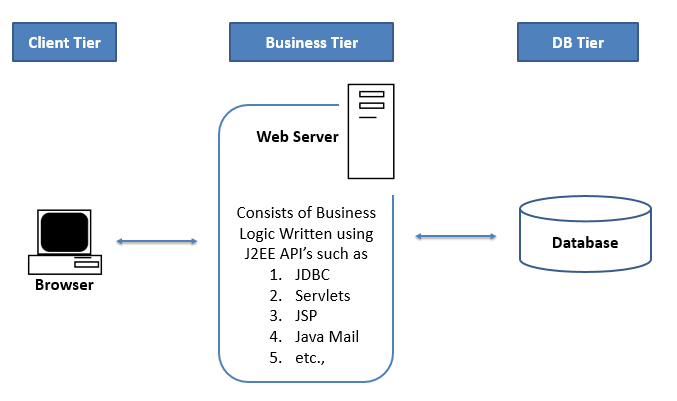
**Recap the initial discussion on J2EE & revisit the J2EE Architecture**

**Should**

**Exercise:**

* Refresh your knowledge about HTML
* Study more about the tag <form> especially about the attributes **“action”** & **“method”** present in side this tag
* Difference between Absolute & Relative Paths/Links in Html



**Web Browser**

* It is a desktop application which helps us to communicate with Webapplications. It’s also called by the name “HTTP Client”
* Browser also understands HTML and present the information accordingly

**Steps to Install Web Browser. Is it Necessary? Instruct students to install Google Chrome**

[**http://bookboon.com/ & Show Images of HDFC Net Banking Transaction Download & explain about Static & Dynamic Web Resources**](http://bookboon.com/)

**Web Application:**

* Web Application is an application which is accessed over the network with the help of web browser
* Resources present inside web application is called as Web Resources & web application a collection of such web resources
* There are two types of web resources

1. Static Web Resources:

Static resources **does exists** at server side before making the request & Content does not change for every request

1. Dynamic Web Resources:

Dynamic Resources **does not exists** at web application side before making the request, they are generated at the time of request & Content may change for every request

Now make a point that J2EE is used to develop dynamic web applications

* If a web application consists of ONLY static resources then it is called as Static Web Application

<http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html>

* If a web application consists of one or more dynamic resources then it is called as Dynamic Web Application

[www.gmail.com](http://www.gmail.com) [www.facebook.com](http://www.facebook.com)

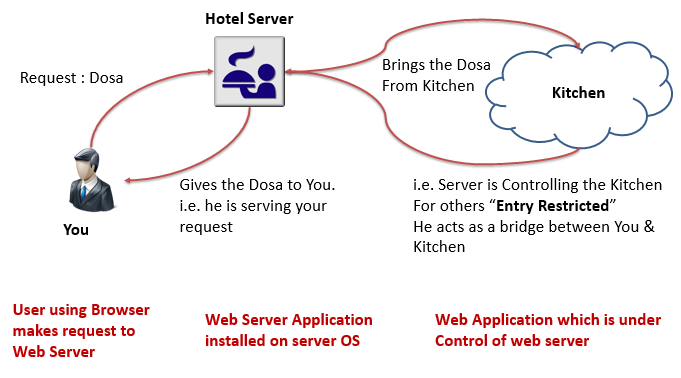
**Explain different ways to make request to web application i.e.**

**1. By typing URL in Browser**

**2. By Clicking on the Hyperlink &**

**3. By Clicking on the Submit button which internally has url information in form**

* Web URL, uniquely identifies web resources in a web application & hence every web resource must need to have its unique address in the form of **Web URL.** In case of static resources URL consists of the static resource file name & in case of dynamic resource URL consists of the configured URL. (Explain Later show the tomcat download hyperlink)

****

**In a Hotel there are two kinds of Food Items available**

1. **Items which are prepared when you make the request like Dosa, Poori, Fruit Juices, etc.,**
2. **Items which are already prepared before you make the request like Ice Creams, Cool Drinks, etc.,**

**Another example is playing a Movie file with the help of Media Player**

**Web Server**

* Like any other application (Adobe Reader, Media Player, etc.,), Webserver is an application which runs on Operating System.
* Webserver as the name implies “Serves Web Requests” for web application. I.e. It helps us both web browser & web application to interact with each other (like how media player help us to play the media file).
* Hence every web application is directly under the control of webserver. Like how each application has its own specific functionality,
* Few Examples of Webservers :
* Apache Tomcat
* JBoss
* Oracle Weblogic
* IBM Websphere

**Steps to Install Tomcat Server:**

1. Download the Apache Tomcat (ZIP version) & extract it into some directory
2. Set the JAVA\_HOME and CATALINA\_HOME environment variables as shown below

JAVA\_HOME = C:\Program Files\Java\jdk1.6.0\_27

CATALINA\_HOME = C:\Praveen\tools\apache-tomcat-6.0.26

1. Go to <Tomcat\_Location>\bin folder & double click on the “Startup.bat” file.
2. Server should start without throwing any exception in the console
3. If server throws an exception, it means that server started up in error mode. In this case most probably there is some issue with setting up environment variables i.e. step 2

**Limitations of Webserver:**

* Webserver cannot generate Dynamic Data (HTML file, PDF file, Excel file, etc.)
* Webserver cannot Interact with Database
* In the above scenario’s, Webserver takes the help of “Other Technologies (PHP, ASP .Net, J2EE, etc.)” to do the job.

**Servlets**

* J2EE helps us to develop Dynamic web applications. Hence Servlets & JSP acts like a Dynamic Resources
* Servlets is an API of J2EE, it accepts web request from web server & generates Dynamic Response.
* In the world of Java, Servlets are the one and only API that helps to capture the web request and generate dynamic response
* Since Servlets are like Dynamic Resources & hence Servlets must have its unique address in the form of Web URL
* **Servlets are like any other Java program**, it accepts the web request via webserver & generate the dynamic Data, gives it back to webserver. Webserver in turn gives this content back to requested user
* Hence Servlets are like dynamic resources present in a network & also every resource in a network must have its unique address in the form of URL & hence Servlets must be accessed via configured URL’s (**one / more**) in web.xml

**Exercise 1: Creating Static Web Application**

**Create a Static Web Application, Give little bit of background of URL & Show them to the students**

**NOTE: - Directly keeping files under webapps will not be accessible using browser. Hence you must keep it in side sub folders of webapps**

**Exercise 2: Creating Dynamic Web Application**

**Necessary steps to create First Servlet:**

1. Create the dynamic web project by selecting web module 2.5 to develop **web application**
2. Created the Plain HTML Page under **WebContent** folder
3. Create the first **servlet** under “src folder” by extending the **HttpServlet**
   1. Copy the **servlet.jar** to **WebContent/WEB-INF/lib folder** to fix the compilation error
   2. Override the **doGet() method**
4. Configured the developed servlet in **web.xml** file
5. Build the Web Project & Generated the **WAR file (Build Process)**
6. Copied the WAR file to **webapps** directory in the webserver (**Deployment Process**)

**> “Web Path / Deploy Path”** is the path or directory location in web server in which web applications are present.

> Web Path varies from server to server & In case of Tomcat the deploy path is **<Tomcat\_Location>\webapps**

1. Started the **Web Server**
2. Accessed the web application using **Web browser** by typing some **URL**
3. Accessed the HTML file (**Static Page**)
4. Accessed my Servlet (**Dynamic page**)

**Index.html:**

**Servlet Code:**

**Web.xml Changes:**

**Web URL (Uniform Resource Locators)**

Every web resource (static / dynamic) should have its unique address in the form of web URL

URL Syntax: Protocol://Domain:Port/Path?query\_string #fragment\_id

Protocol**:**

* When one application wants to communicate with other (in our case browser & server), there needs to be a common language which both application understands & that language should have set of rules and instructions. In software this common language is known as “Protocol” **where protocol is a set of rules**
* Protocol is a communication language used between Browser & Server to communicate with each other where **Protocol is a set of rules**
* The most commonly used protocols are HTTP (Hypertext Transfer Protocol) & HTTPS (Hypertext Transfer Protocol Secure).
* HTTPS encrypts any data which is shared between browser & server so it can't be understood by any hackers who try to intercept the request & response.
* As the name implies **most of the time** HTTP Response contain HTML
* In URL it’s an optional information & default protocol is HTTP

Domain:

* It is a computer name / IP address on which web server is installed and under that web application is deployed
* In URL it is a **mandatory** Information.

