Advance Java (JEE)

Servlet

JSP

**Web Application**

1. The applications which can be consume from the browser/client. And it is deployed at Server side.
2. These applications are also known as client-server application.
3. Web Applications are of 2 types
   1. Static Web Application
      1. The content of the page is same for every user.
      2. These applications can be build using HTML, CSS, JS, JQuery etc.
   2. **Dynamic web application**
      1. The content of the page is changes as per user.
      2. These application can be build using HTML, CSS, JS, Jquery, JSP, Servlet.

**Client-Server**

* + - 1. Client is a browser from there end user will access the web application via a network.
      2. Server is a hardware and software where applications/code will be deployed.
      3. Servers are of two types
         1. Web Server

It contains the EJB container and Servlet Container.

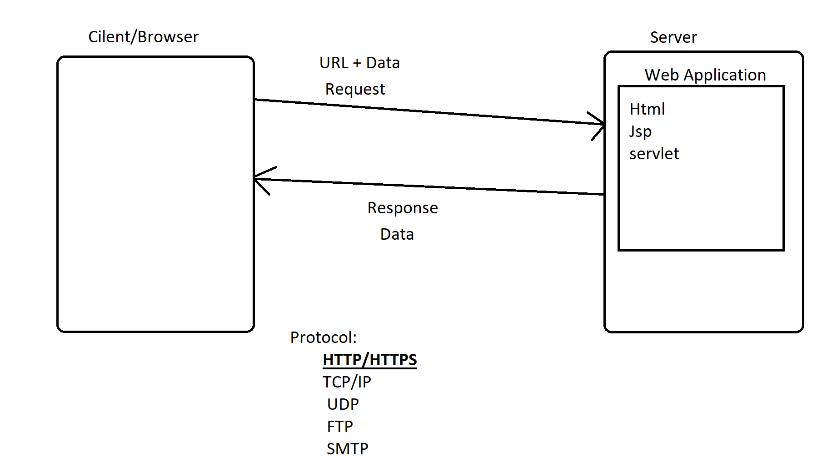
Examples, Web Spare server, JBoss server

* + - * 1. Web Application Server

It contains a servlet container.

Example, Apache Tomcat.

* + - 1. Client and server communicate with each other using Request and Response.



**Server Setup into Eclipse**

1. Download Tomcat Server ZIP file.

<https://tomcat.apache.org/download-90.cgi>

1. Extract a ZIP file into a specific location.
2. Steps to setup server into eclipse
   1. Open an eclipse workspace.
   2. Verify the perspective, it must be Java EE.
   3. Select a Servers tab at the bottom of the screen.
   4. Click On the link which is display under Server section to create new server.
   5. Expand Apache and select appropriate version of the tomcat.
   6. Click on “Next” -> Browse for the server location where you extractor the ZIP file.
   7. Server location must be a parent folder of bin, conf, log
   8. Click On “Finish”
3. To Start server, right click on the server and select “Start” option.
4. Check of “INFO: Server startup in [746] milliseconds” log to verify server started or not.

**Steps to Create Dynamic Web Project**

1. File -> New -> select **Dynamic Web Project**
2. Provide name of the Project
3. Make Sure that Target Runtime is selected to “Apache Tomcat”. It must not ne NONE
4. Click On **Next ->** Click **Next** -> **Checked the check box (Generating Deployment Descriptor web.xml)**
5. Click on **Finish**

**Project Structure**



HTML Reference Link : <https://www.w3schools.com/html/html_forms.asp>

**Servlet**

1. Servlet is a java class.
2. This class executes at server side.
3. Servlets are used to create dynamic web pages.
4. Inside servlet you can use HTML along with Java. Which is also known as HTML in Java.
5. Servlet file extension in .java.
6. Main method is not allowed in the servlet.
7. Servlet is mainly use to **get the request and Data, process a request and generate a response**.
8. In Web application every thing will be access using the URL.
9. Every Servlet has a URL. This URL must be unique so that servlets can access using this URL.
10. Servlet Object will be created and maintain by Servlet Container which is a part of tomcat server.

