Servlets are protocol and platform independent server-side software components, written in Java. They run inside a Java enabled server or application server, such as the WebSphere Application Server. Servlets are loaded and executed within the Java Virtual Machine (JVM) of the Web server or application server, in much the same way that applets are loaded and executed within the JVM of the Web client. Since servlets run inside the servers, however, they do not need a graphical user interface (GUI). In this sense, servlets are also faceless objects. Servlets more closely resemble Common Gateway Interface (CGI) scripts or programs than applets in terms of functionality. As in CGI programs, servlets can respond to user events from an HTML request, and then dynamically construct an HTML response that is sent back to the client. Servlet process flow Servlets implement a common request/response paradigm for the handling of the messaging between the client and the server. The Java Servlet API defines a standard interface for the handling of these request and response messages between the client and server.

Figure 31 shows a high-level client-to-servlet process flow:

1. The client sends a request to the server.

2. The server sends the request information to the servlet.

3. The servlet builds a response and passes it to the server. That response is dynamically built, and the content of the response usually depends on the client's request. External resources may also be used.

4. The server sends the response back to the client.

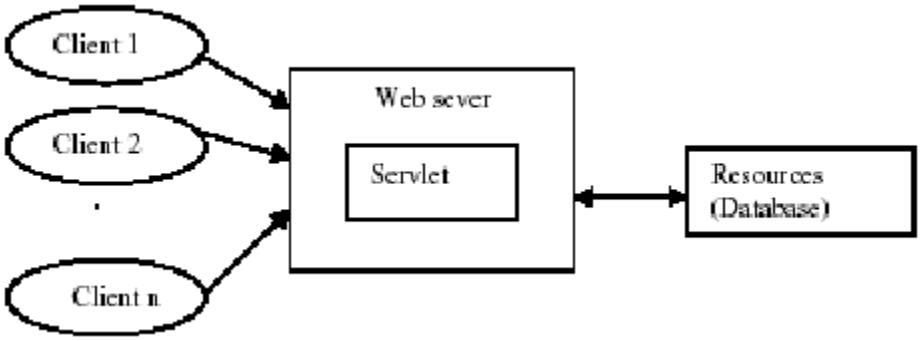


Figure 32. Basic client-to-servlet interaction

The life cycle of a servlet is expressed in the Java Servlet API in the init, service (doGet or doPost), and destroy methods of the Servlet interface. We will discuss the functions of these methods in more detail and the objects that they manipulate. Figure 33 is a visual diagram of the life-cycle of an individual servlet.

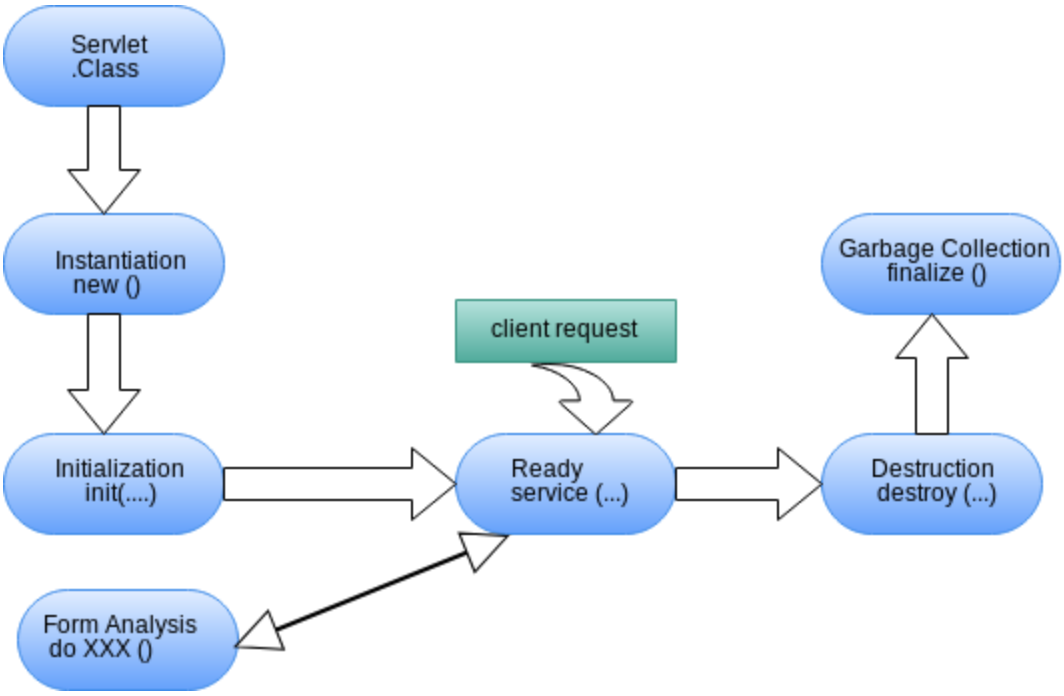


Figure 33. Servlet life-cycle

The WebSphere administrator can set an application and its servlets to be unavailable for service. In such cases, the application and servlets remain unavailable until the administrator changes them to available. Understanding the life-cycle This section describes in detail some of the important servlet life-cycle methods of the Java Servlet API.

Servlet Initialization: init method Servlets can be dynamically loaded and instantiated when their services are first requested, or the Web server can be configured so that specific servlets are loaded and instantiated when the Web server initializes. In either case, the init method of the servlet performs any necessary servlet initialization, and is guaranteed to be called once for each servlet instance, before any requests to the servlet are handled. An example of a task which may be performed in the init method is the loading of default data parameters or database connections. The most common form of the init method of the servlet accepts a ServletConfig object parameter. This interface object allows the servlet to access name/value pairs of initialization parameters that are specific to that servlet. The ServletConfig object also gives us access to the SevletContext object that describes information about our servlet environment. Each of these objects will be discussed in more detail in the servlet examples sections. Servlet request handling Once the servlet has been properly initialized, it may handle requests (although it is possible that a loaded servlet may get no requests). Each request is represented by a ServletRequestobject, and the corresponding response by a ServletResponse object in the Java Servlet API. Since we will be dealing with HttpServlets, we will deal exclusively with the more specialized HttpServletRequest and HttpServletResponse objects. The HttpServletRequest object encapsulates information about the client request, including information about the client's environment and any data that may have been sent from the client to the servlet. The HttpServletRequest class contains methods for extracting this information from the request object. The HttpServletResponse is often the

dynamically generated response, for instance, an HTML page which is sent back to the client. It is often built with data from the HttpServletRequest object. In addition to an HTML page, a response object may also be an HTTP error response, or a redirection to another URL, servlet, or JavaServer Page. Each time a client request is made, a new servlet thread is spawned which services the request. In this way, the server can handle multiple concurrent requests to the same servlet. For each request, usually the service, doGet, or doPost methods will be called. These methods are passed the HttpServletRequest and HttpServletResponse parameter objects.

doPost: Invoked whenever an HTTP POST request is issued through an HTML form. The parameters associated with the POST request are communicated from the browser to the server as a separate HTTP request. The doPost method should be used whenever modifications on the server will take place. doGet: Invoked whenever an HTTP GET method from a URL request is issued, or an HTML form. An HTTP GET method is the default when a URL is specified in a Web browser. In contrast to the doPost method, doGet should be used when no modifications will be made on the server, or when the parameters are not sensitive data. The parameters associated with a GET request are appended to the end of the URL, and are passed into the QueryString property of the HttpServletRequest. Other servlet methods worth mentioning destroy: The destroy method is called when the Web server unloads the servlet. A subclass of HttpServlet only needs to implement this method if it needs to perform cleanup operations, such as releasing database connections or closing files. getServletConfig: The getServletConfig method returns a ServletConfig instance that can be used to return the initialization parameters and the ServletContext object. getServletInfo: The getServletInfo method is a method that can provide information about the servlet, such as its author, version, and copyright. This method is generally overwritten to have it return a meaningful value for your application. By default, it returns an empty string.

Basic servlet structure:

packageitso.servjsp.servletapi;

Figure 35.SimpleHttpServlet package declaration

shows the import statements used to give us access to other Java packages. The import of java.io is so that we have access to some standard IO classes. More importantly, the javax.servlet.* and javax.servlet.http.* import statements give us access to the Java Servlet API set of classes and interfaces.

shows the import statements used to give us access to other Java packages.

The import of java.io is so that we have access to some standard IO classes.

More importantly, the javax.servlet.* and javax.servlet.http.* import statements give us access to the Java Servlet API set of classes and interfaces.

shows the SimpleHttpServlet class declaration. We extend the HttpServlet class (javax.servlet.http.HttpServlet) to make our class an HTTP protocol servlet.

public class SimpleHttpServlet extends HttpServlet {

Figure 37. The SimpleHttpServlet class declaration

is the heart of this servlet, the implementation of the service method for the handling of the request and response objects of the servlet

```
protected void service (HttpServletRequestreq, HttpServletResponse res) throws
ServletException, IOException {
res.setContentType("text/html");
PrintWriter out = res.getWriter();
out.println("<html><title>Simple http servlet</title>");
out.println("Servlet API Example - SimpleHttpServlet");
out.println("This is about as simple a servlet as it gets!");
out.println("</Body></html>");
out.close();
```

fig: SimpleHttpServlet service method

What the service method does Let's examine this service method in more detail. Notice that the method accepts two parameters, HttpServletRequest and HttpServletResponse. The request object contains information about and from the client. In this example, we don't do anything with the request. This method is declared Abstract in the basic GenericServlet class, and so subclasses, such as HttpServlet, must override it. In our subclass of HttpServlet, when using this method, we must implement this method according to the signature defined in HttpServlet, namely, that it accepts HttpServletRequest and HttpServletResponse arguments. We do some handling of the response object, which is responsible for sending our response back to the client. Our response here is a formatted HTMLpage, so we first set the response content type to text/html by coding res.setContentType("text/html"). Next, we request a PrintWriter object to write text to the response by coding PrintWriter out = res.getWriter(). We could also have used a ServletOutputStream object to write out our response, but getWriter gives us more flexibility with Internationalization. In either case, the content type of the response must be set before references to these objects can be made. The remaining out.println statements write our HTML to the PrintWriter, which is sent back to the client as our response. It is pretty simple HTML, so we do not display it here. We use out.close more for completeness, because the Web application server automatically closes the PrintWriter when the service method exits.

How the servlet gets invoked We could invoke this servlet with either a GET or POST form action method; the service method will execute for either. If we knew something about how this servlet was ultimately to be called, for instance, what the HTML form method was going to be, we could have implemented the above functionality through specific doGet or doPost methods. The result would be the same. The simplest way to invoke the servlet would be by specifying a URL in the Web browser. This does not work for every servlet, but would work for the above example. A URL forces the Web browser to send the request using GET, similar to the way a standard HTML page is requested.

Running the servlet At this point we have not discussed the specifics of running servlets in a Web server environment. If you want to run this servlet, you should be able to follow the steps in Chapter 7, “Development and testing with VisualAge for Java” on page 167, code the SimpleHttpServlet, and run it under the WebSphere Test Environment. The WebSphere Test Environment provides a simulated Web server environment within the VisualAge for Java product and enables you to test and debug your servlets. Later, in Chapter 6, “WebSphere Application Server” on page 123, we discuss deploying servlets to the actual application server environment.

init method :

This servlet implements the init method. The initmethod only prints a message to standard output and call the super-class constructor. As we mentioned before, the init method is called only once, when the servlet is loaded. This message, therefore, should only be printed to the Web server’s console or log once (wherever standard output is defined), regardless of how many times the servlet is actually invoked.

doGet method :

We decided that this servlet is always called through a GET request, we have chosen to implement the doGet method, instead of the more generic service method. We developed a performTask method to which we pass a method posting type and a target URL.

doPost method :

Incidentally, this servlet has been designed to handle the particular type of request from the HTML page that was generated in the previous servlet example. In that HTML page, the user could fill out information in the form and submit it. The action in the HTML form causes the HTMLFormHandler servlet to be invoked, and the doPost request handler method to be called: In the doPost method, we handle the HttpServletResponse in the same way as before, except that this time, we are also handling the HttpServletRequest.

Getting form values:

We use the getParameter method of the request to extract the values of the request parameters (name/value fields passed in from the HTML page). We extract parameters named firstname and title from the request: req.getParameter("firstname") req.getParameter("title") These are two of the input fields that were passed from the HTML form. The getParameter method requires as an argument the name of the parameter that we want to extract (so it must be known), and returns the value of that parameter, or null. To get a list of the all parameter names, we could use the getParameterNames method. This method returns an enumeration of all the parameter names in the request, which we could then iterate through to get the individual parameter values. To extract the value of the tools parameter, however, we must apply a slightly different technique. The tools’ parameter is a multi-value input field (in this case, a checkbox). Because there could be more than one value to extract, we use the getParameterValues method, which returns an array of values.

Servlet initialization parameters :

The SimpleInitServlet servlet shows how to retrieve initialization parameters from the servlet configuration object (Figure 45). ServletConfig object The ServletConfig object is a parameter that can be passed into the init method of the servlet. You can also get the ServletConfig object from the request object through the getServletConfig method, but it is most commonly used in the init method to initialize the servlet's instance variables. Methods of the ServletConfig object allow us to extract the parameter information from this object. This parameter information is in a name/value pairs format, and can be stored in a file in XML format. We do not have to read the file, however, because the methods of the class provide us with some handy helper methods. What this servlet does This servlet simply extracts the parameter information from the configuration file, and stores those values in instance variables. It then echoes this information back to the client that invoked the servlet. In a real-life application, these variables would most likely be used to make a connection to the database, and this connection would be stored in a global instance variable for later use in the doGet method.