Port:

* Porrt number in web url uniquely identifies web server application
* Default port number for HTTP is 80 & default port number for HTTPS is 443
* In URL this is an **optional** information. When it's not used default port number is used depending on the protocol in URL

Path:

* We know that web application is a collection of web resources (Static / Dynamic)
* Path is the full path of the web resource at web application side.
* It consists of web application name and file name in case of static resource or configured URL in case of dynamic resources
* In URL It’s an **optional** Information
* The path typically refers to a combination of file & directory on the web server

**Ex:**  http://localhost:8080/studentsApp/index.html

Query String:

* Query String is a name & value string pair which passes information ONLY to Dynamic Resources such as Servlets & JSPs.
* In URL, It’s an **optional** information & if present, it starts with question mark followed by one or more name-value pair which are separated by an ampersand (&)

Example: [http://localhost:8080/studentsApp/currentDateTime?fname=Praveen&lname=D](http://localhost:8080/studentsApp/currentDateTime)

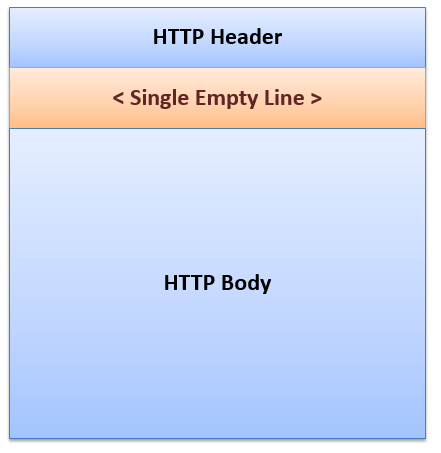
Fragment Id (sometimes called as named anchor)

* A fragment is an internal page reference & it refers to a particular section within a web page.
* In URL, It’s an optional information & if present, it begins with a hash (#) character followed by an identifier

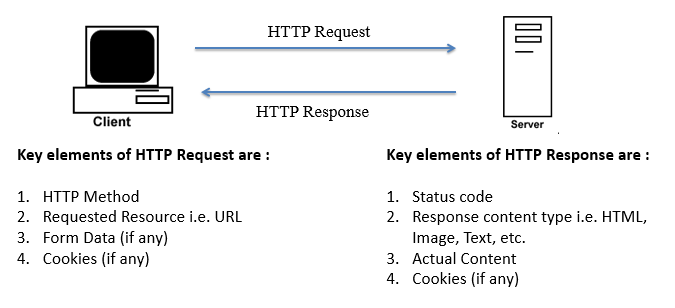
Example: http://tomcat.apache.org/tomcat-6.0-doc/manager-howto.html#Session\_Statistics

**HTTP Structure:**

HTTP consists of Header part as well as Body part which are separated by an empty line



**Key Elements of HTTP Request & HTTP Response:**



**Status Code:**

* Status code represents the status of the HTTP Request. For example,

1. Status code 200 means server successfully handled the request.
2. Status code 404 means that requested resource (static / dynamic) is not found at server side
3. Status code 500 means that server enered an unexpected condition which prevented it from fulfilling the request.

* It’s a **Mandatory** information & it will be **present in Header** of HTTP Response

**Content Type:**

* Content Type OR Multipurpose Internet Mail Extensions (MIME) Type, tells the browser that what type of content it’s going to receive so that it can prepare itself to handle the response data.
* For example, open an Adobe Reader to handle PDF content OR Open Media Player to handle media content etc.
* It’s a **Mandatory** information & it will be **present in Header** of HTTP Response
* The default content type is “text/html”
* Few Examples :

text/html

text/pdf

video/quicktime

image/jpeg

application/x-zip

**Actual Content:**

* It’s a **Mandatory** information & it will be **present in Body** of HTTP Response
* In case of static resource, the content of the resource becomes the “Actual Content”
* In case of dynamic resource the content present in the servlets / JSP becomes the “Actual Content”
* In case of error scenarios webserver generates the error information it becomes the “Actual Content”

**URL:**

**Explain different ways to make request to web application (i.e. 1. By typing URL in Browser, 2. By Clicking on the Hyperlink & 3. By Clicking on the Submit button which internally has url information in form)**

* Every web resource (static / dynamic) should have its unique address in the form of web URL
* It’s a **Mandatory** information & it will be **present in Header** of HTTP Request

**Form Data:**

* Data collected using HTML form is called as Form Data & in HTTP Request it’s an **Optional** Information
* If present then the form data will be present either in HTTP Request Body or in URL depending on the HTTP Method present in the HTTP Request

**HTTP Method:**

* It’s a **mandatory** information present in the header of the HTTP Request
* HTTP method indicates the desired action to be performed on the Dynamic Resource
* HTTP has 8 different methods & Servlet has implementation for these methods (excluding one method).
* All these default implementations are present in the Servlet API class by name “javax.servlet.http.HttpServlet”

(HTTP/1.0 defined 3 methods i.e. GET, POST and HEAD and the HTTP/1.1 added 5 new methods)

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **Related Servlet Method** | **Description** |
| **H**ead | **protected** **void** doHead(  HttpServletRequest req,  HttpServletResponse res)  **throws** ServletException,IOException | This method allows client to see only the headers |
| **T**race | **protected** **void** doTrace(  HttpServletRequest req,  HttpServletResponse res)  **throws** ServletException,IOException | This method Echoes back the received request, so that a client can see changes (if any) happened to the request. |
| **P**ut | **protected** **void** doPut(  HttpServletRequest req,  HttpServletResponse res)  **throws** ServletException,IOException | This method allows client to place a file (document or Web page) on the server and is similar to sending a file by FTP |
| **D**elete | **protected** **void** doDelete(  HttpServletRequest req,  HttpServletResponse res)  **throws** ServletException,IOException | This method allows a client to remove a file (document or Web page) from the server |
| **O**ptions | **protected** **void** doOptions(  HttpServletRequest req,  HttpServletResponse res)  **throws** ServletException,IOException | This method determines which HTTP methods that server supports and returns an appropriate header |
| **Post** | **protected** **void** doPost(  HttpServletRequest req,  HttpServletResponse res)  **throws** ServletException,IOException | * This method allows user to Post the data (unlimited size) to the server using HTML form * **POST requests have a body** & hence data sent using the POST method is included in the **body of the HTTP Request** |
| **Get** | **protected** **void** doGet(  HttpServletRequest req,  HttpServletResponse res)  **throws** ServletException,IOException | * This method allows us to get the data from server * **It’s default method** * **GET requests do not have a body** or empty body & hence the data sent using the GET method is included in the **Header part of HTTP Request in the form of Query String** |
| **C**onnect | **NO Implementation** | **NO Implementation** |