**Servlet Life Cycle**

1. Servlets executes by a servlet container which is a part/component of server.
2. Servlet will be execute into a specific stages/steps which also known as servlet life cycle.
3. Servlet life cycle will be managed by servlet container.
4. There are 3 steps/stages of servlet lifecycle
   1. Init stage
      1. In this stage servlet object will be create and initialize.
      2. If you wants to perform specific operation at the time object creation then you can use **init(ServleConfig) method** of the servlet class.
      3. As a part of this stage init method will be called by servlet container after constructor.
      4. There is only one object of the servlet through the application and hence init method calls only once in a life cycle.
   2. Service stage
      1. In this stage the request will be processed and response will be generated.
      2. In this stage the service method gets executed.
      3. This method will be called for every request from the user.
      4. This method called multiple times in a life cycle for every use request.
   3. Destroy stage
      1. In this stage the servlet object will be destroyed.
      2. To perform cleanup activities before closing the servlet object, it can be perform inside this stage.
      3. Here destroy() method will be called before deleting a servlet object.
      4. This step executes only once in a life cycle.

**Create and Use Servlet**

1. Servlets can be created by 2 ways
   1. Manual Approach
      1. Create a Java Class without Main method.
      2. The Convert Java Class into Servlet
         1. By implementing **Servlet Interface**
         2. By extending **GenericServlet abstract class**
         3. By extending **HttpServlet abstract class**
      3. Override a **service method** from the Parent Class.
      4. Write Your Business logic inside the service method.
   2. Eclipse Predefine Option.
      1. Right Click on src/main/java -> Go To “New” Option
      2. Select “Servlet” option.
      3. Provide a servlet class name-> Click on “Next” button
      4. Use Existing URL patten or You can Edit it -> Click on “Next” button
      5. Select Appropriate method -> Click on “Finish” button
2. Create URL for the servlet
   1. Every Servlet must have a URL.
   2. URL must be start with “/”
   3. There are 2 ways to provide URL for the servlet
      1. Using XML tags inside web.xml