HTTP request handling utility servlet:

We next look at a servlet, ServletEnvironmentSnoop. Because the source for this servlet is rather large, we have chosen to include it in Appendix B, "Utility servlet and utility JSP" on page 407. This is a good utility servlet that extracts a lot of information from the request, and echoes its contents back to the client in the response. You should spend some time looking through the source code to see what kind of data can be extracted from a request object, and how to manipulate that data. Use this servlet as a future reference. Sample output of this servlet is also included in the appendix. The ServletEnvironmentSnoop servlet demonstrates the handling of the following request data:

❑ Request information—HTTP specific request information

❑ Request header— data passed in the header of the request, such as the character and encoding sets

❑ Request parameters—name/value pairs of parameter data

❑ Request attribute names—attributes of the class

❑ Request cookies—an array containing all cookies present in the request

❑ Servlet configuration—values used for initializing the servlet

❑ Servlet context attributes—information about the environment where the application server is running

❑ Session information—session data associated with the request

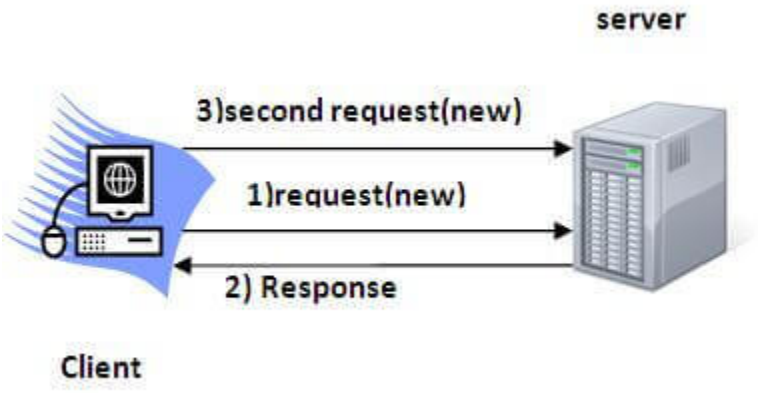
Session Tracking in Servlets

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.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Session Tracking Techniques

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

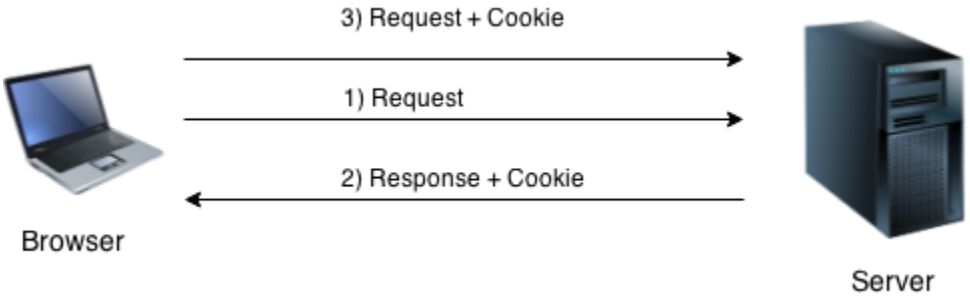
Cookie servlet

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

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies 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.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

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.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

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

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. Th

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add c response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies browser.

How to create Cookie?

Let's see the simple code to create cookie.

```
Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
response.addCookie(ck);//adding cookie in the response
```

How to delete Cookie?

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

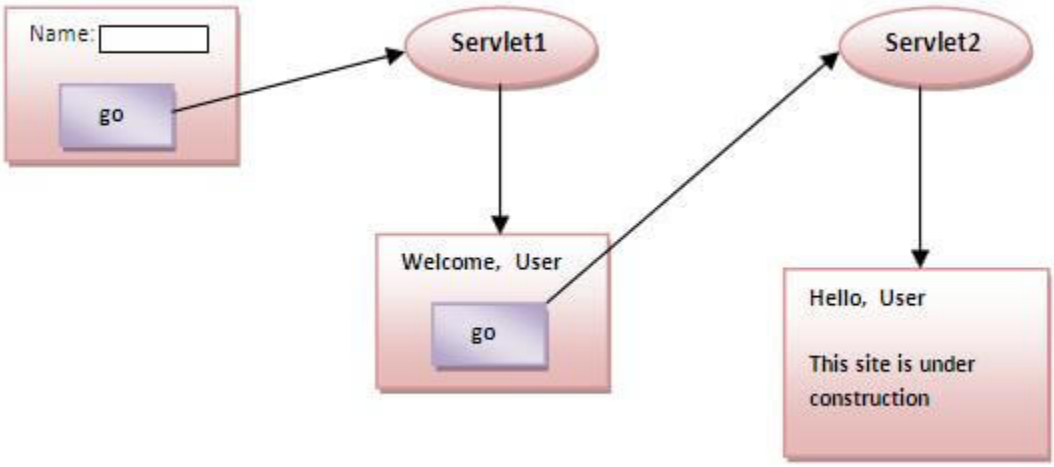
How to get Cookies?

Let's see the simple code to get all the cookies.

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
}
```

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

```
1. <form action="servlet1" method="post">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>
```

FirstServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5.
6. public class FirstServlet extends HttpServlet {
7.
8.   public void doPost(HttpServletRequest request, HttpServletResponse response){
9.     try{
10.
11.    response.setContentType("text/html");
12.    PrintWriter out = response.getWriter();
```

```
13.
14.     String n=request.getParameter("userName");
15.     out.print("Welcome "+n);
16.
17.     Cookie ck=new Cookie("uname",n);//creating cookie object
18.     response.addCookie(ck);//adding cookie in the response
19.
20.     //creating submit button
21.     out.print("<form action='servlet2'>");
22.     out.print("<input type='submit' value='go'>");
23.     out.print("</form>");
24.
25.     out.close();
26.
27.         }catch(Exception e){System.out.println(e);}
28.   }
29. }
```

SecondServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.
5.  public class SecondServlet extends HttpServlet {
6.
7.  public void doPost(HttpServletRequest request, HttpServletResponse response){
8.      try{
9.
10.     response.setContentType("text/html");
11.     PrintWriter out = response.getWriter();
12.
13.     Cookie ck[]=request.getCookies();
14.     out.print("Hello "+ck[0].getValue());
15.
16.     out.close();
```

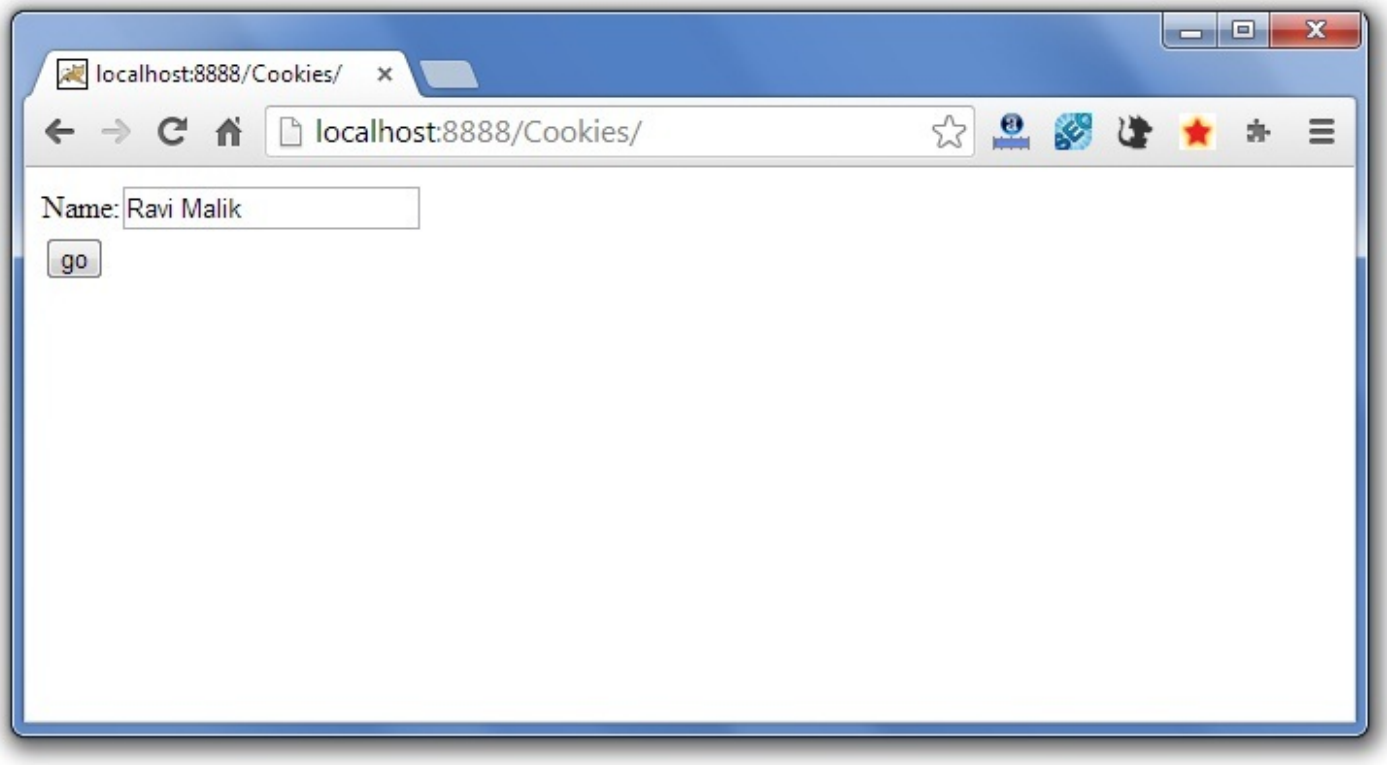
```
17.
18.         }catch(Exception e){System.out.println(e);}
19.     }
20.
21.
22. }
```

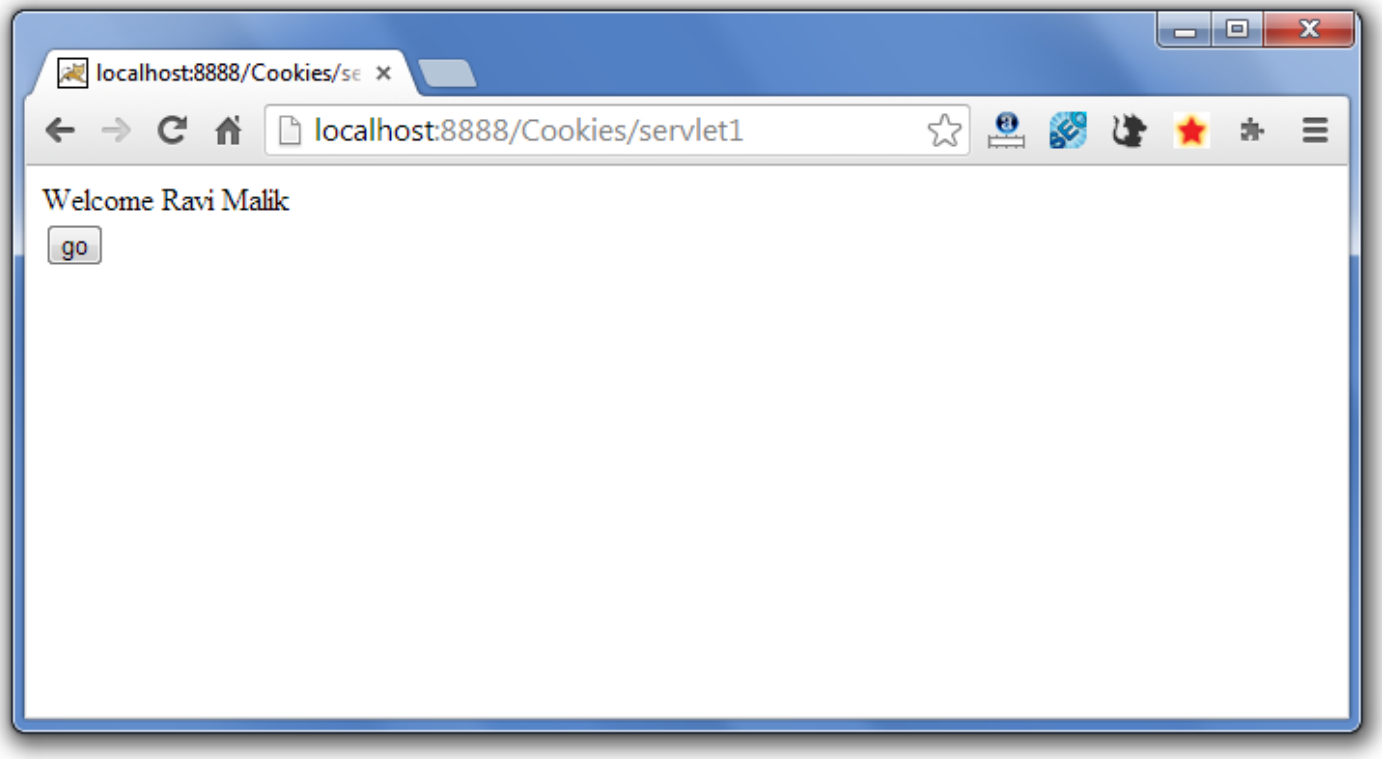
web.xml

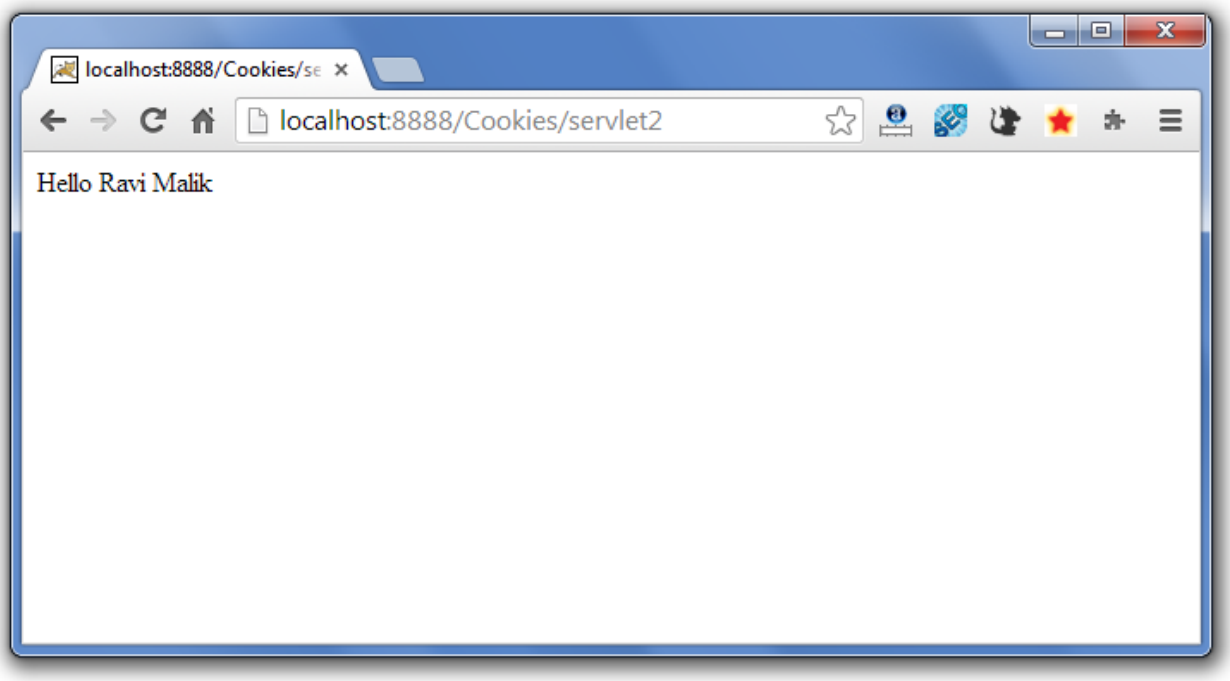
```
1.  <web-app>
2.
3.  <servlet>
4.  <servlet-name>s1</servlet-name>
5.  <servlet-class>FirstServlet</servlet-class>
6.  </servlet>
7.
8.  <servlet-mapping>
9.  <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
12.
13. <servlet>
14. <servlet-name>s2</servlet-name>
15. <servlet-class>SecondServlet</servlet-class>
16. </servlet>
17.
18. <servlet-mapping>
19. <servlet-name>s2</servlet-name>
20. <url-pattern>/servlet2</url-pattern>
21. </servlet-mapping>
22.
23. </web-app>
```

download this example (developed using Myeclipse IDE)
download this example (developed using Eclipse IDE)
download this example (developed using Netbeans IDE)

Output


localhost:8888/Cookies/
localhost:8888/Cookies/
Name: Ravi Malik
go


localhost:8888/Cookies/se
localhost:8888/Cookies/servlet1
Welcome Ravi Malik
go



Next>><<Prev

Servlet Login and Logout Example using Cookies

A **cookie** is a kind of information that is stored at client side.

In the previous page, we learned a lot about cookie e.g. how to create cookie, how to delete cookie, how to get cookie etc.

Here, we are going to create a login and logout example using servlet cookies.

In this example, we are creating 3 links: login, logout and profile. User can't go to profile page until he/she is logged in. If user is logged out, he need to login again to visit profile.

In this application, we have created following files.

1. index.html
2. link.html
3. login.html
4. LoginServlet.java

5. LogoutServlet.java
6. ProfileServlet.java
7. web.xml

File: index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Servlet Login Example</title>
</head>
<body>

<h1>Welcome to Login App by Cookie</h1>
<a href="login.html">Login</a>|
<a href="LogoutServlet">Logout</a>|
<a href="ProfileServlet">Profile</a>

</body>
</html>
```