**Using below Login Page explain the points mentioned in POST & GET methods**

**What determines whether browser sends GET or POST request?**

**RECAP: Explain different ways to make request to web application (i.e. 1. By typing URL in Browser, 2. By Clicking on the Hyperlink & 3. By Clicking on the Submit button which internally has url information in form)**

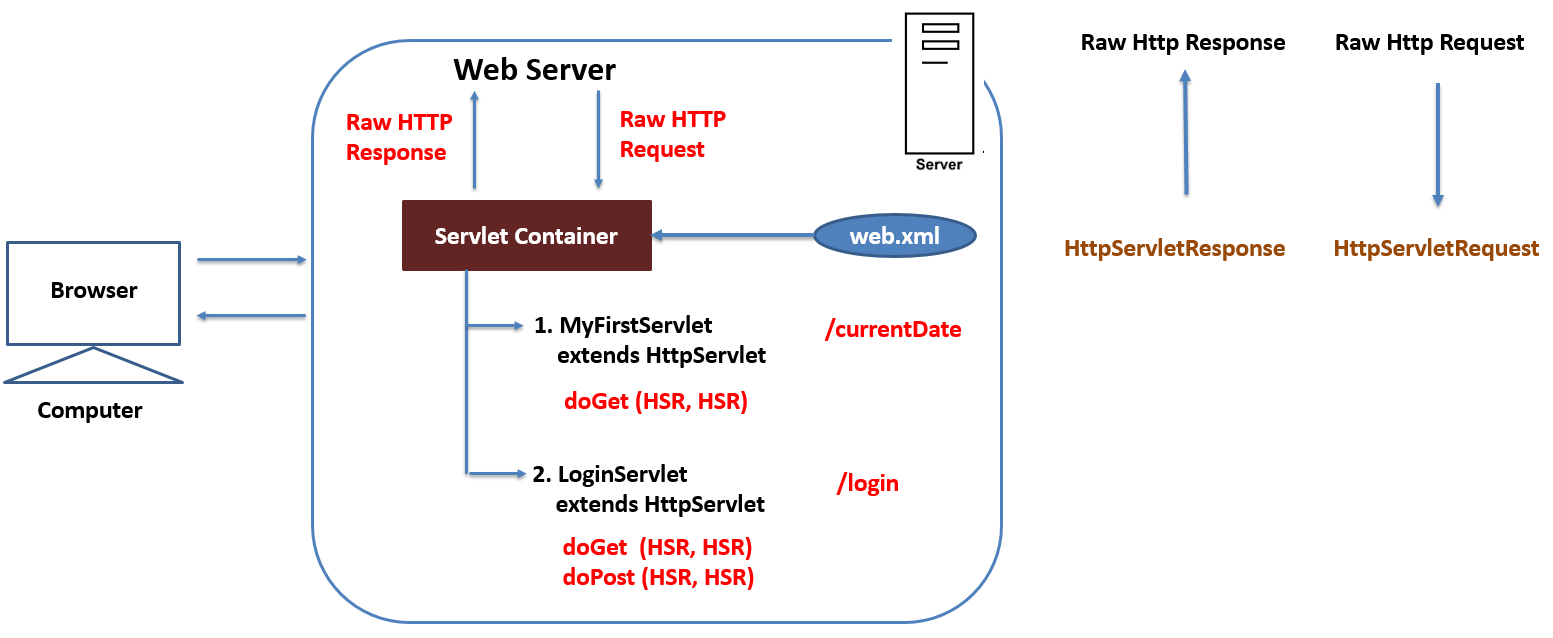
1. Typing the URL in browser makes request to contain GET method
2. By clicking hyperlink on a browser makes request to contain GET method
3. NO method declaration in HTML form makes request to contain GET method
4. **method="get"** in the HTML form makes request to contain GET method
5. **method="post"** in the HTML form makes request to contain POST method

**NOTE:** Depending on the HTTP method in the request, corresponding doXXXX() method get executed at servlet side

**Difference between doGet() & doPost()**

|  |  |  |
| --- | --- | --- |
| **No.** | **doGet() or GET** | **doPost() or POST** |
| 1 | This method allows us to get the data from server | This method allows user to Post data of **unlimited size** to the server |
| 2 | It’s a default method | It’s not default. We have to explicitly declare method=”post” in the HTML form |
| 3 | GET requests do not have a body | POST requests have a body |
| 4 | Data sent using GET will be present in the header part of HTTP Request in the form of query string | Data sent using post will be present in the body of the HTTP Request |
| 5 | Insecure; because the form data get exposed to the outside world | Secure |
| 6 | The amount of data we can send using GET is restricted (URL can contain Only 1024 characters) | There is no restriction on the amount of data send using the POST |
| 7 | We cannot send the files using GET (Resume Upload) | we can send entire files using POST |
| 8 | GET requests are by nature they are “idempotent”. i.e. we can perform the same operation again & again without any side effects. But you can write a BAD, non-idempotent doGet() method | Non-idempotent |
| 9 | We can bookmark the GET requests | We cannot bookmark POST requests |

**Servlet Container**



* Servlet Container is a sub-component of web server that helps both web server & servlet to communicate with each other
* As the name implies, all Servlets of dynamic web application are directly under the control of the Servlet Container

**How Servlet Container Works**

* Whenever the request comes, web server hand over the complete request to servlet container.
* Container by looking at the URL present in the request & referring web.xml it comes to know about the servlet which handles that request.
* **Container then creates an instance of that servlet**
* Once Instance creation is successful then it converts the raw HTTP Request to a Java Object of type “HttpServletRequest” & creates HttpServletResponse object
* Depending on the HTTP Method present in the request container invokes corresponding doXXX() method by passing these request & response objects
* Once the doXXX() method execution is over, container converts the response object to Raw HTTP Response & gives it back to web server
* Once the response has been give back, Servlet Container garbage collects the request & response objects.
* **i.e. for every request container creates new request & response objects**

**javax.servlet.http.HttpServletRequest**

* HttpServletRequest is an object representation of Raw HTTP Request.
* In short it is called as request object & it provides the information about the request
* HttpServletRequest is an interface which extends another interface “java.servlet.ServletRequest”
* Methods getParameter() & getParameterValues() are present in ServletRequest interface helps us to get information from request object. Both methods returns null if the parameter does not exist

**Syntax :**

[String ServletRequest.getParameter(String paramNM)](eclipse-javadoc:%E2%98%82=studentsApp/C:%5C/tools%5C/PersonalWorkspace%5C/BJM1617_Worksapce%5C/studentsApp%5C/WebContent%5C/WEB-INF%5C/lib%5C/servlet.jar%3Cjavax.servlet(ServletRequest.class%E2%98%83ServletRequest~getParameter~Ljava.lang.String;%E2%98%82String)

[String[] ServletRequest.getParameterValues(String paramNM)](eclipse-javadoc:%E2%98%82=studentsApp/C:%5C/tools%5C/PersonalWorkspace%5C/BJM1617_Worksapce%5C/studentsApp%5C/WebContent%5C/WEB-INF%5C/lib%5C/servlet.jar%3Cjavax.servlet(ServletRequest.class%E2%98%83ServletRequest~getParameterValues~Ljava.lang.String;%E2%98%82String)

