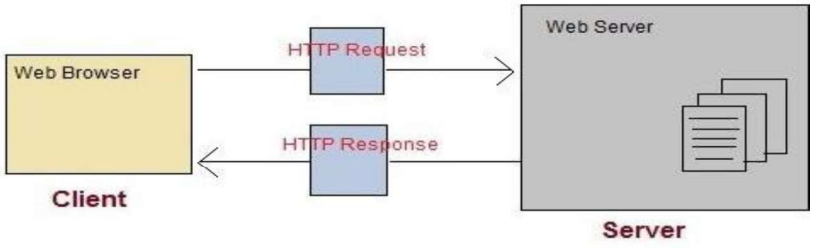
MODULE - 4

SERVLETS

Introduction to Web

The web clients make requests to web server. When a server answers a request, it usually sends some type of content(**MIME-** Multi purpose Internet Mail Exchange) to the client. The client uses web browser to send request to the server. The server often sends response to the browser with a set of instructions written in HTML(HyperText Markup Language)



Before Servlets, **CGI(Common Gateway Interface)** programming was used to create web applications.

Here's how a CGI program works :

Drawbacks of CGI programs

- High resposne time because CGI programs execute in their own OS shell.
- CGI is not scalable.
- CGI programs are not always secure or object-oriented.
- It is Platform dependent.

Because of these disadvantages, developers started looking for better CGI solutions. And then Sun Microsystems developed **Servlet** as a solution.

Servlet

Servlet technology is used to create web application (resides at server side and generates dynamic web page).

Servlet can be described in many ways, depending on the context.

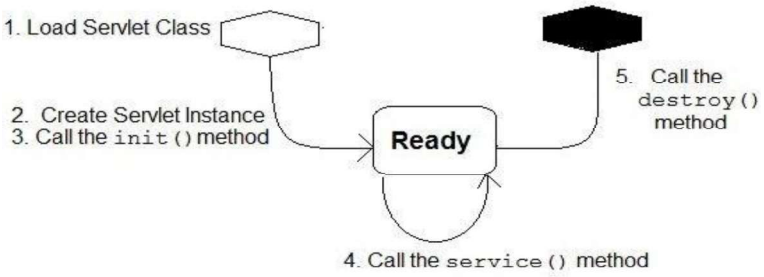
- Servlet is server side program.
- Servlet is an API that provides many interfaces and classes.
- Servlet is a web component that is deployed on the server to create dynamic web page.

Advantages of using Servlets

1. **better performance:** because it creates a thread for each request not process.
2. **Portability:** Servlets are platform independent because it uses java language.
3. **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
4. **Secure:** servlets are object oriented and runs inside JVM.
5. Servlets are scalable.

Life Cycle of a Servlet

The web container maintains the life cycle of a servlet instance



1. **Loading Servlet Class :** A Servlet class is loaded when first request for the servlet is received by the Web Container.
2. **Servlet instance creation :**After the Servlet class is loaded, Web Container creates the instance of it. Servlet instance is created only once in the life cycle.
3. **Call to the init() method :** init() method is called by the Web Container on servlet instance to initialize the servlet.

   public void **init**(ServletConfig config) throws ServletException

4. **Call to the service() method :** The containers call the service() method each time the request for servlet is received. The service() method will then call the doGet() or doPost() methos based ont eh type of the HTTP request, as explained in previous lessons.
   public void **service**(ServletRequest request, ServletResponse response) throws ServletException, IOException
5. **Call to destroy() method:** The Web Container call the destroy() method before removing servlet instance, giving it a chance for cleanup activity.