File: link.html

```
<a href="login.html">Login</a> |
<a href="LogoutServlet">Logout</a> |
<a href="ProfileServlet">Profile</a>
<hr>
```

File: login.html

```
<form action="LoginServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
<input type="submit" value="login">
</form>
```

File: LoginServlet.java

```
1.  package com.javatpoint;
2.  
3.  import java.io.IOException;
4.  import java.io.PrintWriter;
5.  import javax.servlet.ServletException;
6.  import javax.servlet.http.Cookie;
7.  import javax.servlet.http.HttpServlet;
8.  import javax.servlet.http.HttpServletRequest;
9.  import javax.servlet.http.HttpServletResponse;
10. public class LoginServlet extends HttpServlet {
11.   protected void doPost(HttpServletRequest request, HttpServletResponse response)
12.                         throws ServletException, IOException {
13.       response.setContentType("text/html");
14.       PrintWriter out=response.getWriter();
15.   
16.       request.getRequestDispatcher("link.html").include(request, response);
17.   
18.       String name=request.getParameter("name");
19.       String password=request.getParameter("password");
20.   
21.       if(password.equals("admin123")){
22.           out.print("You are successfully logged in!");
23.           out.print("<br>Welcome, "+name);
24.   
25.           Cookie ck=new Cookie("name",name);
26.           response.addCookie(ck);
27.       }else{
28.           out.print("sorry, username or password error!");
29.           request.getRequestDispatcher("login.html").include(request, response);
30.       }
31.   
32.       out.close();
33.   }
```

```
34. 
35. }
```

File: LogoutServlet.java

```
1.  package com.javatpoint;
2.  
3.  import java.io.IOException;
4.  import java.io.PrintWriter;
5.  import javax.servlet.ServletException;
6.  import javax.servlet.http.Cookie;
7.  import javax.servlet.http.HttpServlet;
8.  import javax.servlet.http.HttpServletRequest;
9.  import javax.servlet.http.HttpServletResponse;
10. public class LogoutServlet extends HttpServlet {
11.   protected void doGet(HttpServletRequest request, HttpServletResponse response)
12.                   throws ServletException, IOException {
13.       response.setContentType("text/html");
14.       PrintWriter out=response.getWriter();
15.   
16.   
17.       request.getRequestDispatcher("link.html").include(request, response);
18.   
19.       Cookie ck=new Cookie("name","");
20.       ck.setMaxAge(0);
21.       response.addCookie(ck);
22.   
23.       out.print("you are successfully logged out!");
24.   }
25. }
```

File: ProfileServlet.java

```
1. package com.javatpoint;
2.
```

```
3. import java.io.IOException;
4. import java.io.PrintWriter;
5. import javax.servlet.ServletException;
6. import javax.servlet.http.Cookie;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10. public class ProfileServlet extends HttpServlet {
11.     protected void doGet(HttpServletRequest request, HttpServletResponse response)
12.                         throws ServletException, IOException {
13.         response.setContentType("text/html");
14.         PrintWriter out=response.getWriter();
15.
16.         request.getRequestDispatcher("link.html").include(request, response);
17.
18.         Cookie ck[]=request.getCookies();
19.         if(ck!=null){
20.          String name=ck[0].getValue();
21.         if(!name.equals("")||name!=null){
22.             out.print("<b>Welcome to Profile</b>");
23.             out.print("<br>Welcome, "+name);
24.         }
25.         }else{
26.             out.print("Please login first");
27.             request.getRequestDispatcher("login.html").include(request, response);
28.         }
29.         out.close();
30.     }
31. }
```