<servlet>

<servlet-name>First</servlet-name>

<servlet-class>FirstServlet</servlet-class>

</servlet>

<servlet-mapping>

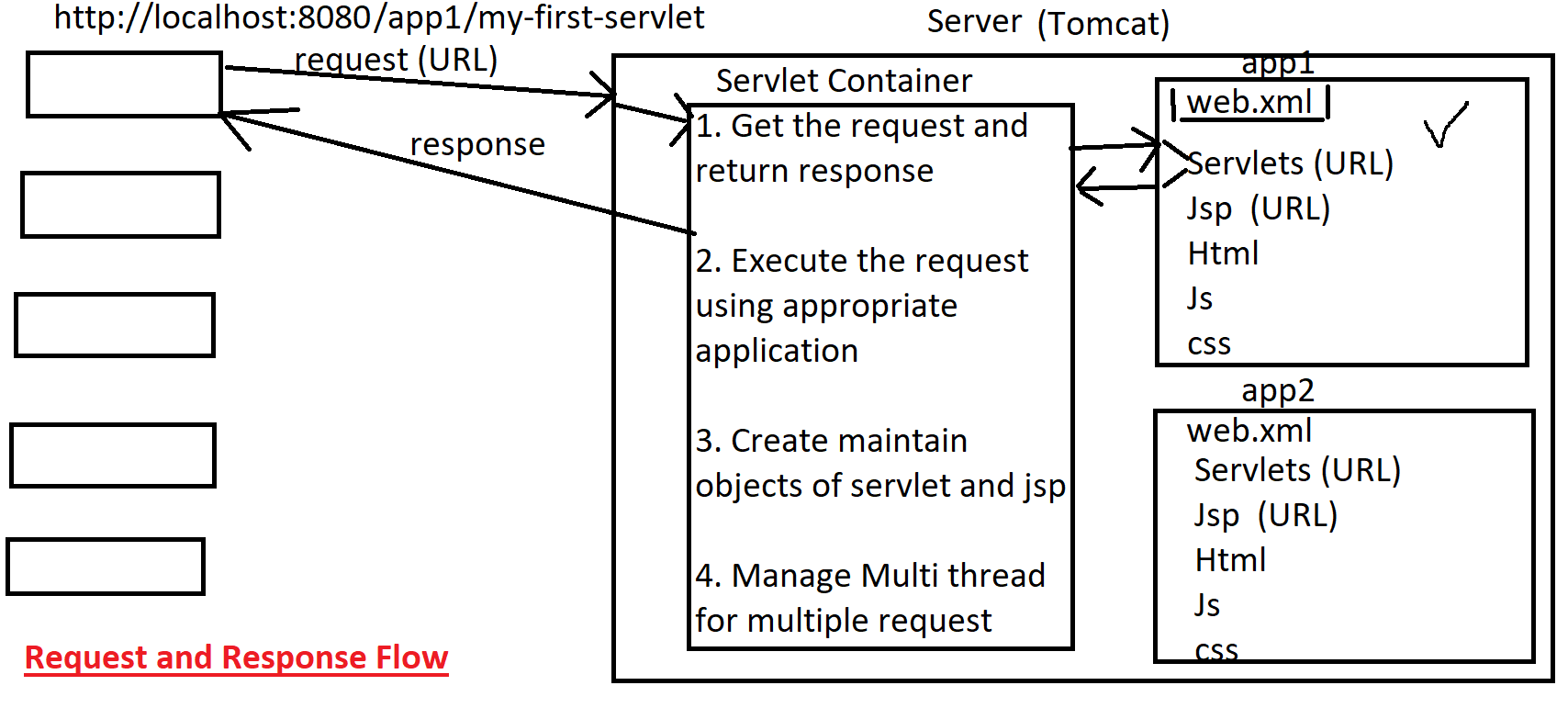
<servlet-name>First</servlet-name>

<url-pattern>/my-first-servlet</url-pattern>

</servlet-mapping>

* + 1. Using Annotation on the servlet class.
       1. You can use annotation on your servlet class to provide URL pattern for your servlet.
       2. This annotation is the alternative for the web.xml tags

@WebServlet("/<url-pattern>")



**Returning a Response to a user from servlet**

1. You can return a response to a user using **HttpServletResponse object**.
2. You can return response of multiple types. Which is called as MIME type

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types>

1. You can set the type of response. Using setContentType(“MIME-Type”). The default content type is “text/html”.
2. To Return a response you have get the object of PrintWirter or OutputStream.
3. **PrintWriter** object is use to return text or html or char type of response.

PrintWriter out = response.getWriter();

1. **OutputStream** is use to return binary type of response like images, audio, video, pdf etc.

OutputStream out = response.getOutputStream();

**Request**

1. Request is always Sends from the user to a server.
2. With a request User can send a data to server.
3. To Handle request in Servlet, can use HttpServletRequest Object.
4. The use information/data can be manage using Parameter or Attribute.
5. Request can be generate from the bowser by entering the URL, by click on an anchor tag (hyper link), after click on the Buttons, refreshing a web page from browser or request can be generated using a java program.

**Parameter**

1. It is a user information/data which has to add from user side to server side.
2. Parameter are always from the URL or from the request Body.
3. Every parameter has a key and value.
4. Every parameter is in string format.
5. The parameters are visible inside url after ‘?’
6. There can be multiple parameters present inside URL all will be separated by ‘&’

Example: [https://www.google.com/search**?q=java&source=hp&ei=ESIPYsdjkIWvvA-RxJKgAg**](https://www.google.com/search?q=java&source=hp&ei=ESIPYsdjkIWvvA-RxJKgAg)

**Re-Direction Techniques**

1. This technique is use to send user request from one page to another.
2. Here the user will be forwarded on the next page without any user action/event.
3. These techniques are also called as redirection techniques.
4. There are 2 techniques.
   1. **RequestDispatcher**
      1. This technique is use to forward the old request from current page to next page.
      2. The data from old request will be available on the new page because the same request forwarded on new page.
      3. RequestDispatcher is an Interface which has 2 method
         1. forward: to forward request from one to another
         2. include: to include one to another
      4. Using this you cannot redirect from one server to another. And hence it is also known as server side redirection.
      5. Syntax:

**RequestDispatcher dis = request.getRequestDispacher(“<URL>”);**

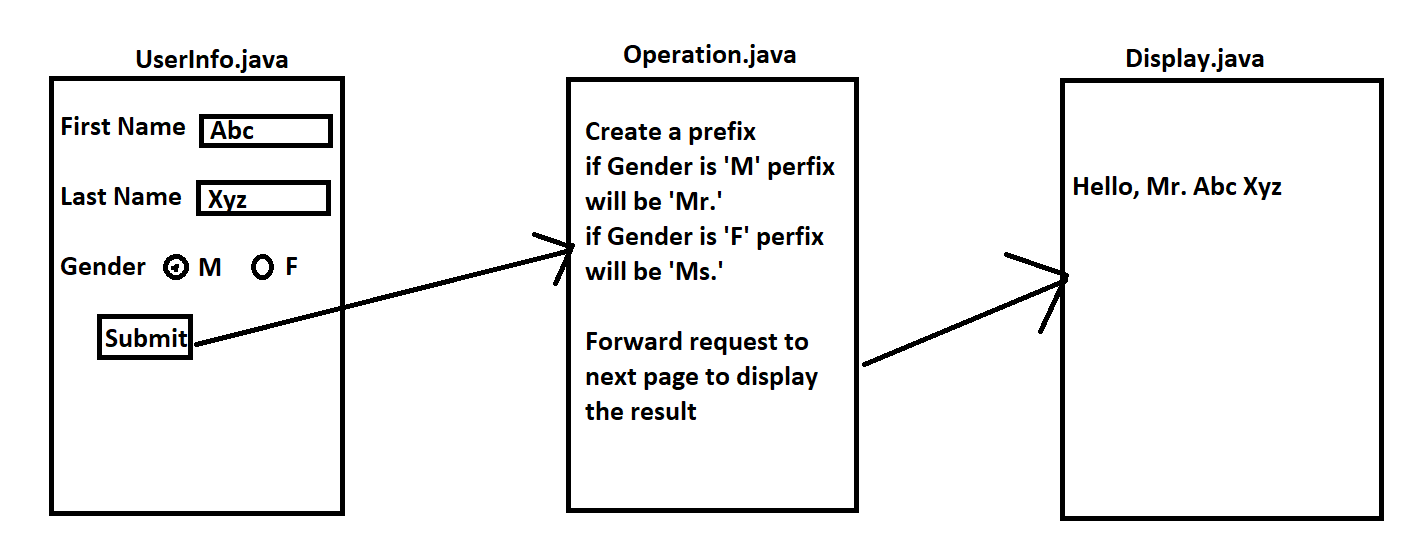
**dis.forward(request, response);**

**dis.include(request, response);**

* 1. **sendRedirect**
     1. This technique is use to forward the request from current page to next page.
     2. This will generate a new request to redirect from one page to another.
     3. The old request details will not be pass to the new page and it gets destroyed.
     4. sendRedirect is a method which can be called using response object.
     5. Using this you can redirect from one server to another. And hence it is also known as client side redirection.
     6. Syntax:

**response.sendRedirect(“<URL>”);**

**Task :**



**JSP (Java Server Pages)**

1. JSP pages are mostly use to design a dynamic web pages.
2. You can use Html, CSS, JS languages on Jsp page directly just like HTML files. Along with front end languages you can also write java code on JSP.
3. Every JSP page internally converts into servlet and every JSP internally use as a servlet.
4. JSP page extension is .jsp.
5. You do not have to configure/provide URL to JSP page.
6. The URL of the jsp page will be a /pageName.jsp

**JSP Life Cycle**

1. JSP life cycle will be manage by Servlet Container.
2. JSP life cycle has 5 stages.
   1. Translation Stage
      1. Every JSP page gets converted into Servlet.
      2. In This stage .jsp file gets converted into .java
      3. Translated Path :

<Workspace\_Path>\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\<Project\_Name>\org\apache\jsp

* 1. Compilation Stage
     1. The java class will be compiled and gets a .class file.
  2. Init stage
     1. In this stage servlet object will be create and initialize.
     2. If you wants to perform specific operation at the time object creation then you can use **\_jspInit() method** of the servlet class.
     3. As a part of this stage init method will be called by servlet container after constructor.
     4. There is only one object of the servlet through the application and hence init method calls only once in a life cycle.
  3. Service stage
     1. In this stage the request will be processed and response will be generated.
     2. In this stage the **\_jspService() method** gets executed.
     3. This method will be called for every request from the user.
     4. This method called multiple times in a life cycle for every use request.
  4. Destroy stage
     1. In this stage the servlet object will be destroyed.
     2. To perform cleanup activities before closing the servlet object, it can be perform inside this stage.
     3. Here **\_jspDestroy() method** will be called before deleting a servlet object.
     4. This step executes only once in a life cycle.

