getString(4)%>

UpdateRegistration.jsp:

* + Here we put form to email and mobile and using request dispatcher we take value from sql and show it in front end update.
* Controller:
* Login Controller.java:
  + Create a package:
  + @Webservlet(“/loginController”)
  + In dopost()🡪 take values from jsp file using request.getParameter(“email”); etc and assign it to the values and create a DAO.java interface.
  + Create class upcasting of both DAO and DAOImpl and call connectDB() and verifyLogin(with email and password).
  + Then got to Model i.e., DAO.java.
  + After taking return value from verifyLogin() assign it to a variable and write a logic with if and else condition where inside if that is if its true then we have to use request dispatcher to make it to forward to newRegistration.jsp file.
  + And in else part write a set.Attribute() value and write and message and make it to go to index.jsp.
* Registrationcontroller.java:
  + Create a package:
  + @Webservlet(“/loginController”)
  + In dopost()🡪 take values from jsp file using request.getParameter(“name”); etc and assign it to the values and create a DAO.java interface.
  + Then got to Model i.e., DAO.java.
  + Create an Class upcasting object of both DAO and DAOImpl
  + After that put if condition where name.length()!=0 &&…..all 4 parameters and call connectDB method using objects reference variable and then create an registerData(with all 4 parameters);

UpdateRegistrationController.java:

* + Write in doget method.
  + Write dopost to update

DeleteRegistrationController.java:

* + Doget to delete record.
* Day-4&5:
  + Put session and validate the protection to all the controllers.

logoutController:

* + anything we click here will take to login page.

1. GET v/s POST:

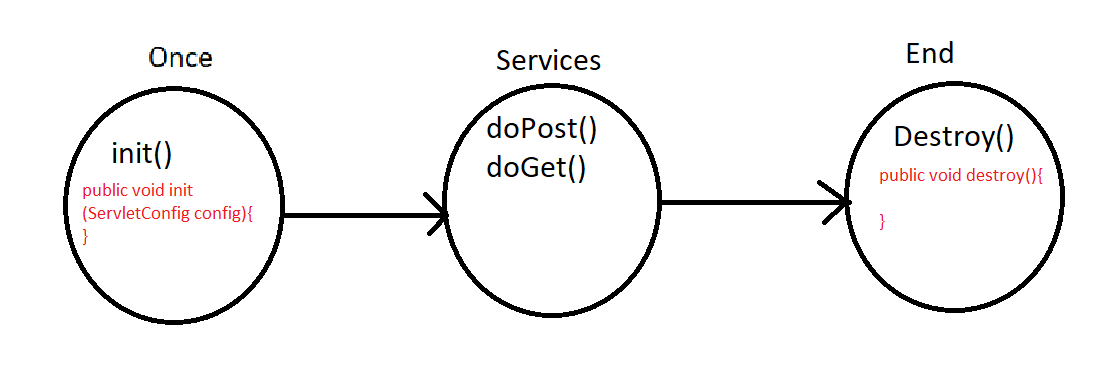
* Differences:

|  |  |
| --- | --- |
| GET | POST |
|  |  |
| * Should be used when we want to get data from database. | * Should be used when we submit data using forms to the database. |
| * Data is Exposed in the URL. | * Data is not Exposed in the URL. |
| * URL data is exposed in browser history | * URL data is not exposed in browser history |
| * When you refresh page you will not get security alert!. | * When you refresh page you will get security alert!. |

1. Application-servlets v/s Web-Servlets:

* Application-servlets:
* If you want to build Dynamic web-Applications using technologies such as servlets, jsp, Hibernate, Spring, Spring-Boot etc. in this case to run these technologies we use Application-Servlets such as Tomcat, J-Boss, Glass-Fish.
* Web-Servlets:
* To build static App we use Technologies like HTML, CSS, JS & to run this we use Webservers like IIS, Apache.

1. Servlet Life Cycle:



* For the first time when we start the server tomcat init() method will run, post that any number of times we can call doGet() and doPost() method, finally when the destroy method is called servlet life cycle comes to an end.
* Note:
  + Code to connect to DB we write itr in init(){} method.
  + Code to close Connection to DB we write it in destroy(){} method.
  + All other code of business logic is written inside doPost() and doGet() methods.

PROGRAM-TOPICS:

* Core java SQL Injections:

1. Inserting data into SQL (normal)
2. Deleting data from SQL (normal)
3. Updating data from SQL(normal)
4. CRUD- all 4 operations using scanner class and using ‘”+variable name+”’ 🡪 Note: during retrieving of data use while loop with condition as result.next() and sysout(result.getString(1---n)).