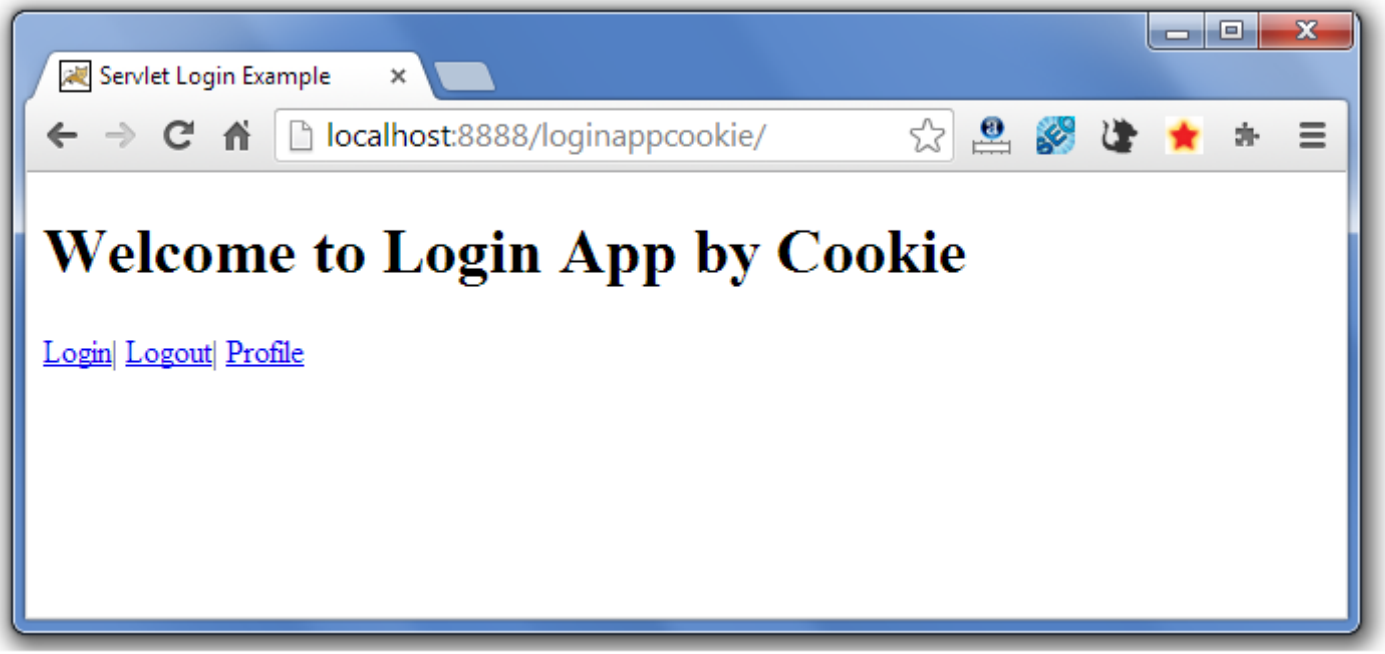
File: web.xml

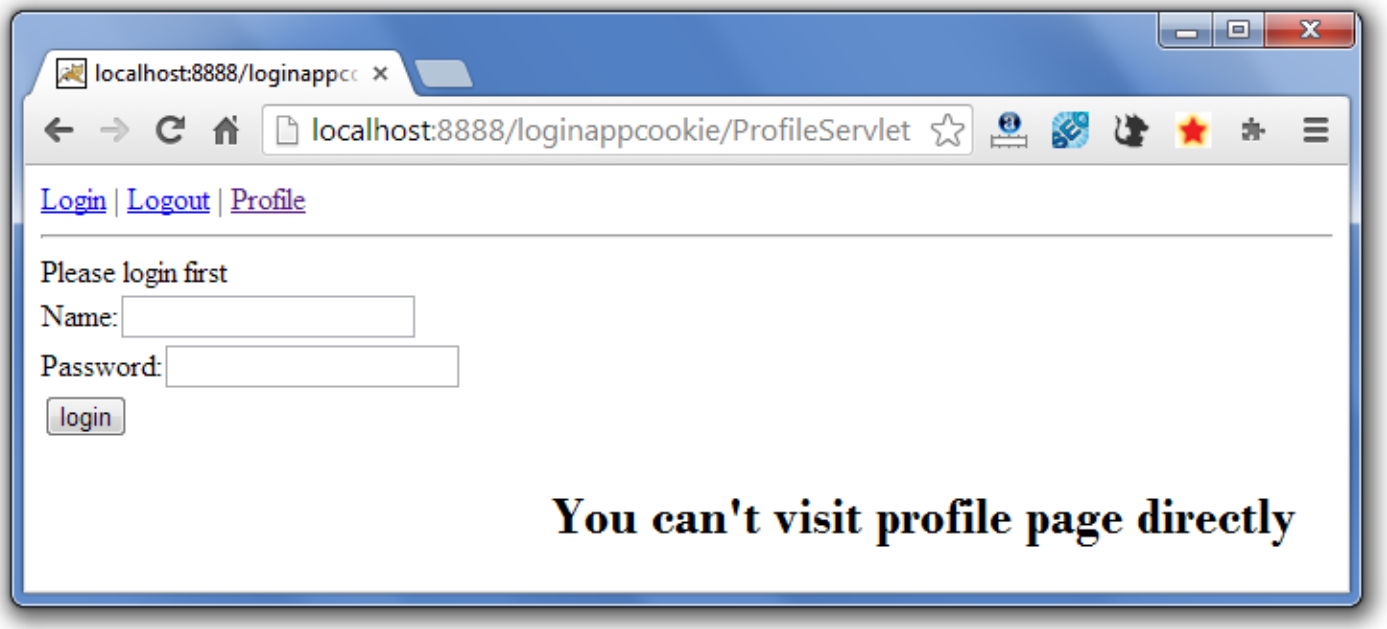
```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

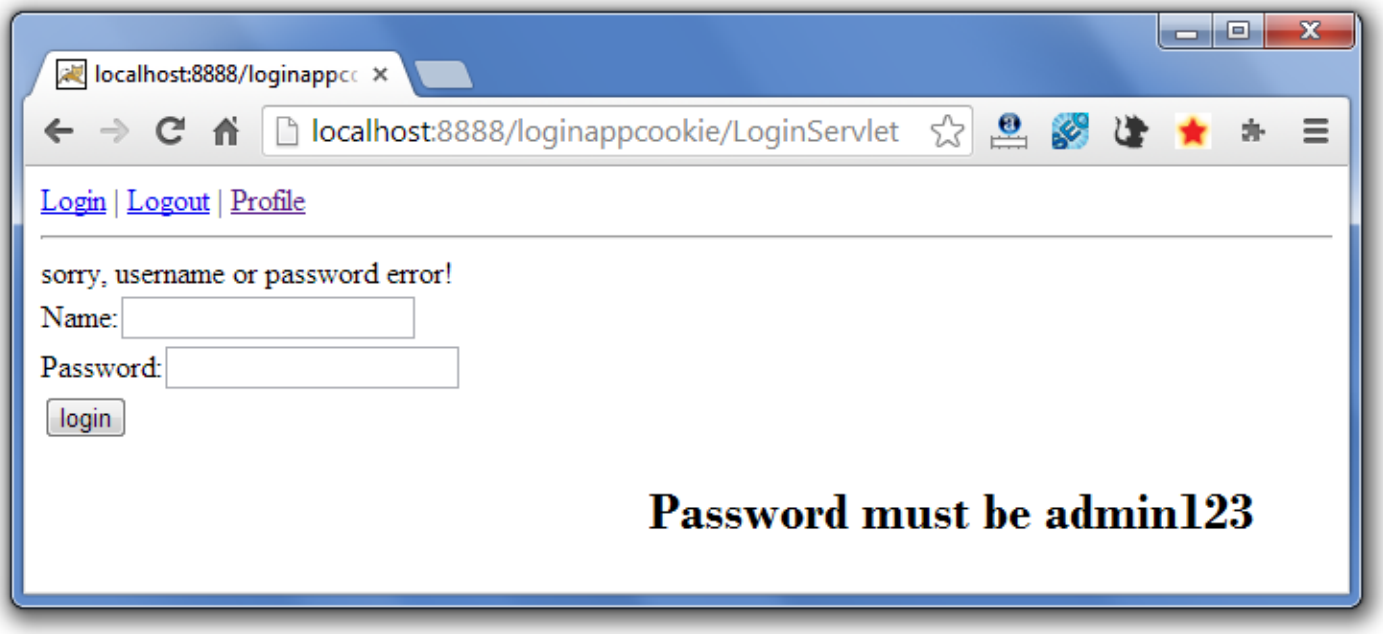
```
3.
   xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml
   /ns/javaee
4. http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
5.
6.   <servlet>
7.     <description></description>
8.     <display-name>LoginServlet</display-name>
9.     <servlet-name>LoginServlet</servlet-name>
10.    <servlet-class>com.javatpoint.LoginServlet</servlet-class>
11.  </servlet>
12.  <servlet-mapping>
13.    <servlet-name>LoginServlet</servlet-name>
14.    <url-pattern>/LoginServlet</url-pattern>
15.  </servlet-mapping>
16.  <servlet>
17.    <description></description>
18.    <display-name>ProfileServlet</display-name>
19.    <servlet-name>ProfileServlet</servlet-name>
20.    <servlet-class>com.javatpoint.ProfileServlet</servlet-class>
21.  </servlet>
22.  <servlet-mapping>
23.    <servlet-name>ProfileServlet</servlet-name>
24.    <url-pattern>/ProfileServlet</url-pattern>
25.  </servlet-mapping>
26.  <servlet>
27.    <description></description>
28.    <display-name>LogoutServlet</display-name>
29.    <servlet-name>LogoutServlet</servlet-name>
30.    <servlet-class>com.javatpoint.LogoutServlet</servlet-class>
31.  </servlet>
32.  <servlet-mapping>
33.    <servlet-name>LogoutServlet</servlet-name>
```

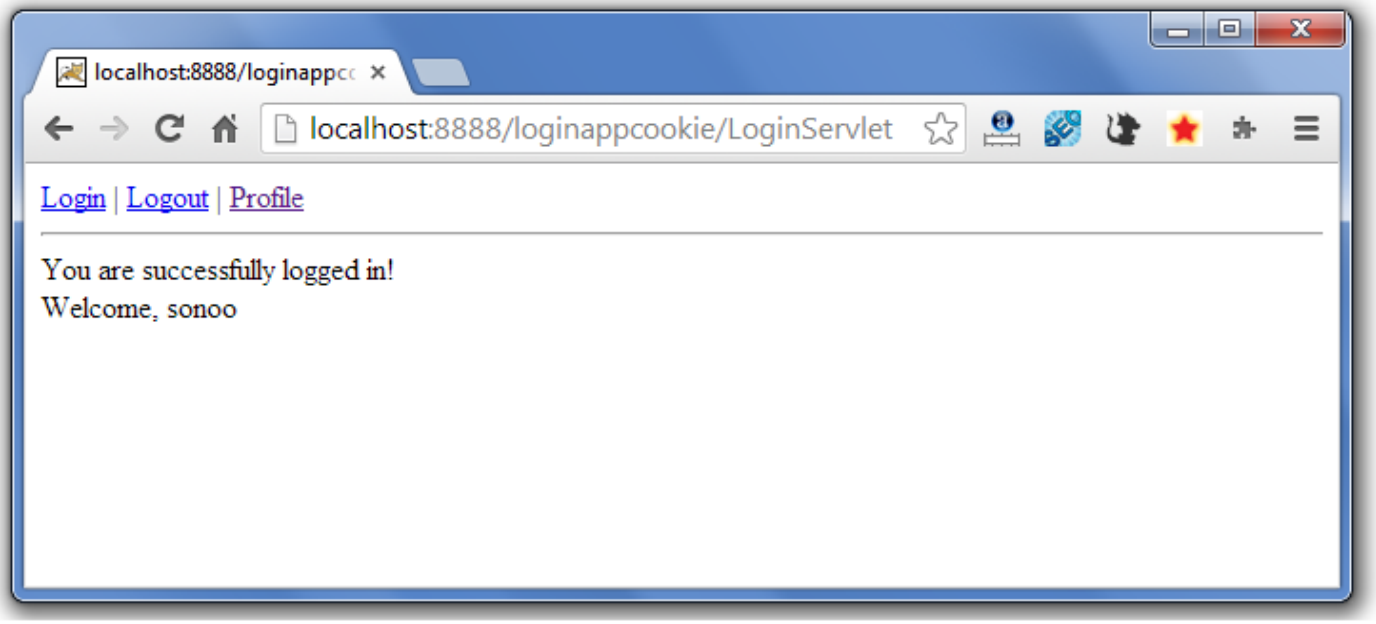
```
34.    <url-pattern>/LogoutServlet</url-pattern>
35.  </servlet-mapping>
36. </web-app>
```

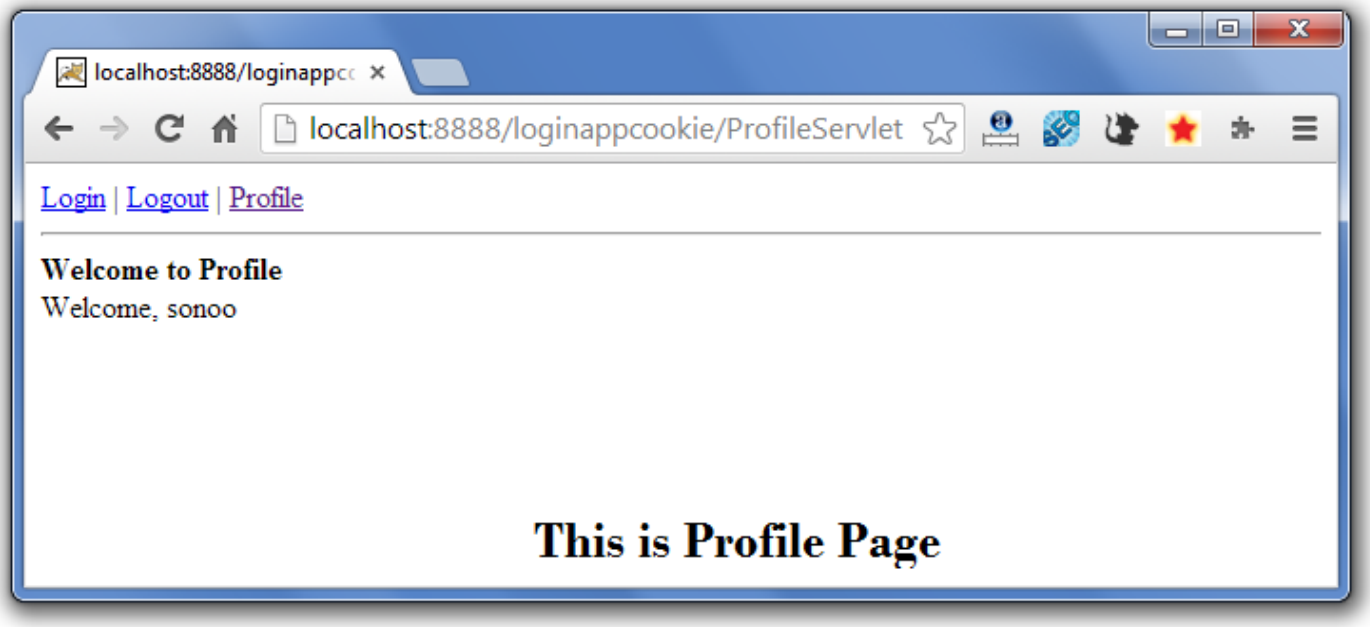
Output

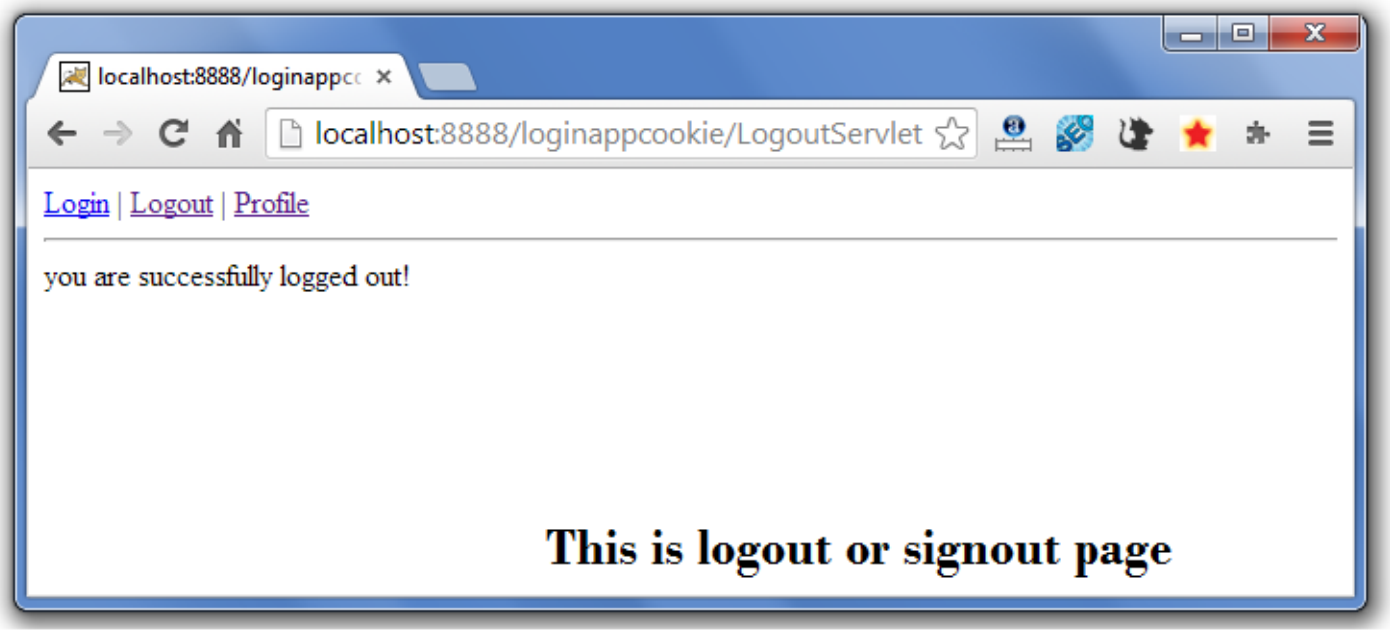





localhost:8888/loginappcc
localhost:8888/loginappcookie/LoginServlet
Login | Logout | Profile
sorry, username or password error!
Name:
Password:
login
Password must be admin123


localhost:8888/loginappcc
localhost:8888/loginappcookie/LoginServlet
Login | Logout | Profile
You are successfully logged in!
Welcome, sonoo





If again you click on the profile link, you need to login first.

Hidden Form Field

1. Hidden Form Field
2. Example of Hidden Form Field

In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Let's see the code to store value in hidden field.

```
1. <input type="hidden" name="uname" value="Vimal Jaiswal">
```

Here, uname is the hidden field name and VimalJaiswal is the hidden field value.

Real application of hidden form field

It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

Advantage of Hidden Form Field

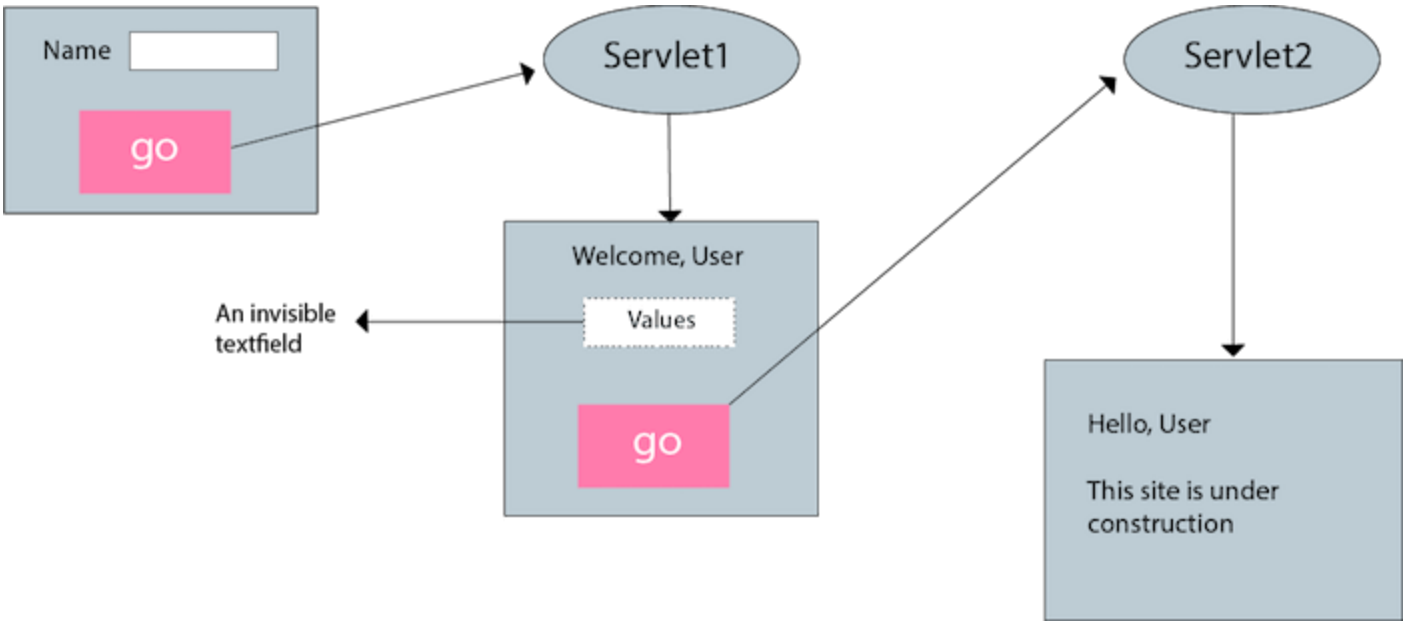
1. It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

Example of using Hidden Form Field

In this example, we are storing the name of the user in a hidden textfield and getting that value from another servlet.



index.html

```
1. <form action="servlet1">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>
```

FirstServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4. 
5. public class FirstServlet extends HttpServlet {
6. public void doGet(HttpServletRequest request, HttpServletResponse response){
7.     try{
8. 
9.     response.setContentType("text/html");
10.    PrintWriter out = response.getWriter();
11.
12.    String n=request.getParameter("userName");
13.    out.print("Welcome "+n);
14.
15.    //creating form that have invisible textfield
```

```
16.     out.print("<form action='servlet2'>");
17.     out.print("<input type='hidden' name='uname' value='"+n+"'>");
18.     out.print("<input type='submit' value='go'>");
19.     out.print("</form>");
20.     out.close();
21. 
22.         }catch(Exception e){System.out.println(e);}
23.   }
24. 
25. }
```

SecondServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.  public class SecondServlet extends HttpServlet {
5.  public void doGet(HttpServletRequest request, HttpServletResponse response)
6.      try{
7.      response.setContentType("text/html");
8.      PrintWriter out = response.getWriter();
9.  
10.     //Getting the value from the hidden field
11.     String n=request.getParameter("uname");
12.     out.print("Hello "+n);
13. 
14.     out.close();
15.         }catch(Exception e){System.out.println(e);}
16.   }
17. }
```

web.xml