**Scripting Tags/Element**

1. By using scripting tags/element you can write a java code on JSP page.
2. There are 3 types of scripting elements
   1. Scriptlet Tag:
      1. It is use to write a java code on JSP page.
      2. The java code which is written inside this tag will be added inside service method of translated servlet.
      3. That is the code written inside scriptlet tag will be a local code.
      4. You cannot create method in Scriptlet tag.
      5. Syntax:

**<% Java Code %>**

* 1. Expression Tag
     1. Is use to write the java expression which executes and display result on browser.
     2. The code which is written inside this tag, will gets added inside out.print() after translation.
     3. This code will be added inside a service method.
     4. Syntax:

**<%= Java Expression %>**

* 1. Declaration Tag 
     1. Is use to write a java code which will be added at instance level (Outside service method and inside class)
     2. Using this tag you can create method, instance variable, static/class variables.
     3. The code written inside this tag will gets added outside service method and inside class.
     4. Syntax:

**<%! Java Code %>**

**Comment Tag**

1. To Comment HTML tags
   1. Syntax:

<!-- HTML Code -->

1. To Comment Scripting Tags
   1. Syntax

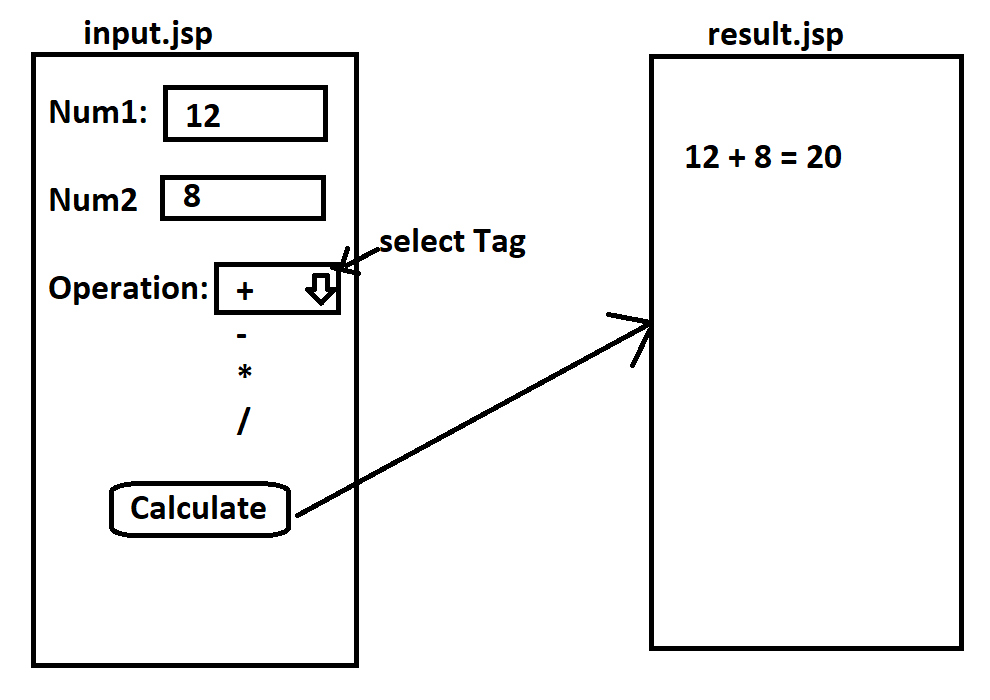
<%-- Scripting tags/elements --%>

**Implicit Object in JSP**

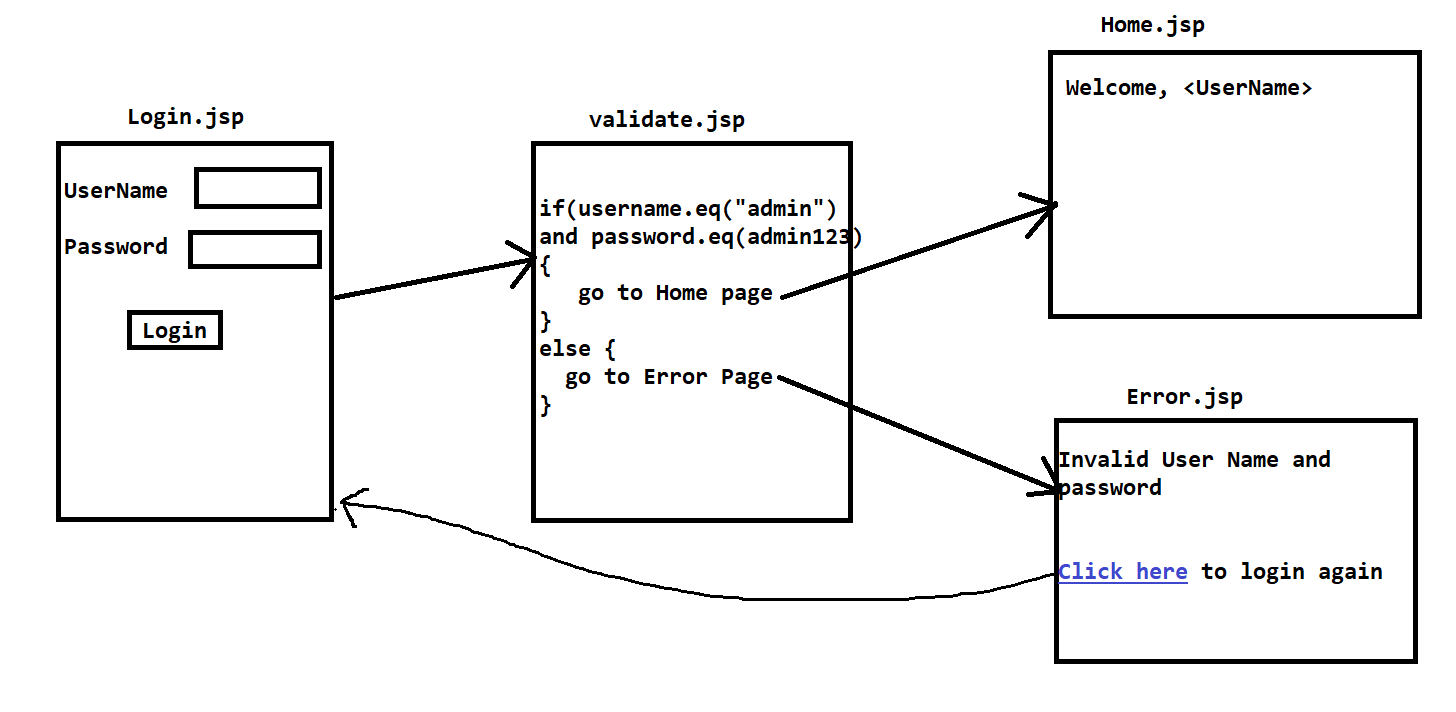
1. The Objects which are available on the JSP page are the implicit object.
2. These objects are directly accessible inside the service method of the JSP.
3. These objects can be use inside scriptlet tag or Expression tag only.
4. Implicit objects are not accessible inside Declaration tag.
5. There is total 9 implicit object.

|  |  |
| --- | --- |
| request | HttpServletRequest |
| response | HttpServletResponse |
| session | HttpSession |