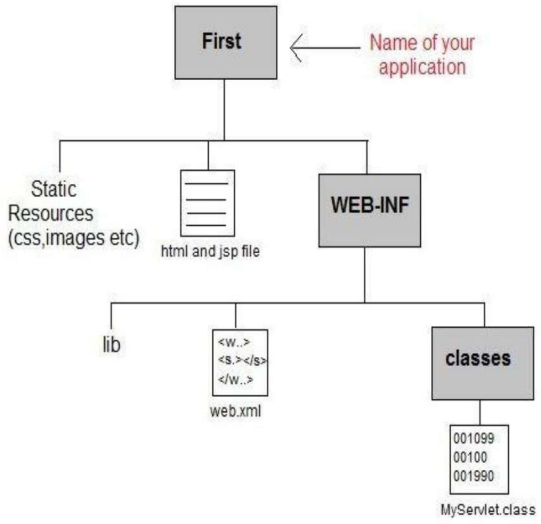
Steps to Create Servlet Application using tomcat server

To create a Servlet application you need to follow the below mentioned steps. These steps are common for all the Web server. In our example we are using Apache Tomcat server. Apache Tomcat is an open source web server for testing servlets and JSP technology. Create directory structure for your application.

1. Create directory structure for your application.
2. Create a Servlet
3. Compile the Servlet
4. Create Deployement Descriptor for your application
5. Start the server and deploy the application
6.

1. Creating the Directory Structure

Sun Microsystem defines a unique directory structure that must be followed to create a servlet application.



Create the above directory structureinside **Apache-Tomcat\webapps** directory.

All HTML, static files(images, css etc) are kept directly under **Web application** folder.

While all the Servlet classes are kept inside `classes` folder.

The `web.xml` (deployement descriptor) file is kept under `WEB-INF`folder.

2. Creating a Servlet

There are three different ways to create a servlet.

- By extending **HttpServlet** class
- By extending **GenericServlet** class
- By implementing **Servlet** interface

```
import javax.servlet.*; import
javax.servlet.http.*;import
java.io.*;

// extending HttpServlet class


public MyServlet extends HttpServlet
{
 public void doGet(HttpServletRequest request,HttpServletResposne response)throws
                ServletException
 {
  response.setContentType("text/html"); // set content type
  //get the stream to write the data
  PrintWriter out = response.getWriter(); // create printwriter object
```

```
    out.println("<h1>Hello Readers</h1>"); // print ur content on client web browser

    }
}
```

Write above code and save it as **MyServlet.java** anywhere on your PC. Compile it from there and paste the class file into WEB-INF/classes/ directory that you have to create inside **Tomcat/webapps** directory.

```
import javax.servlet.*;
import java.io.*;

// extending GenericServlet class


public MyServlet extends GenericServlet
{
 public void service(ServletRequest request,ServletResponse response)throws
    IOException,ServletException{
 {
   response.setContentType("text/html"); // set content type

   //get the stream to write the data
   PrintWriter out = response.getWriter(); // create printwriter object

   out.println("<h1>Hello Readers</h1>"); // print ur content on client web browser

   }
}
```

3. Compiling a Servlet

To compile a Servlet a JAR file is required. Different servers require different JAR files. In Apache Tomcat server servlet-api.jar file is required to compile a servlet class.

- Download **servlet-api.jar** file.
- Paste the servlet-api.jar file inside Java\jdk\jre\lib\ext directory.

NOTE: After compiling your Servlet class you will have to paste the class file into WEB-INF/classes/ directory.

4. Create Deployment Descriptor

Deployment Descriptor(DD) is an XML document that is used by Web Container to run Servlets and JSP pages.