```
1.  <web-app>
2.  
3.  <servlet>
4.  <servlet-name>s1</servlet-name>
```

```
5.  <servlet-class>FirstServlet</servlet-class>
6.  </servlet>
7.  
8.  <servlet-mapping>
9.  <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
12. 
13. <servlet>
14. <servlet-name>s2</servlet-name>
15. <servlet-class>SecondServlet</servlet-class>
16. </servlet>
17. 
18. <servlet-mapping>
19. <servlet-name>s2</servlet-name>
20. <url-pattern>/servlet2</url-pattern>
21. </servlet-mapping>
22. 
23. </web-app>
```

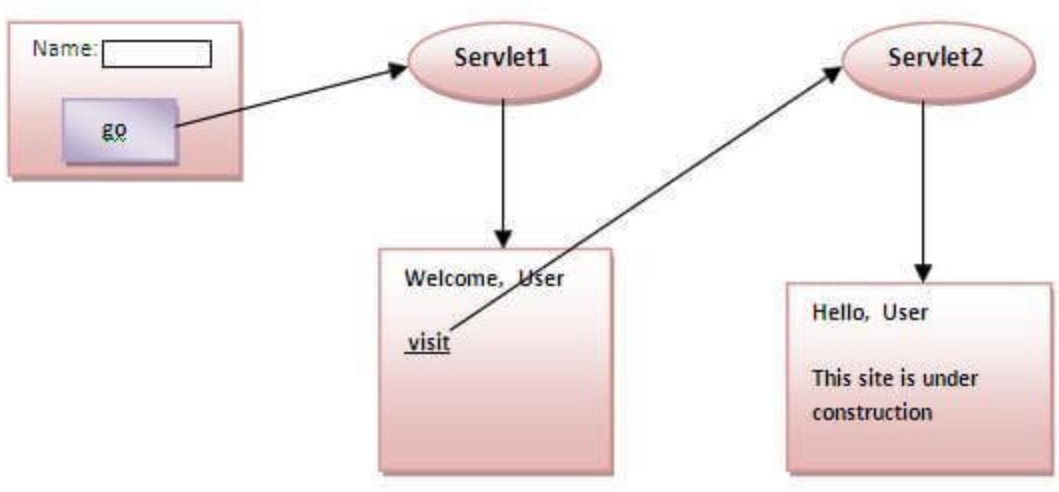
URL Rewriting

1. URL Rewriting
2. Advantage of URL Rewriting
3. Disadvantage of URL Rewriting
4. Example of URL Rewriting

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.



Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

Example of using URL Rewriting

In this example, we are maintaning the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.

index.html

```
1. <form action="servlet1">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>
```

FirstServlet.java

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

```
3. import javax.servlet.http.*;
4. 
5. 
6. public class FirstServlet extends HttpServlet {
7. 
8. public void doGet(HttpServletRequest request, HttpServletResponse response){
9.     try{
10. 
11.     response.setContentType("text/html");
12.     PrintWriter out = response.getWriter();
13.     
14.     String n=request.getParameter("userName");
15.     out.print("Welcome "+n);
16. 
17.     //appending the username in the query string
18.     out.print("<a href='servlet2?uname="+n+"'>visit</a>");
19.     
20.     out.close();
21. 
22.         }catch(Exception e){System.out.println(e);}
23.   }
24. 
25. }
```

SecondServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4. 
5. public class SecondServlet extends HttpServlet {
6. 
7. public void doGet(HttpServletRequest request, HttpServletResponse response)
8.     try{
9. 
10.     response.setContentType("text/html");
```

```
        PrintWriter out = response.getWriter();

        //getting value from the query string
        String n=request.getParameter("uname");
        out.print("Hello "+n);

        out.close();

            }catch(Exception e){System.out.println(e);}
    }

}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
```

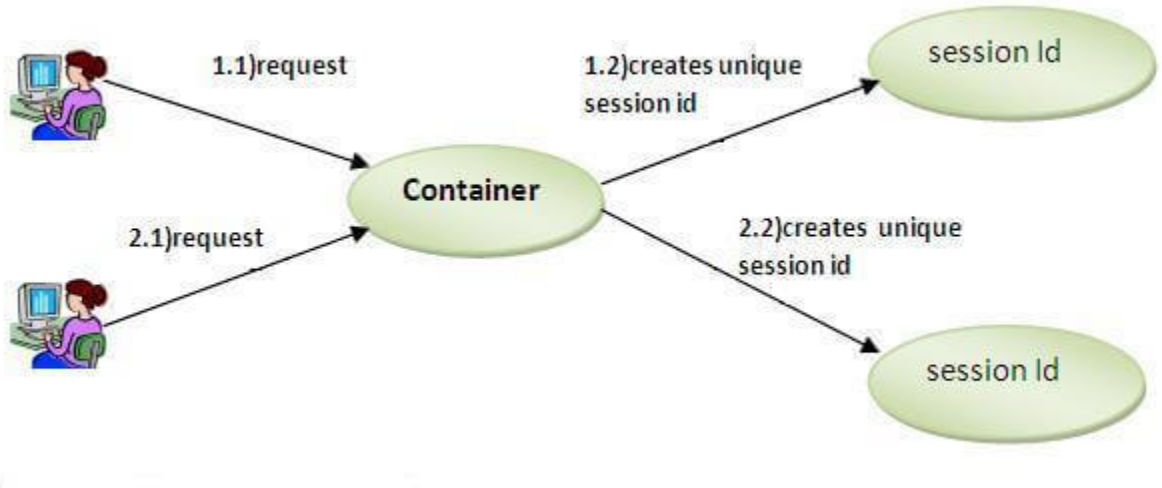
21. </servlet-mapping>
22. _
23. </web-app>

) HttpSession interface

1. HttpSession interface
2. How to get the HttpSession object
3. Commonly used methods of HttpSession interface
4. Example of using HttpSession

In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSessionobject ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **publicHttpSessiongetSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **publicHttpSessiongetSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

Example of using HttpSession

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.

index.html

```
1. <form action="servlet1">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>
```

FirstServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5.
6. public class FirstServlet extends HttpServlet {
```

```
7.
8.  public void doGet(HttpServletRequest request, HttpServletResponse response){
9.          try{
10.
11.         response.setContentType("text/html");
12.         PrintWriter out = response.getWriter();
13.
14.         String n=request.getParameter("userName");
15.         out.print("Welcome "+n);
16.
17.         HttpSession session=request.getSession();
18.         session.setAttribute("uname",n);
19.
20.         out.print("<a href='servlet2'>visit</a>");
21.
22.         out.close();
23.
24.                 }catch(Exception e){System.out.println(e);}
25.    }
26.
27. }
```

SecondServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.
5.  public class SecondServlet extends HttpServlet {
6.
7.  public void doGet(HttpServletRequest request, HttpServletResponse response)
8.          try{
9.
10.         response.setContentType("text/html");
11.         PrintWriter out = response.getWriter();
12.
```

```
13.         HttpSession session=request.getSession(false);
14.         String n=(String)session.getAttribute("uname");
15.         out.print("Hello "+n);
16.
17.         out.close();
18.
19.                 }catch(Exception e){System.out.println(e);}
20.     }
21.
22.
23. }
```

web.xml

```
1.  <web-app>
2.
3.  <servlet>
4.  <servlet-name>s1</servlet-name>
5.  <servlet-class>FirstServlet</servlet-class>
6.  </servlet>
7.
8.  <servlet-mapping>
9.  <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
12.
13. <servlet>
14. <servlet-name>s2</servlet-name>
15. <servlet-class>SecondServlet</servlet-class>
16. </servlet>
17.
18. <servlet-mapping>
19. <servlet-name>s2</servlet-name>
20. <url-pattern>/servlet2</url-pattern>
21. </servlet-mapping>
22.
```

23. </web-app>

Servlet HttpSession Login and Logout Example

We can bind the objects on HttpSession instance and get the objects by using setAttribute and getAttribute methods.

In the previous page, we have learnt about what isHttpSession, How to store and get data from session object etc.

Here, we are going to create a real world login and logout application without using database code. We are assuming that password is admin123.

Visit here for login and logout application using cookies only servlet login and logout example using cookies

In this example, we are creating 3 links: login, logout and profile. User can't go to profile page until he/she is logged in. If user is logged out, he need to login again to visit profile.

In this application, we have created following files.

1. index.html
2. link.html
3. login.html
4. LoginServlet.java
5. LogoutServlet.java
6. ProfileServlet.java
7. web.xml

File: index.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset="ISO-8859-1">
```

```
5. <title>Servlet Login Example</title>
6. </head>
7. <body>
8. 
9. <h1>Login App using HttpSession</h1>
10. <a href="login.html">Login</a>|
11. <a href="LogoutServlet">Logout</a>|
12. <a href="ProfileServlet">Profile</a>
13. 
14. </body>
15. </html>
```

File: link.html

```
1. <a href="login.html">Login</a> |
2. <a href="LogoutServlet">Logout</a> |
3. <a href="ProfileServlet">Profile</a>
4. <hr>
```

File: login.html

```
1. <form action="LoginServlet" method="post">
2. Name:<input type="text" name="name"><br>
3. Password:<input type="password" name="password"><br>
4. <input type="submit" value="login">
5. </form>
```

File: LoginServlet.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3. 
4. import javax.servlet.ServletException;
5. import javax.servlet.http.HttpServlet;
6. import javax.servlet.http.HttpServletRequest;
7. import javax.servlet.http.HttpServletResponse;
```

```
import javax.servlet.http.HttpSession;
public class LoginServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    request.getRequestDispatcher("link.html").include(request, response);

    String name=request.getParameter("name");
    String password=request.getParameter("password");

    if(password.equals("admin123")){
    out.print("Welcome, "+name);
    HttpSession session=request.getSession();
    session.setAttribute("name",name);
    }
    else{
        out.print("Sorry, username or password error!");
        request.getRequestDispatcher("login.html").include(request, response);
    }
    out.close();
  }
}
```

File: LogoutServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class LogoutServlet extends HttpServlet {
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        request.getRequestDispatcher("link.html").include(request, response);

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

        out.print("You are successfully logged out!");