| application | ServletContext |
| config | SevletConfig |
| pageContext | PageContext |
| page | this |
| exception | Throwable |
| out | JspWriter |

Task:



Task:



**Session Management/Tracking**

1. These techniques are use to manage user information into multiple requests.
2. There are 4 ways to manage user session.
   1. **Hidden Form Field**
      1. If you wanted to put the old request data into new request which is generated due to the form and submit button.
      2. To implement this, you just have to add a hidden field (input type hidden) and set the old request information into hidden field value.
   2. **URL Re-writing.**
      1. If you wanted to put the old request data into new request generated dur to a anchor tag or sendRedirect technique.
      2. To implement this, you just have to add a parameter manually inside the URL while providing the URL.
   3. **Cookies**
      1. Cookies are used to store the user information at client side (inside Browser).
      2. These stored cookies will be sent automatically to a server with each and every request.
      3. There can be multiple cookies store for a single application up to 40 cookies.
      4. To create a cookie, you have to create Object of Cookie class and set the key and value which you wanted to store at client side.
   4. **HttpSession**
      1. HttpSession is used to store the user information at server side.
      2. HttpSession internally use cookies to store the tokens which is also known as JSessionId in java.
      3. There is no limit to store the user details inside session.
      4. To create a session, you can use following syntax

**HttpSession session = request.getSession();**

Using above syntax, you will not get a new session always, it will create a new session only if no session created for the user already. If session is already present then it will return an existing session.