* Advance Java Without SQL:

1. Normal HTML 🡪 servelet 🡪 String name =request.getParameter(“name”);🡪 print entered values into console.

* Advance Java with SQL:

1. Registration (SQL, HTML & Servlet)🡪 Class.forName(“com.mysql.cj.jdbc.Driver”);
2. update (SQL, HTML & Servlet)🡪 Class.forName(“com.mysql.cj.jdbc.Driver”);
3. delete (SQL, HTML & Servlet)🡪 Class.forName(“com.mysql.cj.jdbc.Driver”);
4. Retrieving Data and printing into html table 🡪 use updateQuery and assign it to local reference variable 🡪 PrintWriter out = response.getWriter(); 🡪 out.println() to write an html code to display in the front end.
5. Login page simple to print welcome If valid and invalid if its wrong into console🡪 during validation of sql data like login page we use SQL Query i.e.,🡪 select \* from login where email=’”+email+”’ and password=’”+password+”’.
6. Login page where Index (login page)🡪servlet (login Servlet)🡪index (welcome page), in if-condition write forward code i.e.,

RequestDispatcher rd=request.getRequestDispatcher("WEB-INF/views/welcome.html");

rd.forward(request, response);

and in else condition we write include code i.e.,

RequestDispatcher rd=request.getRequestDispatcher("index.html");

rd.include(request, response);

1. Inter Servlet Communication:

* Create a welcome html page with name and submit button.
* Create a first servlet where store name in reference variable.
* Set the attribute using request.setAttribute(“nameParam”,name); that is, what is there in name variable will be copied to nameParam variable such that it can be used in other servlet.
* Always make sure you write session first then request dispatcher.
* Create session and set it to variable using session.setAttribute(“sessionName”, name); that is, what is there in name variable will be copied to sessionName variable such that it can be used in other servlet and most important it is not an temporary memory like request dispatcher.
* Then write requestdispatcher concept and put second servlet name in it
* Create a second servlet where use request.getSession(); and write a return type and use reference variable of same and use session.getAttribute(“sessionName ”); and type cast it to string type and use it to print into html using response.getWriter concept.
* After that in second servlet write request.getAttribute(“nameParam”); and store it in string return type and use it to print into html using response.getWriter concept.
* Note: always session should come first then requestDispatcher.

1. Login And Logout Page:

* Create a database and store two tables one is of login credentials and other to store of registration values.
* Create a index.html i.e., home/login page. Where email and password attributes are present.
* Create welcome.html page where name email city mobile register and logout is also present.
* Create login servlet i.e., for index.html
  + Here first take both email and password and assign it to variables and create a connection and read sql query for email and password match using and condition in sql.
  + In if condition i.e if result.next() is true then execute session and make session to true.
  + Use request dispatcher and put welcome.html file path and take a return type and use the reference variable to forward the page.
  + In else condition use request dispatcher and give same index.html file path and use include to stay on same page.
* Create registration servlet i.e., for welcome.html
  + Here first take all name city email and mobile store it in variable use session get and set it to false, and put and if condition if session is not equal to null write all code i.e., for getting connection udating data into sql and even request dispatcher and put welcome.html page path and forward it.
  + In else use request dispatcher and put index path and make it to include/forward.
* Create LogoutServlet i.e., for welcome.html
  + Here use get session and make it false and use session.invalidate(); method to delete the session so that it will go to null if we press logout and no data will be registering if session is equal to null as in registrationServlet.
  + Use request dispatcher and give index.html path.
  + And forward it.

1. Practice all concepts of JSP Tags: script-let Tag, Declaration Tag, Expression Tag, Directive Tag (page and include directive Tag).
2. Creating navbar using jsp to communicate between all jsp files using include directive Tag.
3. Adding two numbers using MVC Architecture:

* addNumber.jsp(VIEW):
  + create form to take input of 2 numbers and create a submit button too.
  + After all business logic we should also add script-let tag to fetch data from servlet and display it.
* AddServlet.java (CONTROLLER):
  + Create a package 🡪 com.web\_app\_1.controller 🡪 company.projectname.purpose.
  + In do post method, take the 2 numbers as input using request.getParameter(“”);
  + Later type cast it to integer using Intiger.parseInt(request.getParameter(“num1”)) assign those two to diff variables.
  + Create an object of Add.java(model) and store object address in reference variable.
  + Call a method to add those two numbers and assign it to variable(result).
  + Then request.setAttribute(“res”, result)
  + Then use request.getRequestDispatcher(“addNumber.jsp”); take a return statement.
  + Rd.forward(request, response);
* Add.java (MODEL):
  + Create a package 🡪 com.web\_app\_1. model 🡪 company. projectname. purpose.
  + Create a method of same name which was created in servlet(controller), return num1+num2;

1. MINI-PROJECT 🡪 refer 17th -18th part in theory of advance for detailed explanation.