        out.close();
    }
}
```

File: ProfileServlet.java

```
1.  import java.io.IOException;
2.  import java.io.PrintWriter;
3.  import javax.servlet.ServletException;
4.  import javax.servlet.http.HttpServlet;
5.  import javax.servlet.http.HttpServletRequest;
6.  import javax.servlet.http.HttpServletResponse;
7.  import javax.servlet.http.HttpSession;
8.  public class ProfileServlet extends HttpServlet {
9.    protected void doGet(HttpServletRequest request, HttpServletResponse response)
10.                 throws ServletException, IOException {
11.     response.setContentType("text/html");
12.     PrintWriter out=response.getWriter();
13.     request.getRequestDispatcher("link.html").include(request, response);
14.
15.     HttpSession session=request.getSession(false);
16.     if(session!=null){
17.     String name=(String)session.getAttribute("name");
```

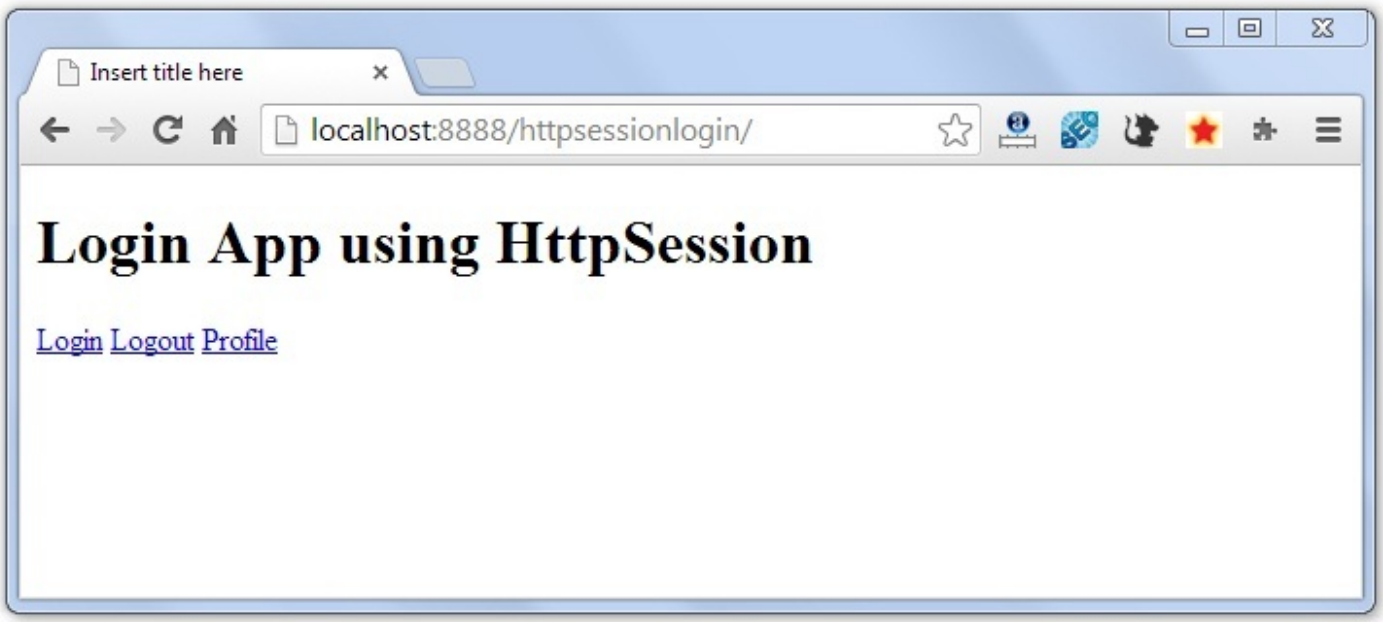
```
18.
19.     out.print("Hello, "+name+" Welcome to Profile");
20.     }
21.     else{
22.         out.print("Please login first");
23.         request.getRequestDispatcher("login.html").include(request, response);
24.     }
25.     out.close();
26.   }
27. }
```

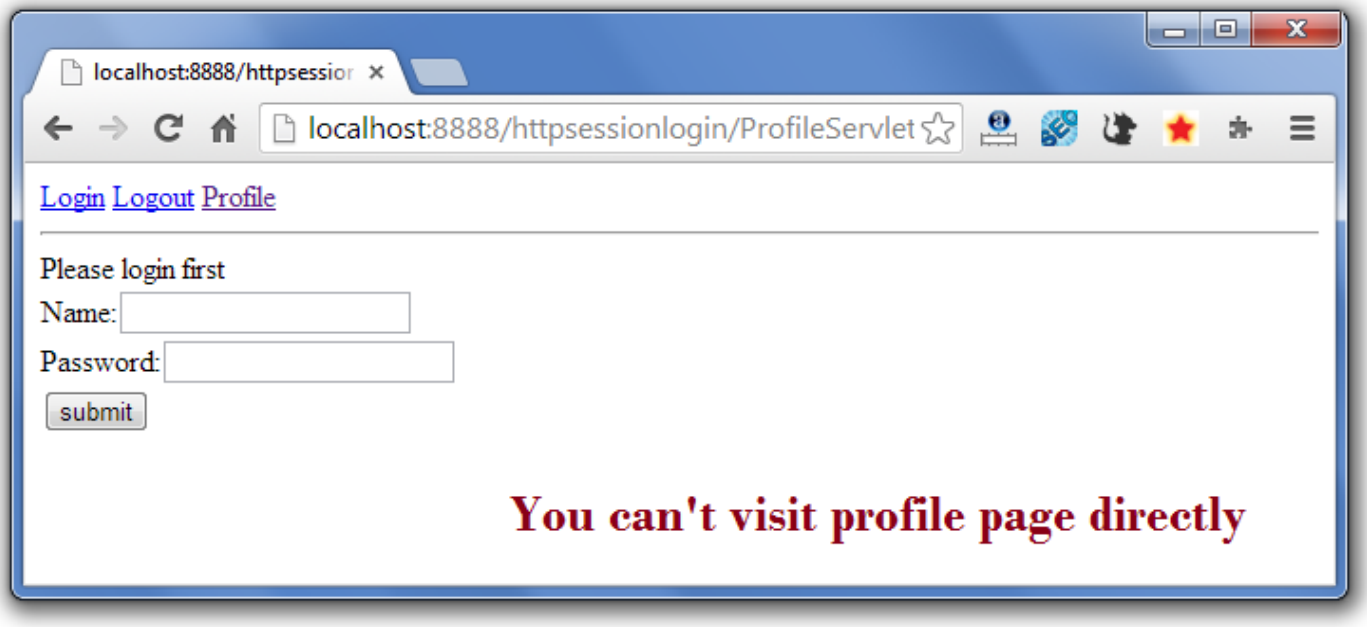
File: web.xml

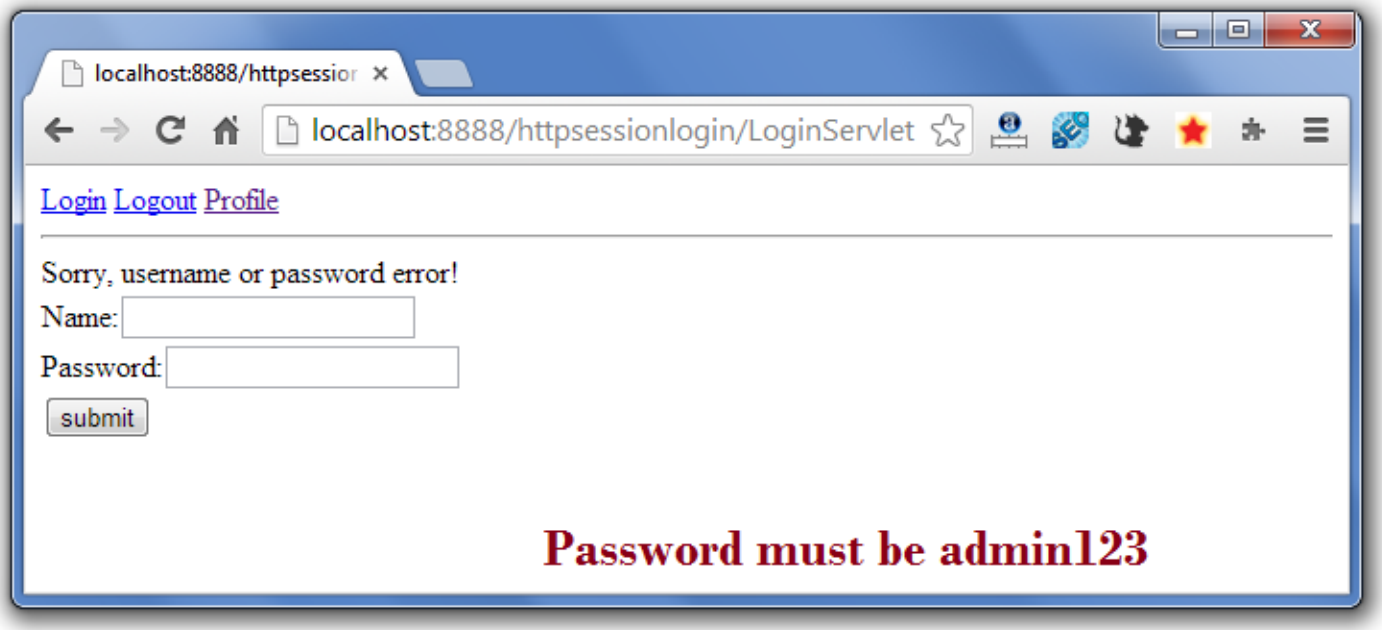
```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.
   xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
4. http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
5.
6.   <servlet>
7.     <description></description>
8.     <display-name>LoginServlet</display-name>
9.     <servlet-name>LoginServlet</servlet-name>
10.    <servlet-class>LoginServlet</servlet-class>
11.  </servlet>
12.  <servlet-mapping>
13.    <servlet-name>LoginServlet</servlet-name>
14.    <url-pattern>/LoginServlet</url-pattern>
15.  </servlet-mapping>
16.  <servlet>
17.    <description></description>
18.    <display-name>ProfileServlet</display-name>
19.    <servlet-name>ProfileServlet</servlet-name>
```

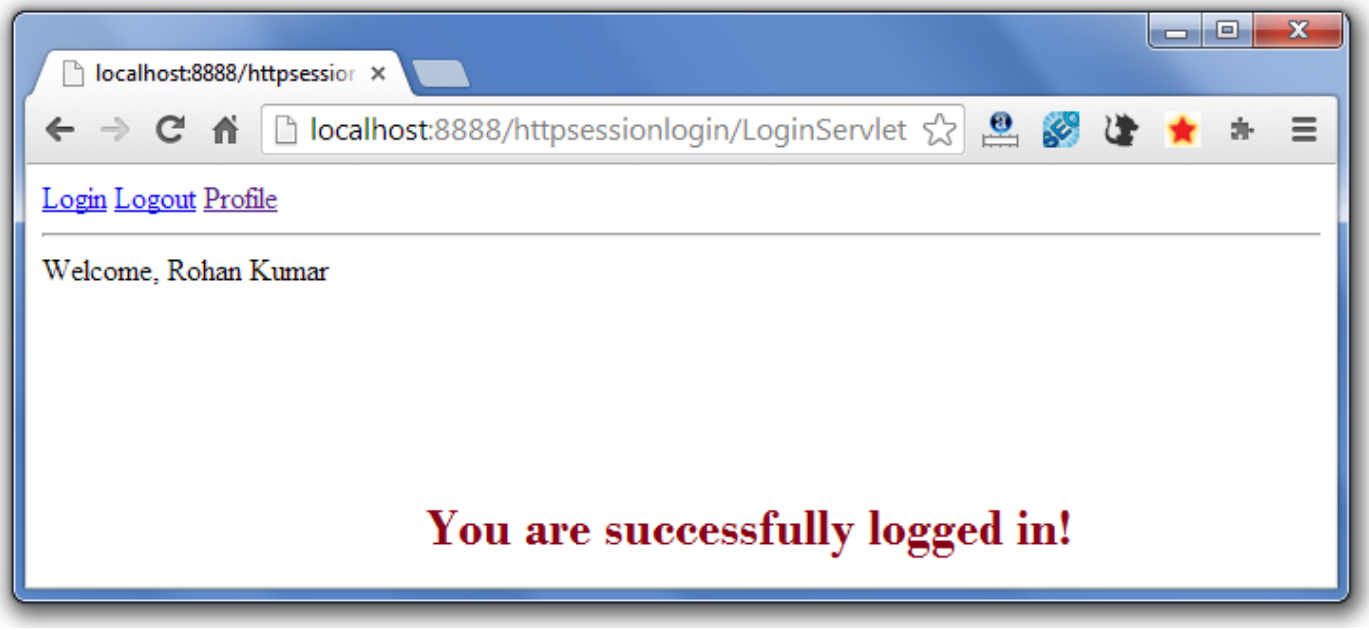
```
20.   <servlet-class>ProfileServlet</servlet-class>
21.  </servlet>
22.  <servlet-mapping>
23.   <servlet-name>ProfileServlet</servlet-name>
24.   <url-pattern>/ProfileServlet</url-pattern>
25.  </servlet-mapping>
26.  <servlet>
27.   <description></description>
28.   <display-name>LogoutServlet</display-name>
29.   <servlet-name>LogoutServlet</servlet-name>
30.   <servlet-class>LogoutServlet</servlet-class>
31.  </servlet>
32.  <servlet-mapping>
33.   <servlet-name>LogoutServlet</servlet-name>
34.   <url-pattern>/LogoutServlet</url-pattern>
35.  </servlet-mapping>
36. </web-app>
```

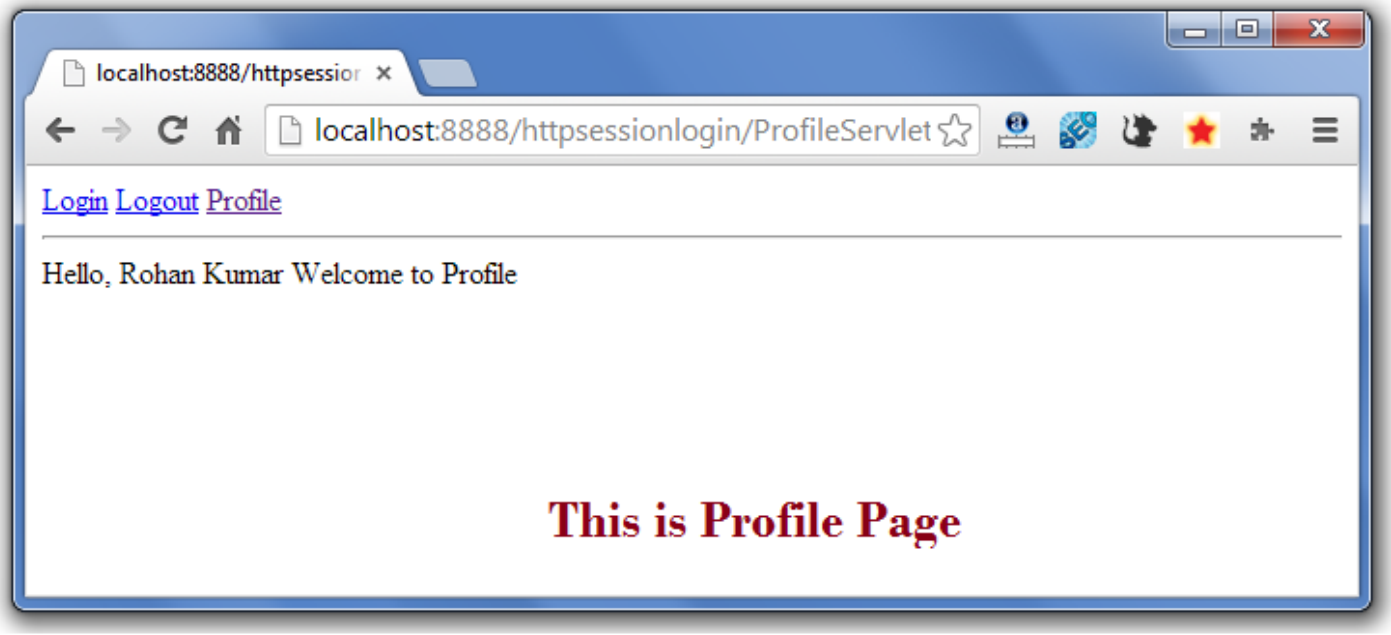
Output

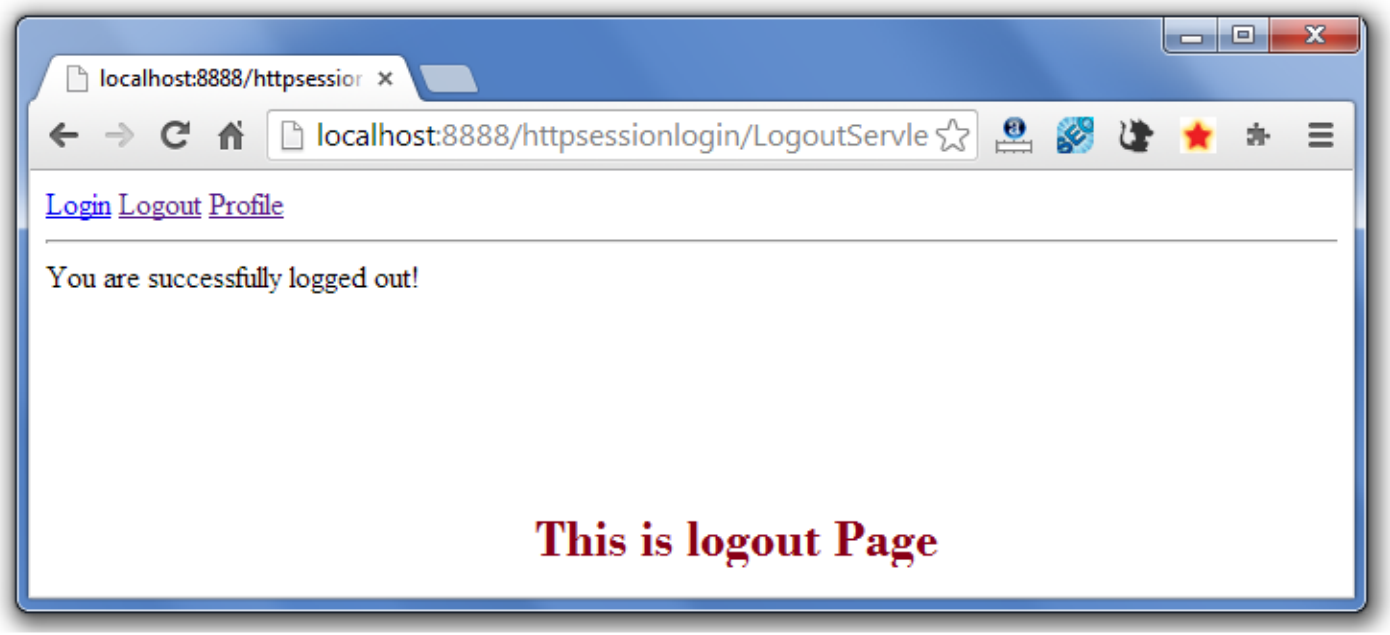



localhost:8888/httpsessior
localhost:8888/httpsessionlogin/ProfileServlet
Login Logout Profile
Please login first
Name:
Password:
submit
You can't visit profile page directly


localhost:8888/httpsessior
localhost:8888/httpsessionlogin/LoginServlet
Login Logout Profile
Sorry, username or password error!
Name:
Password:
submit
Password must be admin123


localhost:8888/httpsessior
localhost:8888/httpsessionlogin/LoginServlet
Login Logout Profile
Welcome, Rohan Kumar
You are successfully logged in!


localhost:8888/httpsessior
localhost:8888/httpsessionlogin/ProfileServlet
Login Logout Profile
Hello, Rohan Kumar Welcome to Profile
This is Profile Page



If again you click on the profile link, you need to login first.

HTTP Requests

The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests.

The HTTP client sends the request to the server in the form of request message which includes following information:

- The Request-line
- The analysis of source IP address, proxy and port
- The analysis of destination IP address, protocol, port and host
- The Requested URI (Uniform Resource Identifier)
- The Request method and Content
- The User-Agent header
- The Connection control header
- The Cache control header

The HTTP request method indicates the method to be performed on the resource identified by the **Requested URI (Uniform Resource Identifier)**. This method is case-sensitive and should be used in uppercase.

The HTTP request methods are:

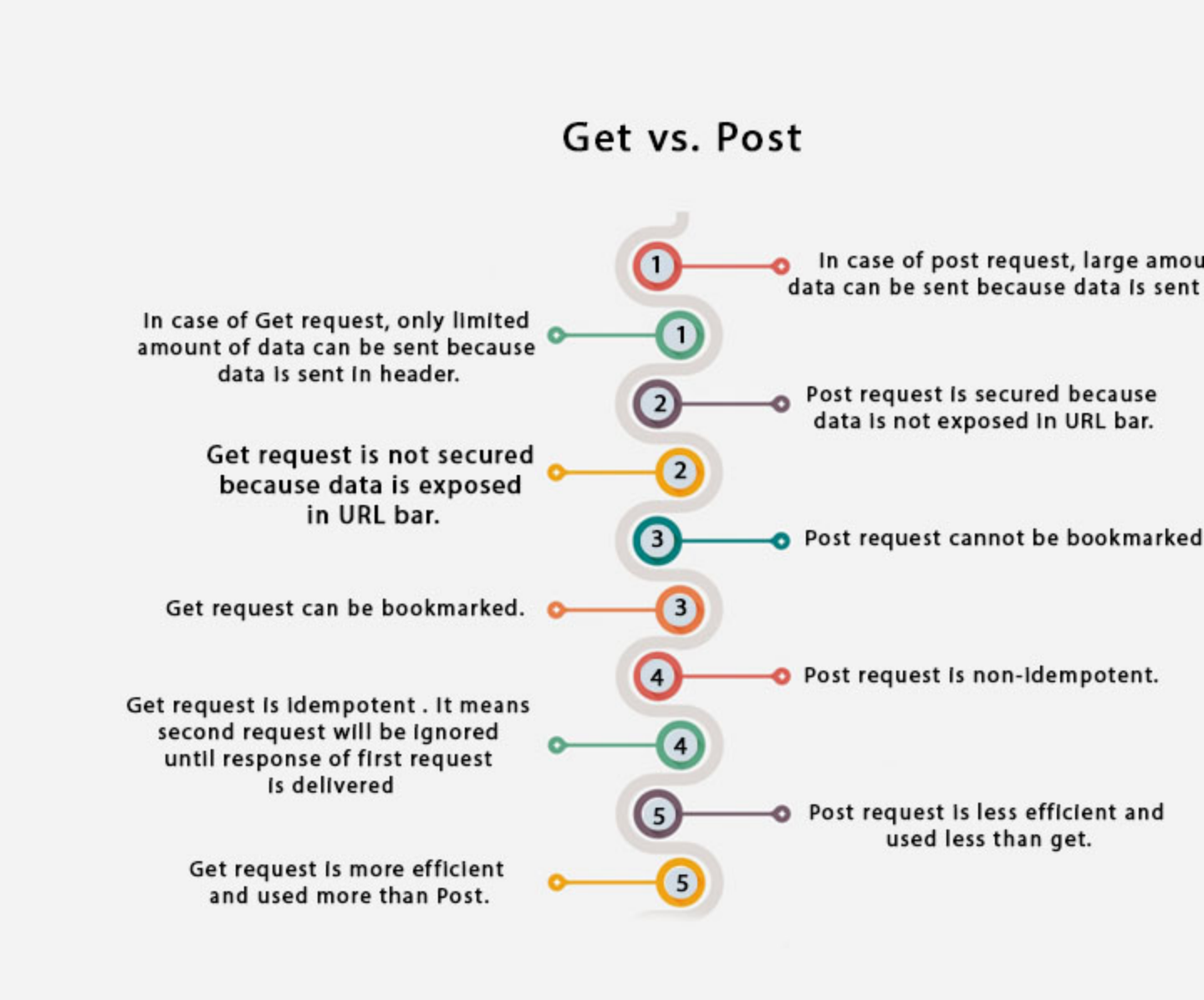
HTTP Request	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET request with extra info with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but w

	body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respon

Get vs. Post

There are many differences between the Get and Post request. Let's see these differences:

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amou data** can be sent because data is sent in b
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data i exposed in URL bar.
3) Get request **can be bookmarked.**	Post request **cannot be bookmarked.**
4) Get request is **idempotent** . It means second request will be ignored until response of first request is delivered	Post request is **non-idempotent.**
5) Get request is **more efficient** and used more than Post.	Post request is **less efficient** and used than get.



GET and POST

Two common methods for the request-response between a server and client are:

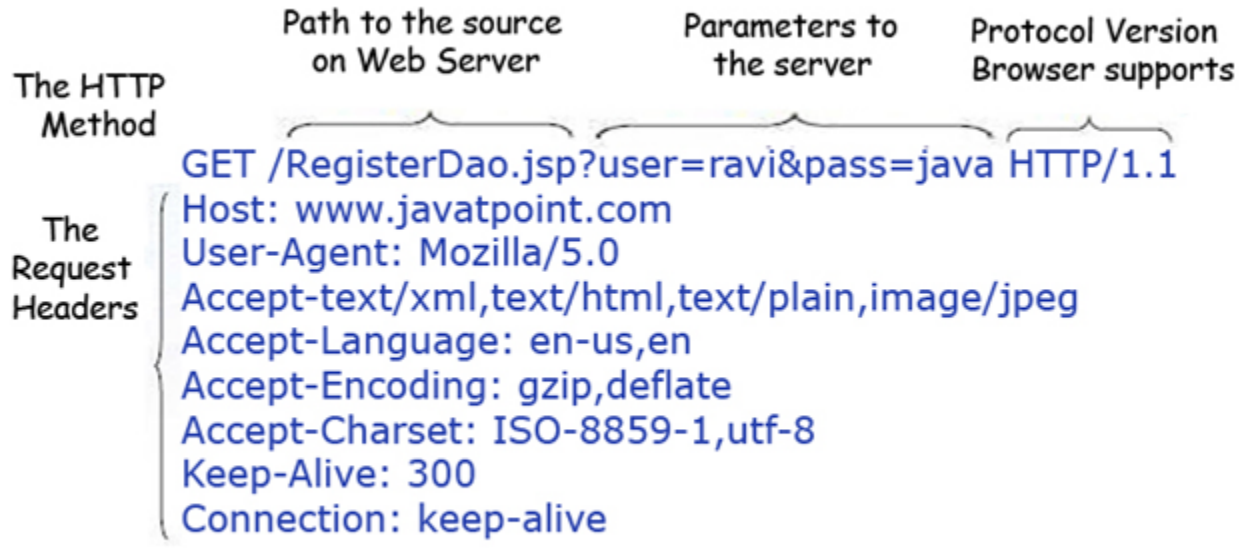
- **GET**- It requests the data from a specified resource
- **POST**- It submits the processed data to a specified resource

Anatomy of Get Request

The query string (name/value pairs) is sent inside the URL of a GET request:

1. GET/RegisterDao.jsp?name1=value1&name2=value2

   As we know that data is sent in request header in case of get request. It is the default request type. Let's see what information is sent to the server.



Some other features of GET requests are:

- It remains in the browser history
- It can be bookmarked
- It can be cached
- It have length restrictions
- It should never be used when dealing with sensitive data
- It should only be used for retrieving the data

Anatomy of Post Request

The query string (name/value pairs) is sent in HTTP message body for a POST request:

1. POST/RegisterDao.jsp HTTP/1.1

2. Host: www. javatpoint.com
3. name1=value1&name2=value2

As we know, in case of post request original data is sent in message body. Let's see how information is passed to the server in case of post request.

Some other features of POST requests are:

- This requests cannot be bookmarked
- This requests have no restrictions on length of data
- This requests are never cached
- This requests do not retain in the browser history
- **Reading Servlet Parameters**
-

- The **ServletRequest** interface includes methods that allow you to read the names and values of parameters that are included in a client request. We will develop a servlet that illustrates their use. The example contains two files. A web page is defined in **PostParameters.html**, and a servlet is defined in **PostParametersServlet.java**.
- The HTML source code for **PostParameters.html** is shown in the following listing. It defines a table that contains two labels and two text fields. One of the labels is Employee and the other is Phone. There is also a submit button. Notice that the action parameter of the form tag specifies a URL. The URL identifies the servlet to process the HTTP POST request.