* + 1. To set and get the data into session.

**sessionObject.setAttribute(“Key”, Object-Value);**

**sessionObject.getAttribute(“Key”): Object**

* + 1. To Invalid/destroy Session
       1. By Using method

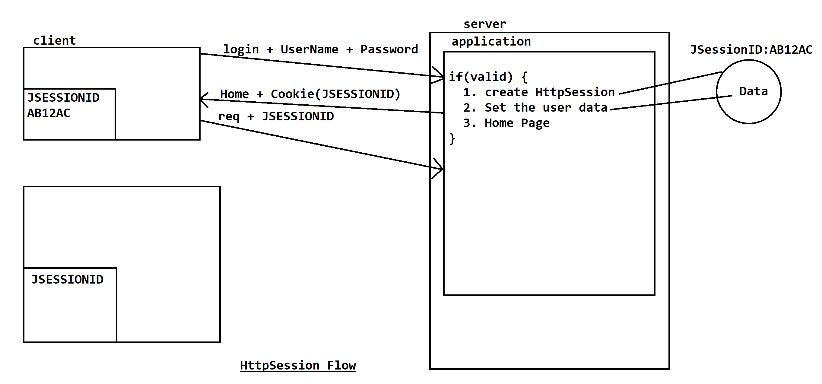
**sessionObject.invalidate();**

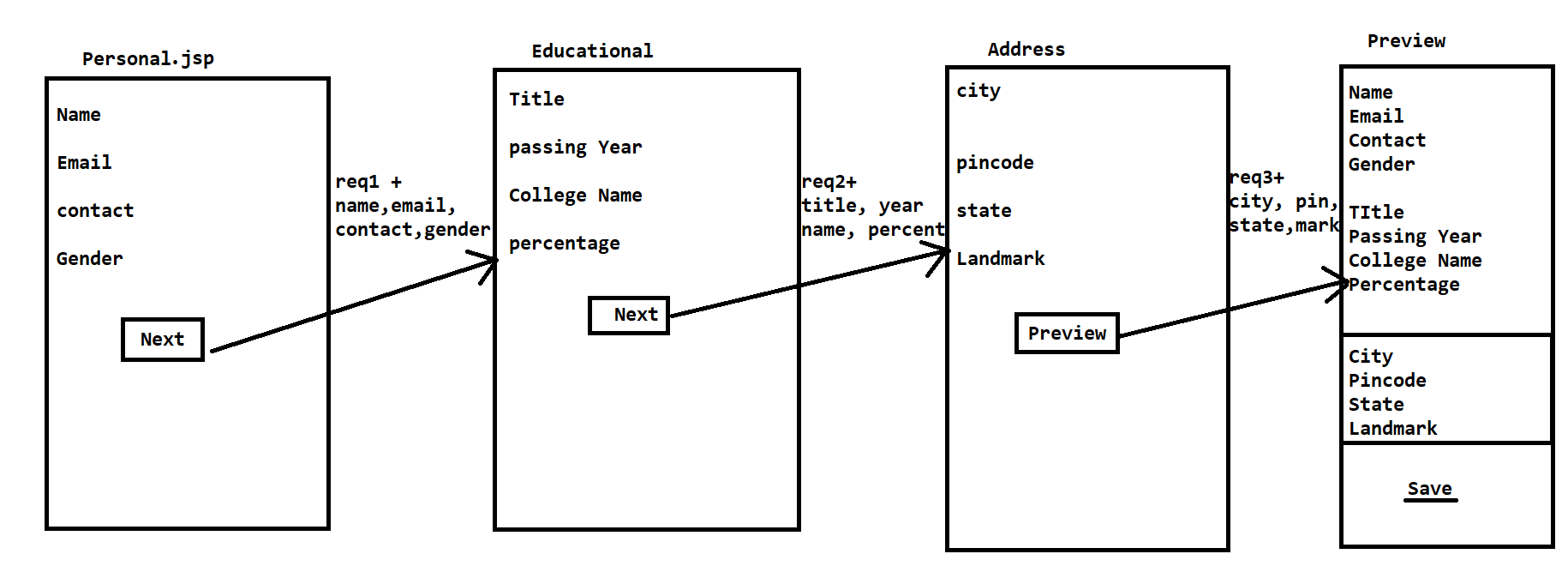
* + - 1. By Xml tags/timeout. Provide time in minutes.