web.xml file

```
<web-app>
    <servlet>
        <servlet-name> MyServlet </servlet-name>
        <servlet-class> MyServlet </servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name> MyServlet </servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```

5. Starting Tomcat Server for the first time

set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).

Go to My Computer properties -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path of jdk folder in variable value -> ok

Run Servlet Application

Open Browser and type **http:localhost:8080/First/hello**

Servlet API

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

1. **javax.servlet.*;**

INTERFACES	CLASSES
Servlet	ServletInputStream
ServletContext	ServletOutputStream
ServletConfig	ServletException
ServletRequest	UnavailableException
ServletResponse	GenericServlet

Interface Summary	
Servlet	Defines methods that all servlets must implement.
ServletRequest	Defines an object to provide client request information to a servlet.
ServletResponse	Defines an object to assist a servlet in sending a response to the client.
ServletConfig	A servlet configuration object used by a servlet container to pass information to a servlet during initialization.
ServletContext	Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.

Interface Servlet

Method	Description
void destroy( )	Called when the servlet is unloaded.
ServletConfig getServletConfig( )	Returns a **ServletConfig** object that contains any initialization parameters.
String getServletInfo( )	Returns a string describing the servlet.
void init(ServletConfig *sc*) throws ServletException	Called when the servlet is initialized. Initialization parameters for the servlet can be obtained from *sc*. An **UnavailableException** should be thrown if the servlet cannot be initialized.
void service(ServletRequest *req*, ServletResponse *res*) throws ServletException, IOException	Called to process a request from a client. The request from the client can be read from *req*. The response to the client can be written to *res*. An exception is generated if a servlet or IO problem occurs.

Interface ServletRequest

Method	Description
Object getAttribute(String *attr*)	Returns the value of the attribute named *attr*.
String getCharacterEncoding( )	Returns the character encoding of the request.
int getContentLength( )	Returns the size of the request. The value –1 is returned if the size is unavailable.
String getContentType( )	Returns the type of the request. A **null** value is returned if the type cannot be determined.
ServletInputStream getInputStream( ) throws IOException	Returns a **ServletInputStream** that can be used to read binary data from the request. An **IllegalStateException** is thrown if **getReader( )** has already been invoked for this request.
String getParameter(String *pname*)	Returns the value of the parameter named *pname*.
Enumeration getParameterNames( )	Returns an enumeration of the parameter names for this request.
String[ ] getParameterValues(String *name*)	Returns an array containing values associated with the parameter specified by *name*.
String getProtocol( )	Returns a description of the protocol.
BufferedReader getReader( ) throws IOException	Returns a buffered reader that can be used to read text from the request. An **IllegalStateException** is thrown if **getInputStream( )** has already been invoked for this request.
String getRemoteAddr( )	Returns the string equivalent of the client IP address.
String getRemoteHost( )	Returns the string equivalent of the client host name.
String getScheme( )	Returns the transmission scheme of the URL used for the request (for example, "http", "ftp").
String getServerName( )	Returns the name of the server.
int getServerPort( )	Returns the port number.

Interface ServletResponse

Method	Description
String getCharacterEncoding( )	Returns the character encoding for the response.
ServletOutputStream getOutputStream( ) throws IOException	Returns a **ServletOutputStream** that can be used to write binary data to the response. An **IllegalStateException** is thrown if **getWriter( )** has already been invoked for this request.
PrintWriter getWriter( ) throws IOException	Returns a **PrintWriter** that can be used to write character data to the response. An **IllegalStateException** is thrown if **getOutputStream( )** has already been invoked for this request.
void setContentLength(int *size*)	Sets the content length for the response to *size*.
void setContentType(String *type*)	Sets the content type for the response to *type*.

Interface ServletConfig

Method	Description
ServletContext getServletContext( )	Returns the context for this servlet.
String getInitParameter(String *param*)	Returns the value of the initialization parameter named *param*.
Enumeration getInitParameterNames( )	Returns an enumeration of all initialization parameter names.