```
<html>

<body>

<center>

<form name="Form1" method="post"

action="http://localhost:8080/examples/servlets/

servlet/PostParametersServlet">

<table>

<tr>

<td><B>Employee</td>

<td><input type=textbox name="e" size="25" value=""></td>
</tr>

<tr>

<td><B>Phone</td>

<td><input type=textbox name="p" size="25" value=""></td>
</tr>
```

```
</table>

<input type=submit value="Submit"> </body>

</html>
```

The source code for **PostParametersServlet.java** is shown in the following listing. The **service( )** method is overridden to process client requests. The **getParameterNames( )** method returns an enumeration of the parameter names. These are processed in a loop. You can see that the parameter name and value are output to the client. The parameter value is obtained via the **getParameter( )** method.

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

public class PostParametersServlet extends GenericServlet {

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException {

// Get print writer.

PrintWriter pw = response.getWriter();

 Get enumeration of parameter names. Enumeration e = request.getParameterNames();

 Display parameter names and values. while(e.hasMoreElements()) {

String pname = (String)e.nextElement(); pw.print(pname + " = ");
```

```
String pvalue = request.getParameter(pname);
pw.println(pvalue);
}

pw.close();

}
```

- Compile the servlet. Next, copy it to the appropriate directory, and update the **web.xml** file, as previously described. Then, perform these steps to test this example:
  - Start Tomcat (if it is not already running).
  - Display the web page in a browser.
  - Enter an employee name and phone number in the text fields.
  - Submit the web page.
- After following these steps, the browser will display a response that is dynamically generated by the servlet.

READING INITALIZATION PARAMETER

ServletConfig Interface

1. ServletConfig Interface
2. Methods of ServletConfig interface
3. How to get the object of ServletConfig
4. Syntax to provide the initialization parameter for a servlet
5. Example of ServletConfig to get initialization parameter
6. Example of ServletConfig to get all the initialization parameter

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

Advantage of ServletConfig

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

Methods of ServletConfig interface

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

How to get the object of ServletConfig

1. **getServletConfig() method** of Servlet interface returns the object of ServletConfig.

Syntax of getServletConfig() method

```
1. public ServletConfig getServletConfig();
```

Example of getServletConfig() method

```
1. ServletConfig config=getServletConfig();
2. //Now we can call the methods of ServletConfig interface
```

Syntax to provide the initialization parameter for a servlet

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

```
1. <web-app>
2.   <servlet>
3.     .....
```

```
4.
5.      <init-param>
6.        <param-name>parametername</param-name>
7.        <param-value>parametervalue</param-value>
8.      </init-param>
9.      .....
10.   </servlet>
11. </web-app>
```

Example of ServletConfig to get initialization parameter

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

DemoServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.
5.  public class DemoServlet extends HttpServlet {
6.  public void doGet(HttpServletRequest request, HttpServletResponse response)
7.     throws ServletException, IOException {
8.
9.     response.setContentType("text/html");
10.    PrintWriter out = response.getWriter();
11.
12.    ServletConfig config=getServletConfig();
13.    String driver=config.getInitParameter("driver");
14.    out.print("Driver is: "+driver);
15.
16.    out.close();
17.    }
18.
19. }
```

web.xml

```
1. <web-app>
2. 
3. <servlet>
4. <servlet-name>DemoServlet</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6. 
7. <init-param>
8. <param-name>driver</param-name>
9. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
10. </init-param>
11. 
12. </servlet>
13. 
14. <servlet-mapping>
15. <servlet-name>DemoServlet</servlet-name>
16. <url-pattern>/servlet1</url-pattern>
17. </servlet-mapping>
18. 
19. </web-app>
```

download this example (developed in Myeclipse IDE)
download this example(developed in Eclipse IDE)
download this example(developed in Netbeans IDE)

Example of ServletConfig to get all the initialization parameters

In this example, we are getting all the initialization parameter from the web.xml file and printing this information in the servlet.

HTML Tutorial

DemoServlet.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3. import java.util.Enumeration;
4. 
5. import javax.servlet.ServletConfig;
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10. 
11. 
12. public class DemoServlet extends HttpServlet {
13. public void doGet(HttpServletRequest request, HttpServletResponse response)
14.     throws ServletException, IOException {
15. 
16.   response.setContentType("text/html");
17.   PrintWriter out = response.getWriter();
18.   
19.   ServletConfig config=getServletConfig();
20.   Enumeration<String> e=config.getInitParameterNames();
21.   
22.   String str="";
23.   while(e.hasMoreElements()){
24.   str=e.nextElement();
25.   out.print("<br>Name: "+str);
26.   out.print(" value: "+config.getInitParameter(str));
27.   }
28.   
29.   out.close();
30. }
31. 
32. }
```

web.xml

```
1. <web-app>
2. _
3. <servlet>
4. <servlet-name>DemoServlet</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6. _
7. <init-param>
8. <param-name>username</param-name>
9. <param-value>system</param-value>
10. </init-param>
11. _
12. <init-param>
13. <param-name>password</param-name>
14. <param-value>oracle</param-value>
15. </init-param>
16. _
17. </servlet>
18. _
19. <servlet-mapping>
20. <servlet-name>DemoServlet</servlet-name>
21. <url-pattern>/servlet1</url-pattern>
22. </servlet-mapping>
23. _
24. </web-app>
```

download this example (developed in Myeclipse IDE)
download this example(developed in Eclipse IDE)
download this example(developed in Netbeans IDE)

CREATE SIMPLE SERVLET

Steps to create a servlet example

1. Steps to create the servlet using Tomcat server

1. Create a directory structure

2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the application

There are given 6 steps to create a **servlet example**. These steps are required for all the servers.