<session-config>

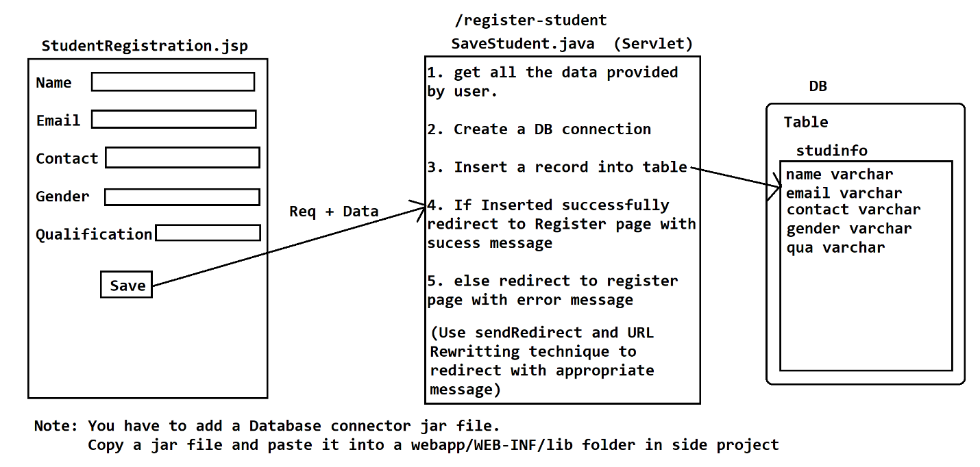
<session-timeout>2</session-timeout>

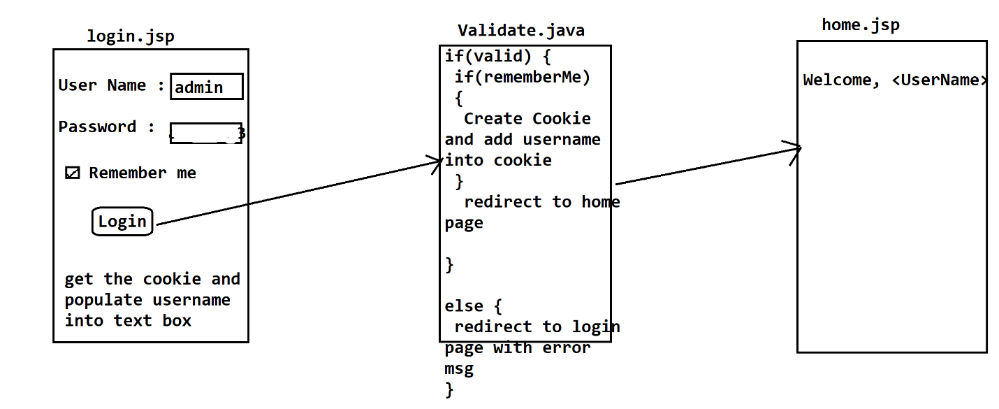
</session-config>

****



**Task**





**Design Custom Error Pages.**

1. Design You Error Page using JSP
2. Declare that jsp page as an error page.



1. Configure the error page inside web.xml

<error-page>

<error-code>404</error-code>

<!-- <exception-type>SqlException</exception-type> -->

<location>/NotFoundError.jsp</location>

</error-page>

**Filter**

1. Is use to perform a pre and post processing of the request.
2. Filter is applied for JSP and servlets. These filters get executed before execution of the actual jsp/servlet and after execution of jsp/servlet.
3. Mostly these filters are used for validation, authentication, authorization and transaction management etc.
4. To Create a filter you can follow the septs.
   1. Create a class and implement Filter interface from the javax.servlet package
   2. Override a methods from the Filter interface.
   3. Write you pre and post processing logic inside doFilter overridden methods.
   4. Set the URL of the filter. The URL of the filter must be same as a jsp/servlet URL before which you want to apply filter.
   5. Use following annotation to set the URL of the filter

@WebFilter(urlPatterns = {"/<URL>"})

@WebFilter(urlPatterns = {"/<URL1>", "/<URL2>", "/<URL3>"})