**javax.servlet.http.HttpServletResponse**

* HttpServletResponse object is an object representation of Raw HTTP Response.
* We should make use of response object to send the response information
* HttpServletResponse is an interface which extends another interface “ServletResponse”
* Response object has a method by name “setContentType()” which helps us to provide “Content Type” information for the response

[void ServletResponse.setContentType(String contentType)](eclipse-javadoc:%E2%98%82=studentsApp/C:%5C/tools%5C/PersonalWorkspace%5C/BJM1617_Worksapce%5C/studentsApp%5C/WebContent%5C/WEB-INF%5C/lib%5C/servlet.jar%3Cjavax.servlet(ServletRequest.class%E2%98%83ServletRequest~getParameter~Ljava.lang.String;%E2%98%82String)

* To send the actual content information in the response we get “java.io.PrintWriter” object from response object with the help of getWriter()
* PrintWriter object has a method by name “println() / print()” which helps us to add the actual content to response object

Using the following HTML Form options we can collect the different information’s from user:

* Text Box, Password, Text area, Radio button, Drop-down menu with Single Selection & Hidden field
* Check box & Drop-down menu with Multiple Selection

<input type=*"checkbox"* name=*"vehicle"* value=*"Bicycle"*> I have a Bicycle

<input type=*"checkbox"* name=*"vehicle"* value=*"Car"*> I have a car

<select multiple name=*"tech"* size=*"6"*>

<option value=*"Java"*>Core Java</option>

<option value=*"Servlets"*>Servlets</option>

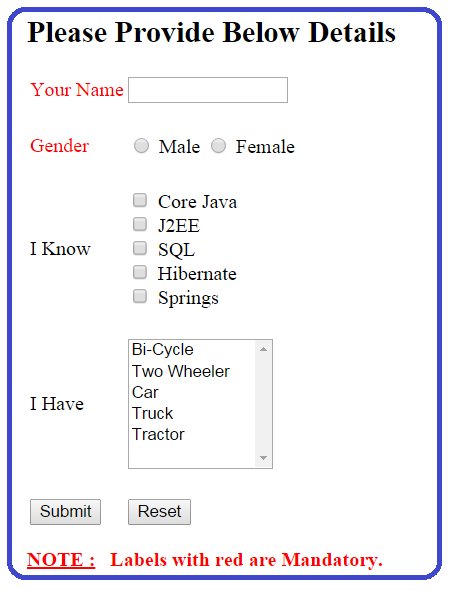
</select>

**Advantages of Servlet Container:-**

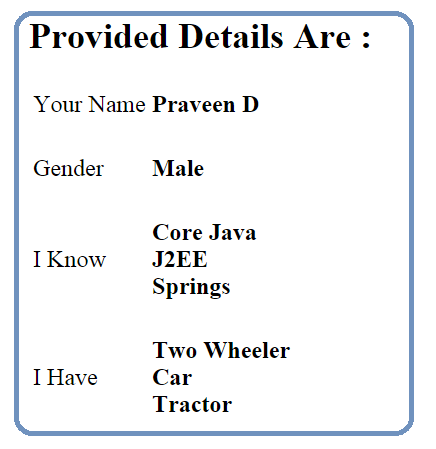
1. **Communication Support:** Container helps both web server & servlets to communicate with each other
2. **Multi-Threading Support:** the container automatically creates a new thread for every incoming request
3. **Declarative Support:** With web.xml, which is used by servlet container, we can change the behaviour of web application without changing anything in the source code
4. **Life Cycle Management:** Container controls the Life Cycle of servlets
5. **JSP Support:** the container takes care of translating JSP into Servlet

**Assignment 1:**

* Create a Form as shown below



* Create a servlet which collects information from above form & prints in the Browser. For example,

****

**Deployment Descriptor (web.xml)**

**In an examination there will be an “Instruction Sheet” which tells about how to answer the questions, marks for the set of questions & Duration etc. Likewise even web.xml gives the specific instruction to servlet container**

* Deployment Descriptor (DD) is an XML file which is used by Servlet Container to handle the request.
* In other words, It’s kind of interaction sheet to servlet container
* The deployment descriptor must have the name “web.xml” and must be present in the WEB-INF folder. Every dynamic web application must have ONLY ONE web.xml
* Servlets must be accessed via URL & web.xml defines mappings between URL paths and the servlets that handle requests with those paths. Container uses this information to identify the servlet to handle a given request

<servlet>

<servlet-name>someName</servlet-name>

<servlet-class>pkg-name.class-name

</servlet-class>

</servlet>

<!-- 1. Exact Match -->

<servlet-mapping>

<servlet-name>someName</servlet-name>

<url-pattern>/firstUrl</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>someName</servlet-name>

<url-pattern>/anotherUrl</url-pattern>

</servlet-mapping>

<!-- 2. Directory Match -->

<servlet-mapping>

<servlet-name>someName</servlet-name>

<url-pattern>/abc/\*</url-pattern>

</servlet-mapping>

<!-- 3. Pattern / Extension Match -->

<servlet-mapping>

<servlet-name>someName</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

**Order of Preference:**

1. Exact Match
2. Directory Match
3. Pattern / Extension Match

* Using DD, we can customize server response in case of Error Scenarios. The server can display an alternate page in such cases

<error-page>

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

<location>/students</location>

</error-page>

<error-page>

<exception-type>java.lang.Throwable</exception-type>

<location>/error</location>

</error-page>

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)

**throws** ServletException, IOException

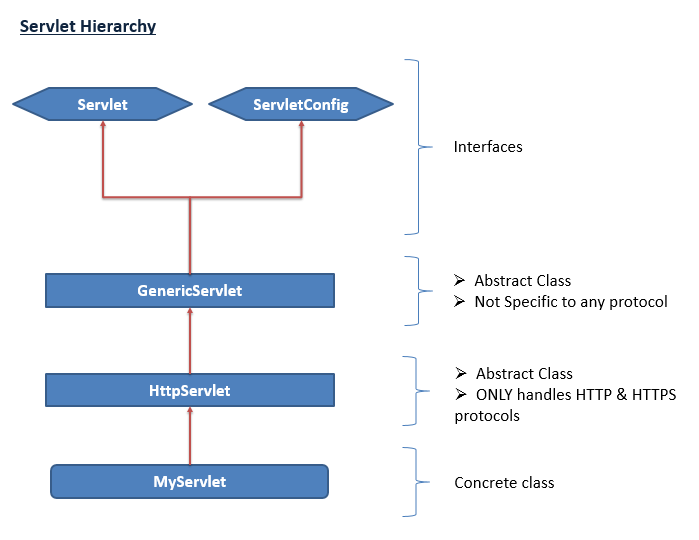
{

**throw** **new** ServletException("Hello !!!");

}

* In web.xml <welcome-file-list> tag is used to configure default page for the web application. If no resource is specified in the URL, then container searches the resources present in this tag in the same order. If container is able to find the file resource then it gives that as the response if not then it looks out for the other resource in the order

**Servlet Hierarchy:**

****

**javax.servlet.GenericServlet:**

* It's an abstract Class, part of Servlet API & A sub-class of GenericServlet is called as Servlet & it can handle any type of protocol **including HTTP & HTTPS.**
* In other words, It becomes **protocol-independent servlet**
* Since the service() method present inside the GenericServlet class is declared as Abstract, **GenericServlet is an Abstract Class**.