String getServletName( )	Returns the name of the invoking servlet.

Interface ServletContext

Method	Description
Object getAttribute(String *attr*)	Returns the value of the server attribute named *attr*.
String getMimeType(String *file*)	Returns the MIME type of *file*.
String getRealPath(String *vpath*)	Returns the real path that corresponds to the virtual path *vpath*.
String getServerInfo( )	Returns information about the server.
void log(String *s*)	Writes *s* to the servlet log.
void log(String *s*, Throwable *e*)	Writes *s* and the stack trace for *e* to the servlet log.
void setAttribute(String *attr*, Object *val*)	Sets the attribute specified by *attr* to the value passed in *val*.

Class Summary	
GenericServlet	Defines a generic, protocol-independent servlet.
ServletInputStream	Provides an input stream for reading binary data from a client request, including an efficient readLine method for reading data one line at a time.
ServletOutputStream	Provides an output stream for sending binary data to the client.
ServletException	Defines a general exception a servlet can throw when it encounters difficulty.
UnavailableException	Defines an exception that a servlet or filter throws to indicate that it is permanently or temporarily unavailable.

Class GenericServlet

java.lang.Object

└ **javax.servlet.GenericServlet**

All Implemented Interfaces: Servlet, ServletConfig

Class ServletInputStream

java.lang.Object

└ java.io.InputStream

└ **javax.servlet.ServletInputStream**

Method Summary	
int	readLine(byte[] b, int off, int len) Reads the input stream, one line at a time.

Class ServletOutputStream

Method Summary	
void	println()
void	print(java.lang.String s) **Writes a String to the client, without a carriage return-line feed (CRLF) character at the end.**
void	println() Writes a carriage return-line feed (CRLF) to the client.
void	println(java.lang.String s) Writes a String to the client, followed by a carriage return-line feed (CRLF).

2. **javax.servlet.http.*;**

CLASSES INTERFACES	
Cookie	HttpServletRequest
	HttpServletRespons
HttpSessionBindingEvent	HttpSession

Interface Summary	
HttpServletRequest	Extends the ServletRequest interface to provide request information for HTTP servlets.
HttpServletResponse	Extends the ServletResponse interface to provide HTTP-specific functionality in sending a response.

HttpSession	Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

Interface HttpServletResponse

Method	Description
void addCookie(Cookie *cookie*)	Adds *cookie* to the HTTP response.
boolean containsHeader(String *field*)	Returns **true** if the HTTP response header contains a field named *field*.
String encodeURL(String *url*)	Determines if the session ID must be encoded in the URL identified as *url*. If so, returns the modified version of *url*. Otherwise, returns *url*. All URLs generated by a servlet should be processed by this method.
String encodeRedirectURL(String *url*)	Determines if the session ID must be encoded in the URL identified as *url*. If so, returns the modified version of *url*. Otherwise, returns *url*. All URLs passed to **sendRedirect( )** should be processed by this method.
void sendError(int *c*) throws IOException	Sends the error code *c* to the client.
void sendError(int *c*, String *s*) throws IOException	Sends the error code *c* and message *s* to the client.
void sendRedirect(String *url*) throws IOException	Redirects the client to *url*.
void setDateHeader(String *field*, long *msec*)	Adds *field* to the header with date value equal to *msec* (milliseconds since midnight, January 1, 1970, GMT).
void setHeader(String *field*, String *value*)	Adds *field* to the header with value equal to *value*.
void setIntHeader(String *field*, int *value*)	Adds *field* to the header with value equal to *value*.
void setStatus(int *code*)	Sets the status code for this response to *code*.

Interface HttpServletRequest

Method	Description
String getAuthType( )	Returns authentication scheme.
Cookie[ ] getCookies( )	Returns an array of the cookies in this request.
long getDateHeader(String *field*)	Returns the value of the date header field named *field*.
String getHeader(String *field*)	Returns the value of the header field named *field*.
Enumeration getHeaderNames( )	Returns an enumeration of the header names.
int getIntHeader(String *field*)	Returns the **int** equivalent of the header field named *field*.
String getMethod( )	Returns the HTTP method for this request.
String getPathInfo( )	Returns any path information that is located after the servlet path and before a query string of the URL.
String getPathTranslated( )	Returns any path information that is located after the servlet path and before a query string of the URL after translating it to a real path.
String getQueryString( )	Returns any query string in the URL.
String getRemoteUser( )	Returns the name of the user who issued this request.