Syntax: **public abstract void service(**ServletRequest req, ServletResponse res**)**

**throws** ServletException, IOException**;**

Generic Servlet Example:-

**public** **class** MyGenericServlet **extends** GenericServlet

{

**public** **void** service(ServletRequest req, ServletResponse res)

**throws** ServletException, IOException

{

res.setContentType("text/html");

PrintWriter out = res.getWriter();

out.println("Prtocol Independent Servelt…");

out.close();

}

}

**javax.servlet.http.HttpServlet:**

* A sub-class of HttpServlet is called as Servlet & it can handle **ONLY HTTP & HTTPS protocols.** In other words, It becomes **protocol-dependent servlet**
* It’s an abstract class but none of the methods in this class is declared as abstract. This class must be sub classed to handle HTTP requests
* A subclass of HttpServlet **can override any of the below service() method**

1. service(ServletRequest req, ServletResponse resp)
2. service(HttpServletRequest req, HttpServletResponse resp)
3. doXXX(HttpServletRequest req, HttpServletResponse resp)

* **We should not override the first two version of service methods.** Our job is to override one / more doXXX() method
* **If we won’t override doXXX() methods, then default implementation from HttpServlet is invoked which in turn return 405 error response**

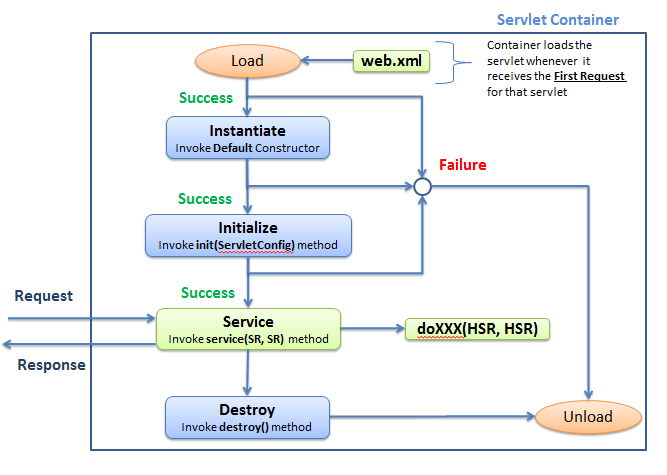
Why HttpServlet is an Abstract Class?

HttpServlet does not have any abstract methods. But being Abstract, we are forced to subclass it. If those methods were all abstract, you would be forced to override them all, even though your business requirements don't need it at all. Thus, to force us to implement our own servlet class, the HttpServlet class is marked as abstract.

**Difference between Generic Servlet & HttpServlet**

|  |  |  |
| --- | --- | --- |
| **No.** | **GenericServlet** | **HttpServlet** |
| 1 | Protocol Independent | Protocol Dependent; supports only HTTP & HTTPS protocols |
| 2 | Abstract Class; because service() method is declared as abstract | Abstract class; but none of the methods are declared as abstract |
| 3 | If we extend the GenericServlet then we must provide the implementation for service() method | There is NO restriction on overriding any version of the service method. But generally we override one or more doXXX() methods. |
| 4 | GenericServlet does not extend any other Servlet API related class | HttpServlet extends GenericServlet which is part of Servlet API |
| 5 | GenericServlet implements Servlet API related interfaces such as Servlet, ServletConfig | HttpServlet does not implements any Servlet API related interfaces |
| 5 | It belongs to “javax.servlet” package | Belongs to “javax.servlet.http” package |

**Servlet Lifecycle:**



The lifecycle of a servlet is controlled by the Servlet Container and it has following phases.

1. Instantiation Phase
2. Initialization Phase
3. Service Phase
4. Destruction Phase
5. **Instantiation Phase:-**

* Whenever the request comes to a container by using the URL information from request & referring the web.xml, it tries to find the servlet name.
* If there is NO servlet found, then it return the 404 error response
* If the servlet is found then it looks out for servlet instance in its cache. If not present then container creates an instance of the servlet by invoking the **public default constructor ONLY**

1. **Initialization Phase:-**

Version 1: **public** **void** init(ServletConfig config)

**throws** ServletException

{

**super**.init(config);

//Initialization Code goes here

}

Version 2: **public** **void** init()

**throws** ServletException

{

//Initialization Code goes here

}

* After successfully creating an instance container automatically invokes init(ServletConfig config) method
* init() method gives us a chance to initialize the servlet before handling any client requests. Like, read data from property file etc.
* This method is called **ONLY Once** in the lifecycle of a servlet
* **We may / may not override this method.** If we don’t override then default implementation from the GenericServlet is invoked
* The first line of version 1 init() method **Should** be “**super**.init(config)”. This allows GenericServlet to save a reference to ServletConfig & make it available to methods outside of the init()
* If an error occurs during the initialization phase, servlet throws a ServletException. Then, servlet container unloads the servlet & put it for garbage collection. **In this case destroy() method is not called**
* Once the instantiation & initialization is successful, container caches the servlet instance
* **During initialization servlet has access to two key objects**
  + - * ServletConfig
      * servletContext
* We can use constructor for initialization purpose, but this approach is not so common. Also init() method has access to ServletConfig & ServletContext objects whereas constructor don’t. Hence we generally make use of init() method for initialization purpose.

1. **Service Phase:-**

**public** **void** service(ServletRequest req, ServletResponse res)

**throws** ServletException, IOException

{

//Service Code goes here

}

* After instantiation & initialization, container creates Request & Response objects
* Creates a thread for the incoming request & instructs the thread to run the **service(ServletRequest req, ServletResponse resp)** method by passing request and response objects
* Depending on the HTTP Method in the request, the service() method invokes corresponding doXXX() method
* Service method is called for every request i. e. It is called **one / more times** in servlet lifecycle
* If servlet is a subclass of GenericServlet then we should override this method & if it is a subclass of HttpServlet then we **SHOULD NOT** override & our job is to override doXXX() methods
* Once the execution of service() method is over, the thread dies, container converts the response object into an HTTP Response & sends it back to the webserver.

1. **Destruction Phase:-**

**public** **void** destroy()

{

//Clean-up Code goes here

}

* Whenever container wants to remove the cached servlet instance then it invoke destroy() method before removing. destroy() method is similar to finalize() method in garbage collection
* The decision of when to destroy a servlet instance rests on the shoulders of servlet engine. Developers should not be concerned with these details, but instead focus on what should be done when the time comes.
* This method is called **ONLY Once** in the lifecycle of a servlet
* **We may / may not override this method**. If you don’t override destroy(), then default implementation from GenericServlet is called

**About <load-on-startup> Tag:**

* Generally, servlets are instantiated & initialized for first request. Therefore, time taken to generate the response for first request is more compared to subsequent request
* This can be avoided by preloading the servlet at server start up time with the help of <load-on-startup> tag specifying a value >= 0. For example,

**<servlet>**

**<servlet-name>MyServlet</servlet-name>**

**<servlet-class>mypackage.MyServlet</servlet-class>**

**<load-on-startup>1</load-on-startup>**

**</servlet>**

Note: Order in which servlets are preloaded depends on the value of the <load-on-startup>

**Note:**

1. Any Java class which extends either javax.servlet.http.HttpServlet or javax.servlet.GenericServlet is called as Servlets. In other words Servlet is an object of type javax.servlet.Servlet Interface
2. Servlets must be concrete class otherwise they fail at runtime
3. **Servlets are protocol independent in nature**, but are most often used with the HTTP & HTTPS protocols
4. If a class extends either HttpServlet or GenericServlet & a subclass of that class is also be called as Servlet
5. Servlets can have local variables, it’s own methods (static / non-static), block of code (static / non-static), inner class etc., **but they cannot have abstract methods**
6. **We can also have main() method in servlet but it’s of no use**
7. Servlets should have public default (i.e. no argument) constructor or combination of any other constructor along with public default constructor
8. There is only one instance exist for any servlet. i.e. **Servlets are Singleton in nature**
9. Every request to a servlet runs in a separate thread.

**SingleThreadModel Servlets**

* Following are the two ways to create Servlet in Single Threaded in nature

1. By implementing SingleThreadModel Interface
2. By Synchronizing Service Method (i.e. doXXX() method)

* SingleThreadModel Marker Interface that ensures that servlets handle only one request at a time. This interface is deprecated in Servlet API version 2.4
* By Synchronizing the service method of the servlet also guaranties that at any given point of time only one thread will be acting on the service method of that servlet. But the difference compared to above approach is that in this case Only One Servlet instance exists.

**Example:**

1. **public** **class** CurrentDateTime **extends** HttpServlet

**implements** SingleThreadModel

{

// Servlet Code Goes Here

}

1. **public** **class** CurrentDateTime **extends** HttpServlet

{

**protected** **synchronized** **void** doXXX(HttpServletRequest req,

HttpServletResponse res)

**throws** ServletException, IOException

{

// Service Method Code Goes Here

}

}

**Assignment 2:**

Create a servlet which s the Number of hits for that page & displays it in the browser (Interview Question)

**Assignment 3:**

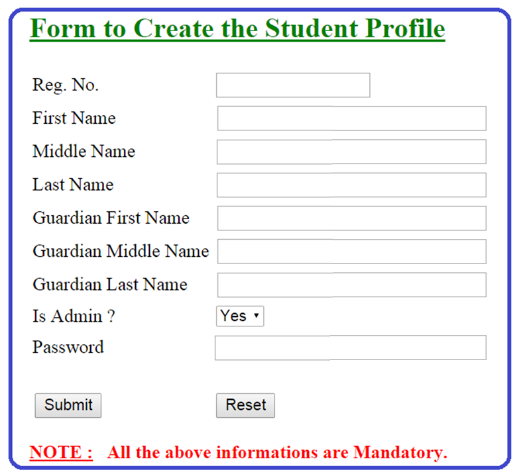
* Create a form which collects register number information from user & sends it to a servlet
* Create a servlet which captures the register number information from this form, fetches corresponding register number information from DB (combination of data from students\_info, guardian\_info & students\_info tables) & prints it into browser

**Assignment 4:**

* Create a servlet which captures the register number information from query string fetches corresponding register number information from DB (combination of data from students\_info, guardian\_info & students\_info tables) & prints it into browser

**Assignment 5:**

* Create a form as shown below



* Create a servlet which captures the above information from this form & inserts the data into DB & gives the proper response

**Assignment 6:**

* Enhance Assignment 3 in such a way that display the information in browser in the form of “HTML Form” by allowing user to edit the information
* Create a servlet which captures the above information from this form & updates the data into DB & gives the proper response

**Assignment 7:**

* Create a form which collects regno, current password, new password information & pass it to a servlet
* Create a servlet which captures the above information from the form & it updates the new password into DB & gives the proper response

**Assignment 8:**

* Create a login page which collects the register number & password from user & pass it to a servlet
* Create a servlet which captures this information from form & authenticates it. If user has provided the valid credentials then print “Valid User Name / Password” success message or If user has provided the in-valid credentials then print “In-Valid User Name / Password” error message in the browser

**ServletContext & ServletConfig**

|  |  |  |
| --- | --- | --- |
| No. | **ServletContext** | **ServletConfig** |
| 1 | ServletContext is an Interface and an Object of Sused by a servlet container to pass information to **ALL the servlets** which are part of an application  Mention about Lifespan | ServletConfig is an Interface used by a servlet container to pass information to a particular servlet during its initialization |
| 2 | There will be only one ServletContext object exists for the entire application. Hence singleton in nature | There will be only one ServletConfig object exists per Servlet |
| 2 | ServletContext object reference is obtained by calling “getServletContext()” method which we inherit from GenericServlet  ServletContext context = getServletContext(); | Servletconfig object reference is obtained by calling “getServletConfig()” method which we inherit from GenericServlet  ServletConfig config = getServletConfig(); |
| 3 | In web.xml, servlet context parameters are declared under <context-param> tag (one / more) | In web.xml, servlet config parameters are declared under <init-param> tag (one / more) which is a sub tag of <servlet> tag |
| 4 | ServletContext does not holds the reference of ServletConfig | ServletConfig holds the reference of ServletContext  ServletConfig config = getServletConfig();  ServletContext context =  config.getServletContext(); |

**Note:**

* Both ServletContext & ServletConfig Objects has the method getInitParameter() which helps us to get the init param information from web.xml

String getInitParameter(String paramName)

* Both ServletContext & ServletConfig allows application to have init params specified outside the code & they can be changed without touching the source code. We can get Context or Config init params at runtime but, **we cannot set them**

Example:

<context-param>

<param-name>Movie1</param-name>

<param-value>DDLJ</param-value>

</context-param>

<servlet>

<servlet-name>serv1</servlet-name>

<servlet-class>

com.jspiders.studentsapp.servlets.Servlet1

</servlet-class>

<init-param>

<param-name>actor1</param-name>

<param-value>Amir Khan</param-value>

</init-param>

</servlet>

ServletContext context = getServletContext();

String movie1Val = context.getInitParameter("Movie1");

out.println("Movie 1 : "+movie1Val);

ServletConfig config = getServletConfig();

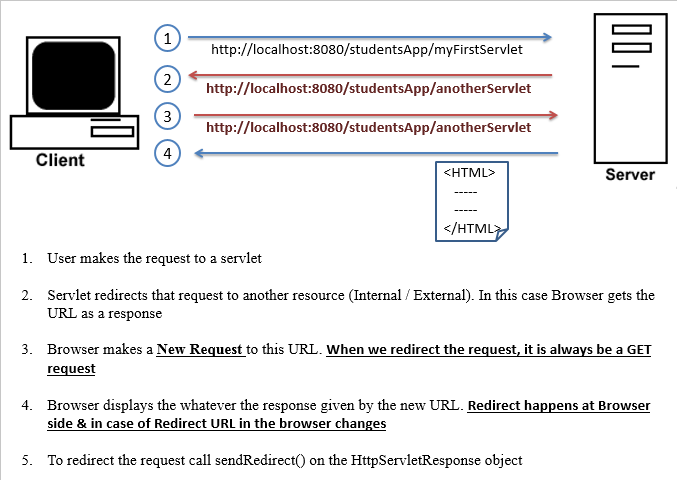
String actor1Val = config.getInitParameter("actor1");

out.println("Actor 1 : "+actor1Val);

**Assignment 8:**

* Enhance the assignment 7 in such way that, after successful authentication check for the admin rights.
* If user is a Admin user then print “Welcome Admin User” otherwise print “Welcome Normal User” in the browser

**Redirect**



res.sendRedirect("http://www.google.com");

http://www.google.com

res.sendRedirect("http://localhost:8080/studentsApp/students?username="