String getRequestedSessionId( )	Returns the ID of the session.
String getRequestURI( )	Returns the URI.
StringBuffer getRequestURL( )	Returns the URL.
String getServletPath( )	Returns that part of the URL that identifies the servlet.
HttpSession getSession( )	Returns the session for this request. If a session does not exist, one is created and then returned.
HttpSession getSession(boolean *new*)	If *new* is **true** and no session exists, creates and returns a session for this request. Otherwise, returns the existing session for this request.
boolean isRequestedSessionIdFromCookie( )	Returns **true** if a cookie contains the session ID. Otherwise, returns **false**.
boolean isRequestedSessionIdFromURL( )	Returns **true** if the URL contains the session ID. Otherwise, returns **false**.
boolean isRequestedSessionIdValid( )	Returns **true** if the requested session ID is valid in the current session context.

Method	Description
Object getAttribute(String *attr*)	Returns the value associated with the name passed in *attr*. Returns **null** if *attr* is not found.
Enumeration getAttributeNames( )	Returns an enumeration of the attribute names associated with the session.
long getCreationTime( )	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when this session was created.
String getId( )	Returns the session ID.
long getLastAccessedTime( )	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the client last made a request for this session.
void invalidate( )	Invalidates this session and removes it from the context.
boolean isNew( )	Returns **true** if the server created the session and it has not yet been accessed by the client.
void removeAttribute(String *attr*)	Removes the attribute specified by *attr* from the session.
void setAttribute(String *attr*, Object *val*)	Associates the value passed in *val* with the attribute name passed in *attr*.

Class Summary	
Cookie	Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server.
HttpServlet	Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site.
HttpSessionBindingEvent	Events of this type are either sent to an object that implements HttpSessionBindingListener when it is bound or unbound from a session, or to a HttpSessionAttributeListener that has been configured in the deployment descriptor when any attribute is bound, unbound or replaced in a session.
HttpSessionEvent	This is the class representing event notifications for changes to sessions within a web application.

Class Cookie

Method	Description
Object clone( )	Returns a copy of this object.
String getComment( )	Returns the comment.
String getDomain( )	Returns the domain.
int getMaxAge( )	Returns the maximum age (in seconds).
String getName( )	Returns the name.
String getPath( )	Returns the path.
boolean getSecure( )	Returns **true** if the cookie is secure. Otherwise, returns **false**.
String getValue( )	Returns the value.
int getVersion( )	Returns the version.
void setComment(String *c*)	Sets the comment to *c*.
void setDomain(String *d*)	Sets the domain to *d*.
void setMaxAge(int *secs*)	Sets the maximum age of the cookie to *secs*. This is the number of seconds after which the cookie is deleted.
void setPath(String *p*)	Sets the path to *p*.
void setSecure(boolean *secure*)	Sets the security flag to *secure*.
void setValue(String *v*)	Sets the value to *v*.
void setVersion(int *v*)	Sets the version to *v*.

ClassHttpServlet

Method	Description
void doDelete(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP DELETE request.
void doGet(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP GET request.
void doHead(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP HEAD request.
void doOptions(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP OPTIONS request.
void doPost(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP POST request.
void doPut(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP PUT request.
void doTrace(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Handles an HTTP TRACE request.
long getLastModified(HttpServletRequest *req*)	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the requested resource was last modified.
void service(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Called by the server when an HTTP request arrives for this servlet. The arguments provide access to the HTTP request and response, respectively.

Class HttpSessionBindingEvent

Method Summary	
java.lang.String	**getName**() Returns the name with which the attribute is bound to or unbound from the session.
HttpSession	**getSession**() Return the session that changed.
java.lang.Object	**getValue**()

	Returns the value of the attribute that has been added, removed or replaced.

Class HttpSessionEvent

Method Summary	
HttpSession	**getSession**() Return the session that changed.

Reading Servlet parameters

In this example, we will show how a parameter is passed to a

index.html