+req.getParameter(“username”));

http://localhost:8080/studentsApp/students?username=praveen

res.sendRedirect("students");

http://localhost:8080/studentsApp/students

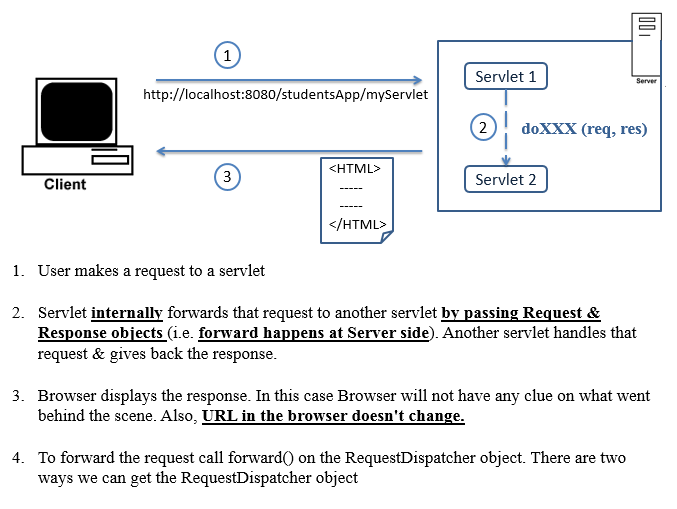
res.sendRedirect("/students");

http://localhost:8080/students

**Assignment 8:**

* Build a simple html form which would take a keyword (Ex: paper, pen, etc.,) and pass it to GRAINGER website and display the results on the supplier website for that particular keyword

**Forward:**

****

1. ServletRequest.getRequestDispatcher(String path)

RequestDispatcher dispatcher = req.getRequestDispatcher("students");

dispatcher.forward(req, res);

1. ServletContext.getRequestDispatcher(String path)

ServletContext context = getServletContext();

RequestDispatcher dispatcher = context.getRequestDispatcher("/students");

dispatcher.forward(req, res);

**Note:** In this case URL **SHOULD** start with ‘/’. Otherwise we get “IllegalArgumentException” at Runtime

**Difference between Redirect & Forward**

|  |  |  |
| --- | --- | --- |
| **No.** | **Redirect** | **Forward** |
| 1 | It happens at the Browser side | It happens at the server side |
| 2 | URL in the browser changes | does not change |
| 3 | We can Redirect the request to External / Internal URL | We can forward the request **ONLY** to Internal URL |
| 4 | Redirect contains more than one request & response cycle | Forward contains only one request & response cycle |
| 5 | Slower in nature | Faster in operation |
| 5 | Redirected request always contain GET method & hence it invokes doGet() method at destination | When we forward the request, it will invoke the corresponding doXXX() method at destination. If the corresponding overridden version of doXXX() method does not exists in another servlet then it invokes the inherited version of doXXX() method |
| 6 | sendRedirect() method is present in HttpServletResponse | forward() method is present in RequestDispatcher which can be obtained by using HttpServletRequest or ServletContext Object |

**Assignment 9:**

* Enhance the assignment 8 in such a way that if User a Normal User then display his own information (combination of data present in students\_info & guardian\_info table) & If user is an Admin User then display ALL students information in browser Approa

**Include**

* It includes the response from the internal resource (static / dynamic ) within the caller response
* When we Include the content of one servlet into an another, it will include the response of corresponding overridden version of doXXX() method in that servlet
* If corresponding doXXX() methods does not exists in another servlet **then it does not throw any Compile time / Runtime Exception.** In this case it just ignore the include statement.

out.println(" 1111111111");

RequestDispatcher dispatcher = req.getRequestDispatcher("index.html");

dispatcher.include(req, res);

out.println(" 22222222222 ");

**Assignment 10:**

* Enhance assignment 9 in such a way that in case of error scenarios (i.e. Invalid Data or Authentication Failure) along with displaying the error information display the login form as well

**Assignment 11:**

* Create a header page which consists of Home, Create Profile, Change Password & Logout hyperlink
* Create a footer page which consists of “Copyright Information”
* Include Header & Footer page in the response of the assignment 9

**Attributes:**

* Attributes helps us to pass information in the form of name= value pair where **name is a String & value is java.lang.Object**. Which means that, any java object can be an attribute value.
* Attributes are the one and only way to pass information in the form of Java object from one servlet/JSP to another servlet/JSP
* Attributes are of 3 types

1. Context Attributes (Application Scope)
2. Request Attributes (Request Scope)
3. Session Attributes (session Scope)

* These 3 attributes can be get or set by using the following objects respectively.

1. Javax.servlet.ServletContext
2. Javax.servlet.ServletRequest
3. Javax.servlet.Http.HttpSession

* The following methods are present in the above 3 objects, which can be used to get, set or remove attributes.

1. **void setAttribute(String name, Object value)**

- Sets an object to a given attribute name.

- **If an attribute with the same name exists then, this method will replace the existing object with the new object**

1. **Object getAttribute(String name)**

Returns an attribute value as java.lang.Object with the given name or returns null if there is no attribute value by that name exists

1. **void removeAttribute(String name)**

- Removes the attribute with the given name.

- After removal, subsequent calls to getAttribute() with the same name will return null

**Cookies: -**

* Cookies are little piece of information in the form of name-value string pair exchanged between browser & server
* **Cookies are created by server and maintained by browser & hence Cookies are Browser Dependent.**
* There are two types of cookies,

1. Non-Persistent Cookies (it’s default)

Non-Persistent Cookies live as long as browser is kept open. Once the browser is closed the cookie disappears.

1. Persistent Cookies

A persistent cookie will persist in browser memory even after the browser is closed. If a persistent cookie has its Age set to 1 year, then within the year, cookie will be sent back to the server **whenever user makes request to the same server by using same browser.**

**Cookie Related Methods:**

1. public Cookie(String name, String value)
   * It’s the one & only constructor available in Cookie concrete class & it constructs a cookie object with a specified name and value.
   * Name can contain only alphanumeric characters and cannot contain commas, semicolons, or white space or should not begin with a $ character.
   * The cookie's name cannot be changed after creation.
   * The value can be anything & value can be changed after creation.
2. void Cookie.setMaxAge(int expiry)

* Sets the maximum age of the cookie **in seconds**
* A positive value makes Cookie Persistent in nature
* A negative value makes Cookie Non-Persistent in nature
* A zero value causes the cookie to be deleted immediately (used to delete the Cookie)

1. String Cookie.getName()

Returns the name of the cookie.

1. String Cookie.getValue()

Returns the current value of the cookie

1. void HttpServletResponse.addCookie(Cookie cookieObj)

- This method adds the specified cookie to the response.

- This method can be called multiple times to set more than one cookie to the response.

1. Cookie[] HttpServletRequest.getCookies()

Returns an array containing all the Cookie objects or returns null if request doesn't have cookies

**Example:-**

1. Create the Cookie

//Non-Persistent Cookie

Cookie myNameCookie = **new** Cookie("MyName", "Praveen Dyamappa");

res.addCookie(myNameCookie);

//Persistent Cookie

Cookie myLocationCookie = **new** Cookie("MyLocation", "Bangalore");

myLocationCookie.setMaxAge(7\*24\*60\*60);

res.addCookie(myLocationCookie);

out.println("Cookies created");

1. Read the Cookie

Cookie[] cookies = req.getCookies();

**for**(Cookie cookie : cookies)

{

out.println(cookie.getName()+" ===== "+ cookie.getValue());

}

1. Remove Cookie from Browser

Cookie[] receivedCookies = req.getCookies();

**if**(receivedCookies != **null**)

{

out.println("Cookies Present in the Request");

**for**(Cookie receivedCookie : receivedCookies)

{

String name = receivedCookie.getName();

String value = receivedCookie.getValue();

out.println("Deleting the Cookie with Name : "+ name

+" Value : "+value);

receivedCookie.setMaxAge(0);

resp.addCookie(receivedCookie);

}

}**else**{

out.println("Cookies are NOT Present in the Request");

}

**Assignment 12:**

* Implement the “Remember Me on this Computer” functionality to studentsApp application. Use Cookies to implement this functionality. Set the Max age of the Cookies as 1 Year

**Assignment 13:**

* Modify the studentsApp application in such a way that if user has enabled the cookies allow him to access the application as usual
* If user has disabled the cookies then do not allow him to login to the application instead show the error message as

“Oops! Your browser seems to have cookies disabled. Make sure cookies are enabled or try opening a new browser window (Google Error Message)”

**===== ===== ===== ========= ======**

**Workshop Exercises**

**===== ===== ===== ===== ======== ======**

**Session**

* A session w.r.t web application is a time difference between Login & Logout.
* Http Session is a functionality provided by Servlet API which helps web application to uniquely identify the user & helps web application to maintain the state's between the requests using session attributes.

**Why do we need HttpSession functionality?**

* HTTP is a "stateless" protocol i. e. it does not maintain a relationship/state between requests also don't help web application to uniquely identify the user. Each request is un-related to any previous requests.
* Hence even if a user sends a sequence of requests to web application then it will not able to tell that those are from the same user also it will not be able to relate them

(Stateless Protocol example = http and State full Protocol example = ftp, telnet,etc )

* To address to this problem, Servlets provide HttpSession functionality.
* Key advantages of Session are
* Helps application to uniquely identify the user
* Session maintain state's between request with the help of session attributes
* Avoids authentication for each request
* Tightly couples the different pages of application

Thumb Rule: Any application which has Login page requirement then it MUST implement Session Handling mechanism.

* HttpSession uses one of the following two mechanisms to handle session:

1. Cookies (it’s default)
2. URL rewriting (explain URL rewriting at this point, refer the notes below to explain)

**Assignment 14:**

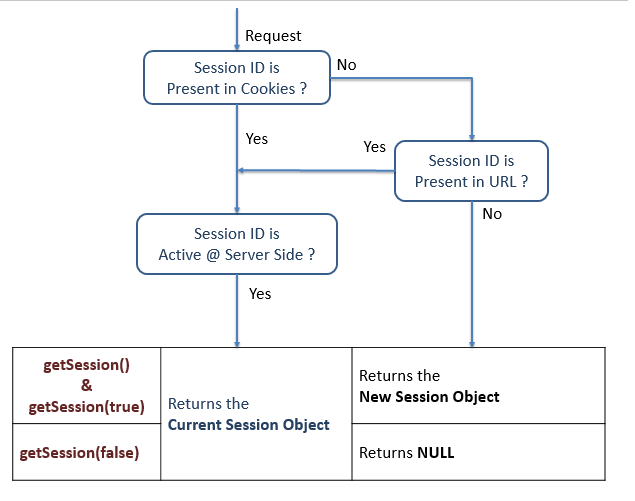
* In the header page of the studentsApp application enable “Create Profile” hyper link ONLY for admin user

**Assignment 15:**

* Modify the studentsApp application in such a way that, if user closes the browser without logging out & if he tries to login to the application next time within 7 days then he should be allowed to see the Home page directly. After 7 days he should be seeing Login Page

**Session Related Methods**

1. HttpSession HttpServletRequest.getSession()
2. HttpSession HttpServletRequest.getSession(boolean create)



1. void HttpSession .setMaxInactiveInterval(int interval)

Specifies the valid session time, in seconds. A negative time indicates the session should never timeout.

1. String HttpServletResponse.encodeURL(String url)
2. String HttpServletResponse.encodeRedirectURL(String url)

Encodes the specified URL by appending the session ID to it

1. void HttpSession.invalidate()

Invalidates the current session and garbage collects the associated session object

1. String getId()

This method returns the unique identifier generated for each session

1. Refer the Attribute section for Attribute related methods

out.println("1 \*\*\*\* "+req.isRequestedSessionIdFromCookie());

out.println("<BR>");

out.println("2 \*\*\*\* "+req.isRequestedSessionIdFromURL());

out.println("<BR>");

out.println("3 \*\*\*\* "+req.isRequestedSessionIdValid());

**Steps to work with session**

1. Create the session

When user tries to login to the application and after successful authentication, create the session for the first time

HttpSession session = req.getSession(); OR

HttpSession session = req.getSession(true);

1. Validate the session

Once the session is created, for the subsequent requests validate the session

HttpSession session = **req.getSession(false);**

**if**(session == **null**)

{

//Invalid Session

//Redirect it to the page which displays Error Message along with Login Page

}**else**{

//Valid Session

//Generate the proper response

}

1. Invalidate the session

A session get invalidated in following 3 ways

1. When application / server goes down
2. When user logout of the application

When user wants to logout of the application, invoke the invalidate() method on session object

HttpSession session = req.getSession(**true**);

session.invalidate();

1. When user is inactive for configured amount of time

There are two ways where we can configure the session timeout

1. In web.xml

<session-config>

<session-timeout>1</session-timeout> ===> Time in Minutes

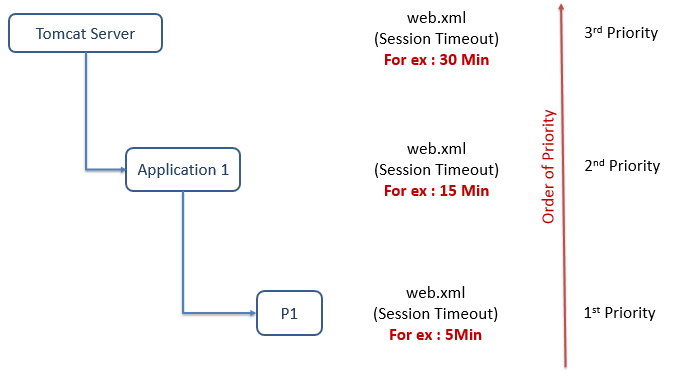
</session-config>

1. In Program

HttpSession session = req.getSession();

session.setMaxInactiveInterval(1\*60); ===> Time in Seconds

One advantage with the above approach is that you can set the different session timeouts to different type of user.



**URL Rewriting:**

* In handling user session, URL Rewriting comes into picture **ONLY if we tell server to encode URL’s**
* If we encode our URL’s with session id (Redirect / Forward / Hyperlink), the container will first attempt to use cookies for session management & fall back to URL rewriting only if cookie approach fails

Note: We cannot use URL Rewriting methods inside static pages

//Hyperlink URL

out.println("<a href=\" "+res.encodeURL("./myServlet")+" \" > Dynamic Page");

out.println("<a href=\" "+res.encodeURL("./index.html")+" \" > Static Page");

//Forward URL's

req.getRequestDispatcher(res.encodeURL("myServlet")).forward(req, res)

getServletContext().getRequestDispatcher(res.encodeURL("/myServlet"))

.forward(req, res);

//Redirect URL

res.sendRedirect(res.encodeRedirectURL("myServlet"));

**===== End of Servlets ======**