```
<form method="post" action="check">
        Name <input type="text" name="user" >
        <input type="submit" value="submit">
</form>
```

MyServlet.java

```
public class MyServlet extends HttpServlet {


  protected void doPost(request, response){
     ... // set content type . . .
        String user=request.getParameter("user");

        out.println("<h2> Welcome "+user+"</h2>");
   }
}
```

NOTE: getParameter( ) returns string to get int value use

Integer.parseInt("sting");

Handling HTTP request and Response

HttpServlet class provides various methods that handle various types of HTTP request.

A servlet typically must override at least one method, usually one of these:

- doGet, if the servlet supports HTTP GET requests
- doPost, for HTTP POST requests
- doPut, for HTTP PUT requests
- doDelete, for HTTP DELETE requests

GET and POST methods are commonly used when handling form input.

NOTE: By default a request is Get request.

GET Request	POST Request
Data is sent in header to the server	Data is sent in the request body
Get request can send only limited amount of data	Large amount of data can be
sent.	

Difference between GET and POST requests

Session

Session simply means a particular interval of time. **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

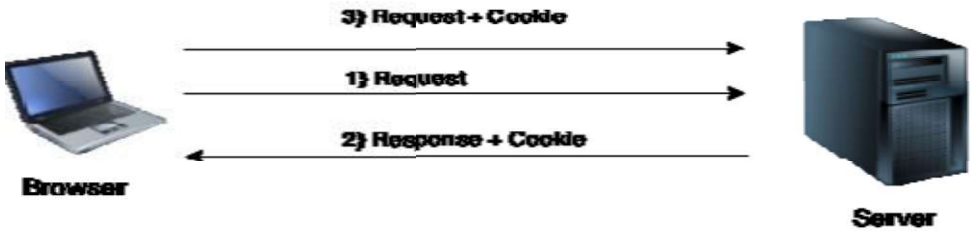
There are 2 techniques used in Session tracking:

1. **Cookies**
2. **HttpSession**

Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

By default, each request is considered as a new request. In coockie technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

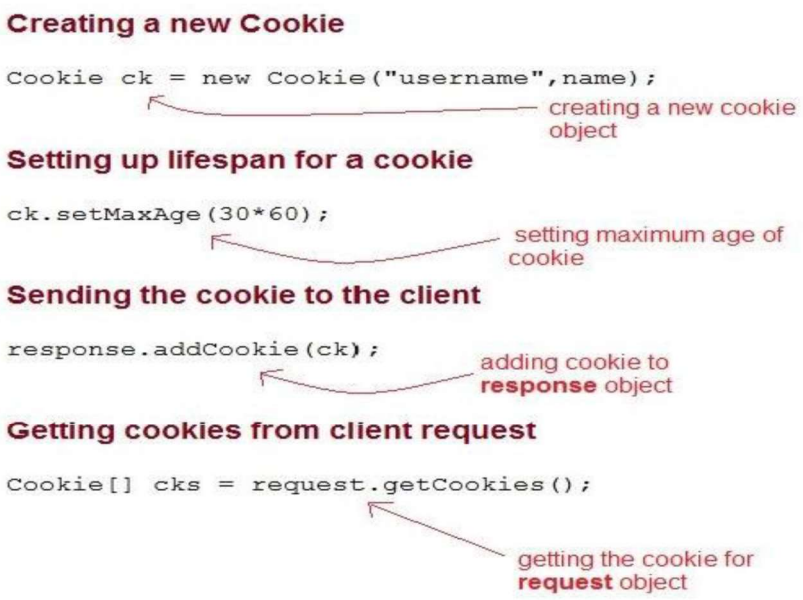


Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.



Example demonstrating usage of Cookies

index.html

```
<form method="post" action=" MyServlet ">
        Name:<input type="text" name="user" /><br/>
        Password:<input type="text" name="pass" ><br/>
            <input type="submit" value="submit">
</form>
```

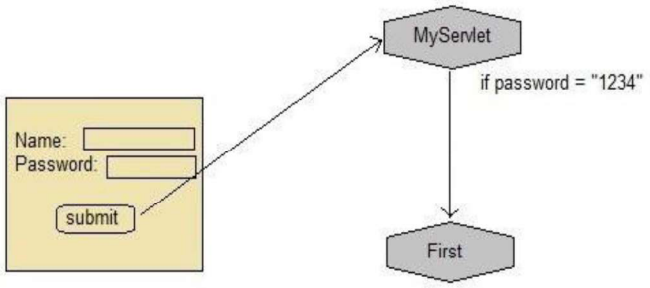
MyServlet.java

```
public class MyServlet extends HttpServlet {


  protected void doPost(HttpServletRequest request, HttpServletResponse response)
     {     // . . . . .
       String name =
       request.getParameter("user");String pass =
       request.getParameter("pass");


       if(pass.equals("1234"))
       {
          Cookie ck = new Cookie("username",name);
          response.addCookie(ck);


                        //response.sendRedirect("First");//call ur servlet


                             //creating submit button
                            out.print("<form action= First >");
                            out.print("<input type='submit' value='go'>");
                             out.print("</form>");
       }
    }
}
```

First.java

```
public class First extends HttpServlet {
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
  {     // . . .
     Cookie[] cks = request.getCookies();
     out.println("Welcome "+cks[0].getValue());

  }
}
```



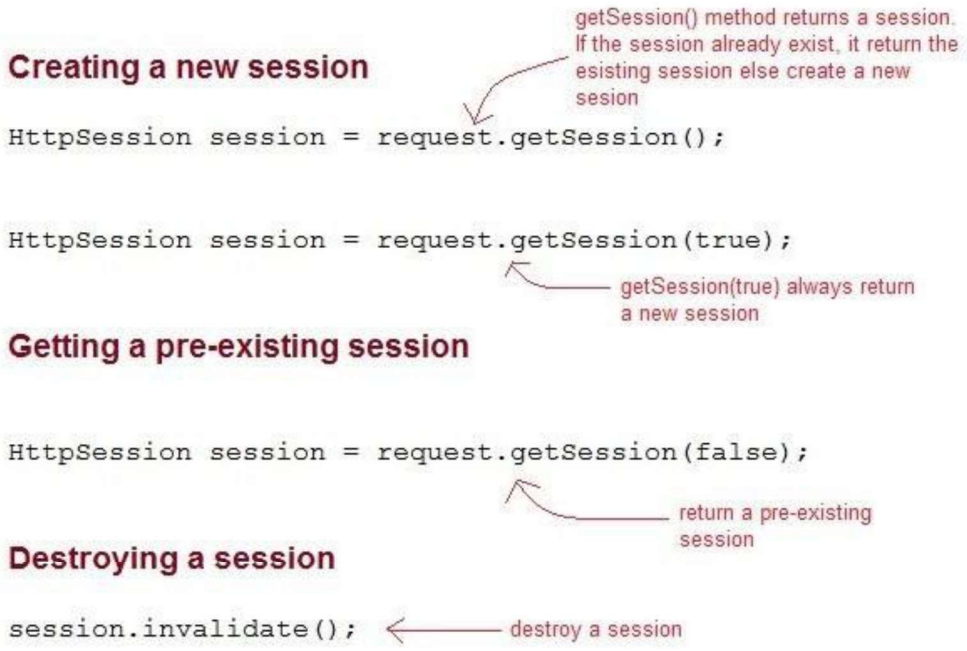
Useful Methods of Cookie class

Method	Description
public **void setMaxAge(int** expiry)	**Sets** the maximum age of the cookie in
public String getName()	**Returns** the name of the cookie. The name cannot be changed after
public String getValue()	**Returns** the value of the cookie.
public void setName(String name	**changes** the name of the
public void setValue(String value)	**changes** the value of the cookie.

Other methods required for using Cookies

HttpSession

HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from **HttpSession** object. Any servlet can have access to **HttpSession** object throughout the getSession() method.



Some Important Methods of HttpSession

Methods	Description
	returns the time when the session was created, measured in milliseconds since midnight January 1, 1970 GMT

String getId()	returns a string containing the unique identifier assigned to the session.
int getMaxInactiveInterval()	returns the maximum time interval, in seconds.
void invalidate()	destroy the session
boolean isNew()	returns true if the session is new else false

Complete Example demonstrating usage of HttpSession

index.html

```
<form method="post" action="Validate">
  User: <input type="text" name="uname " /><br/>
    <input type="submit" value="submit">
</form>
```

Validate.java

```
public class Validate extends HttpServlet {


   protected void doPost(request, response)
{
      // . . . .
       String name = request.getParameter("user");
                  //creating a session
          HttpSession session =
          request.getSession();
          session.setAttribute("user"
          , uname);
          response.sendRedirect("W
          elcome");

       }
}
```

Welcome.java

```
public class Welcome extends HttpServlet {


   protected void doGet(request, response){
       // . . . .
      HttpSession session = request.getSession();
      String user =
      (String)session.getAttribute("user
      ");out.println("Hello "+user);
   }
}
```