The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

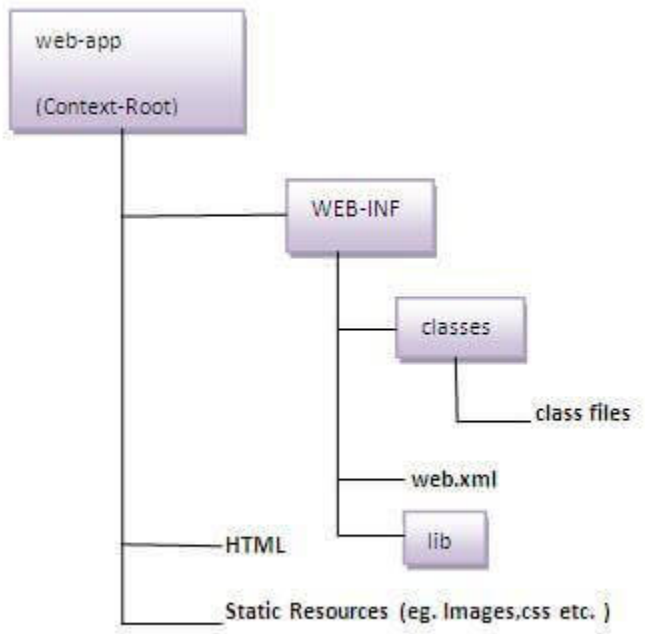
download this example of servlet
download example of servlet by extending GenericServlet
download example of servlet by implementing Servlet interface

1)Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.

Triggers in SQL (Hindi)



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

2)Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are i the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the request.

DemoServlet.java

```
1. import javax.servlet.http.*;
2. import javax.servlet.*;
3. import java.io.*;
4. public class DemoServlet extends HttpServlet{
5. public void doGet(HttpServletRequest req,HttpServletResponse res)
6. throws ServletException,IOException
7. {
8. res.setContentType("text/html");//setting the content type
9. PrintWriter pw=res.getWriter();//get the stream to write the data
10. 
11. //writing html in the stream
12. pw.println("<html><body>");
13. pw.println("Welcome to servlet");
14. pw.println("</body></html>");
15. 
16. pw.close();//closing the stream
17. }}
```

3)Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

4)Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

```
1. <web-app>
2. _
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6. </servlet>
7. _
8. <servlet-mapping>
9. <servlet-name>sonoojaiswal</servlet-name>
10. <url-pattern>/welcome</url-pattern>
11. </servlet-mapping>
12. _
13. </web-app>
```

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

5)Start the Server and deploy the project

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

One Time Configuration for Apache Tomcat Server

You need to perform 2 tasks:

1. set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).
2. Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

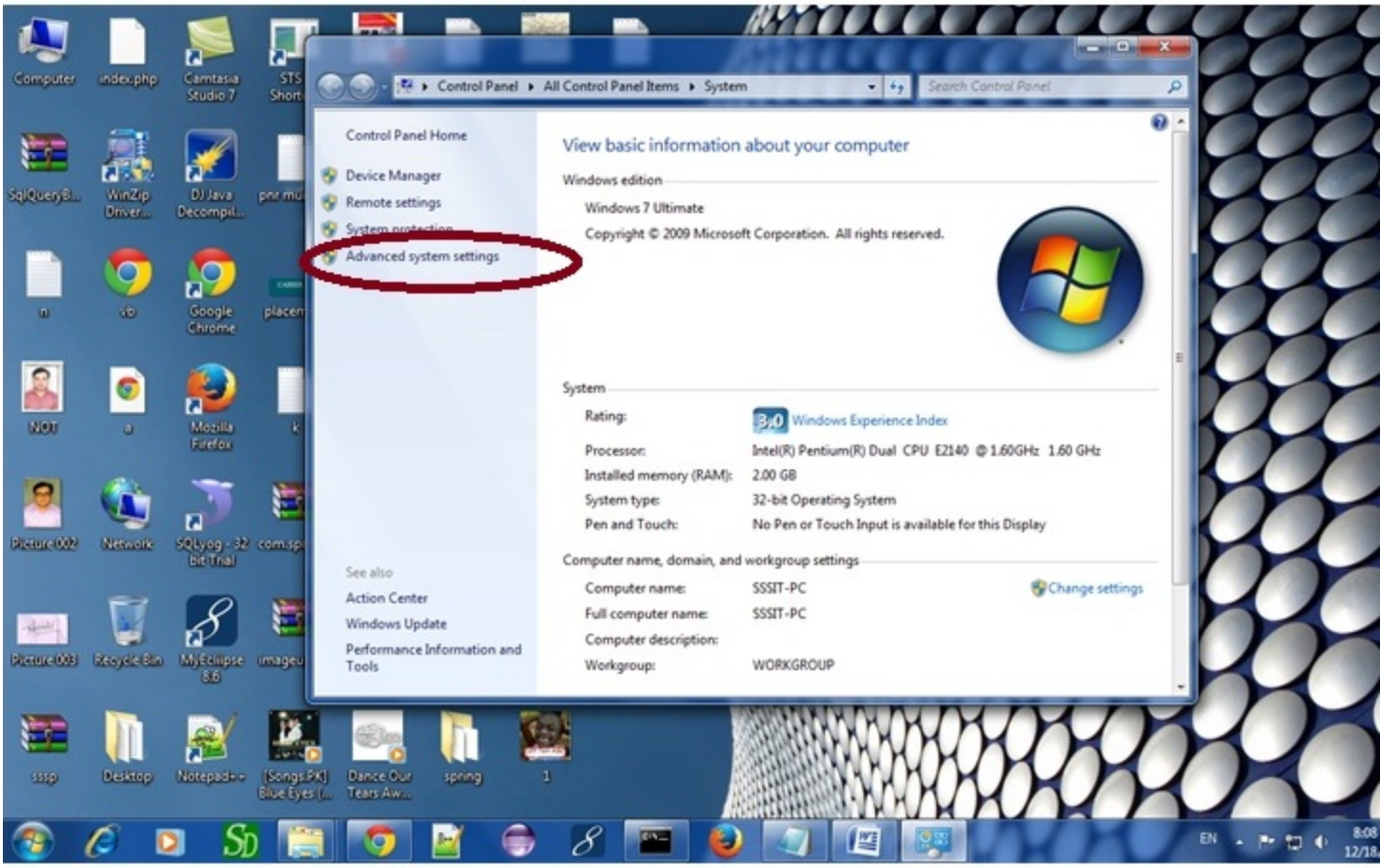
1) How to set JAVA_HOME in environment variable?

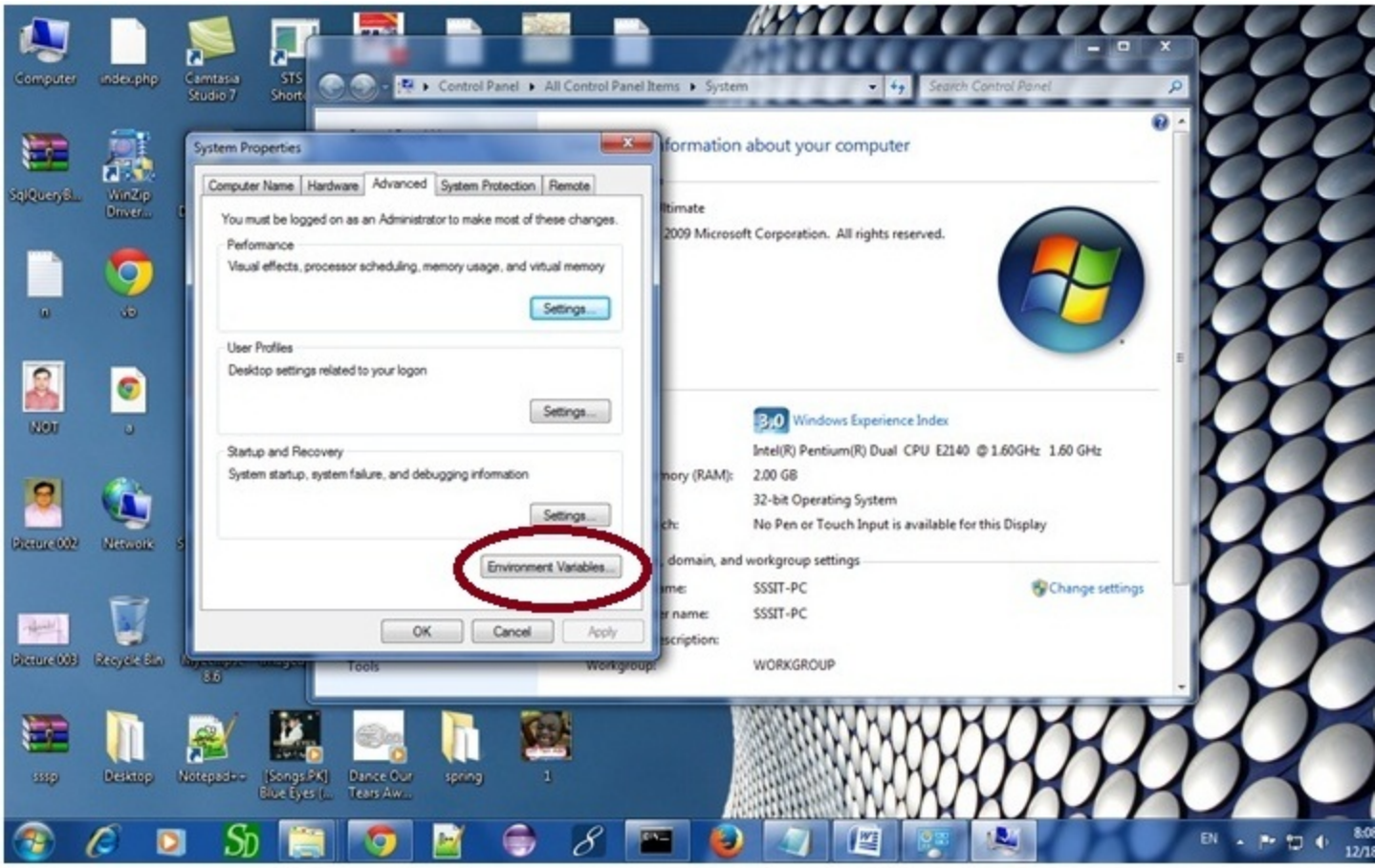
To start Apache Tomcat server JAVA_HOME and JRE_HOME must be set in Environment variables.

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 -> ok -> ok.

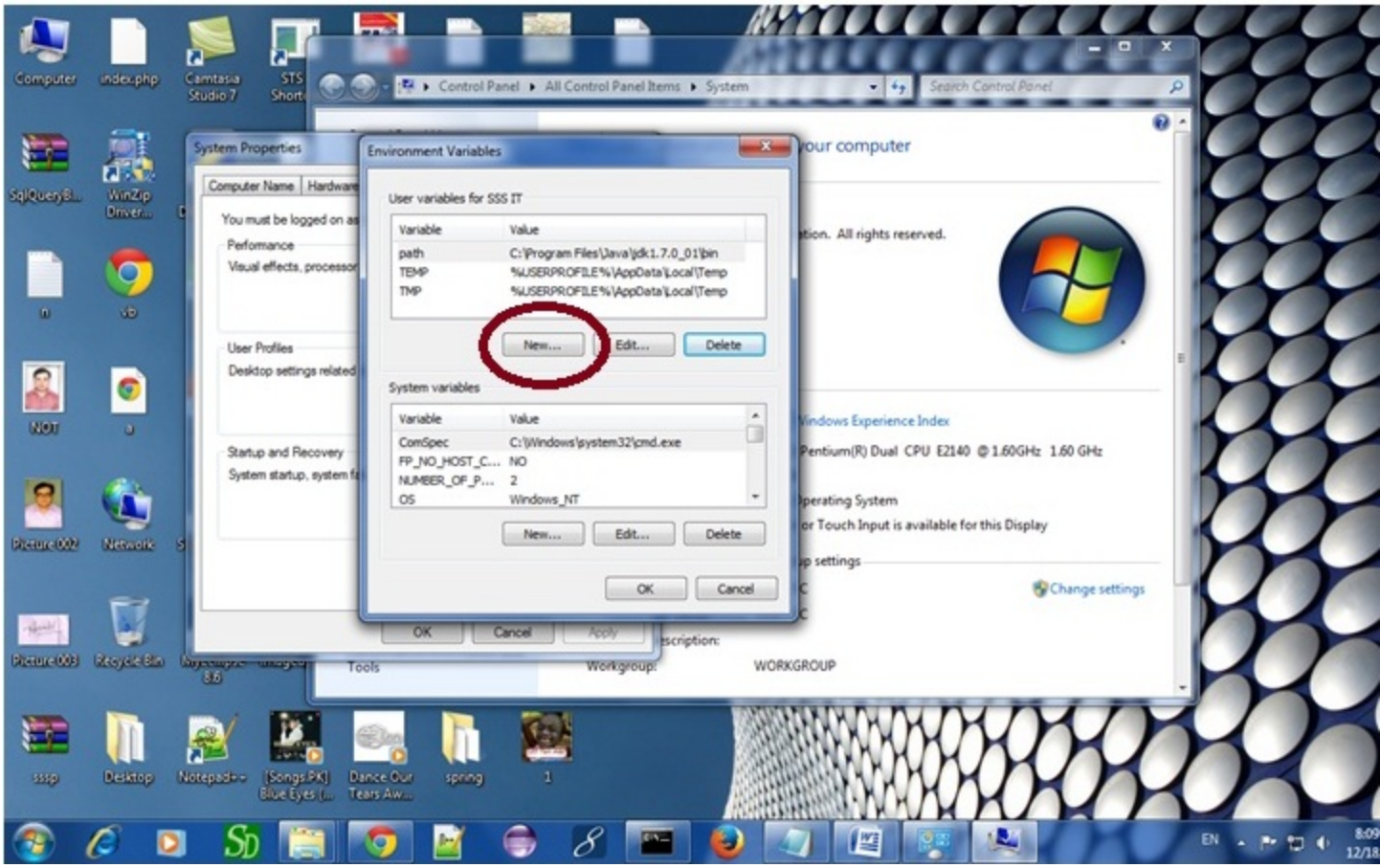
Go to My Computer properties:

Click on advanced system settings tab then environment variables:

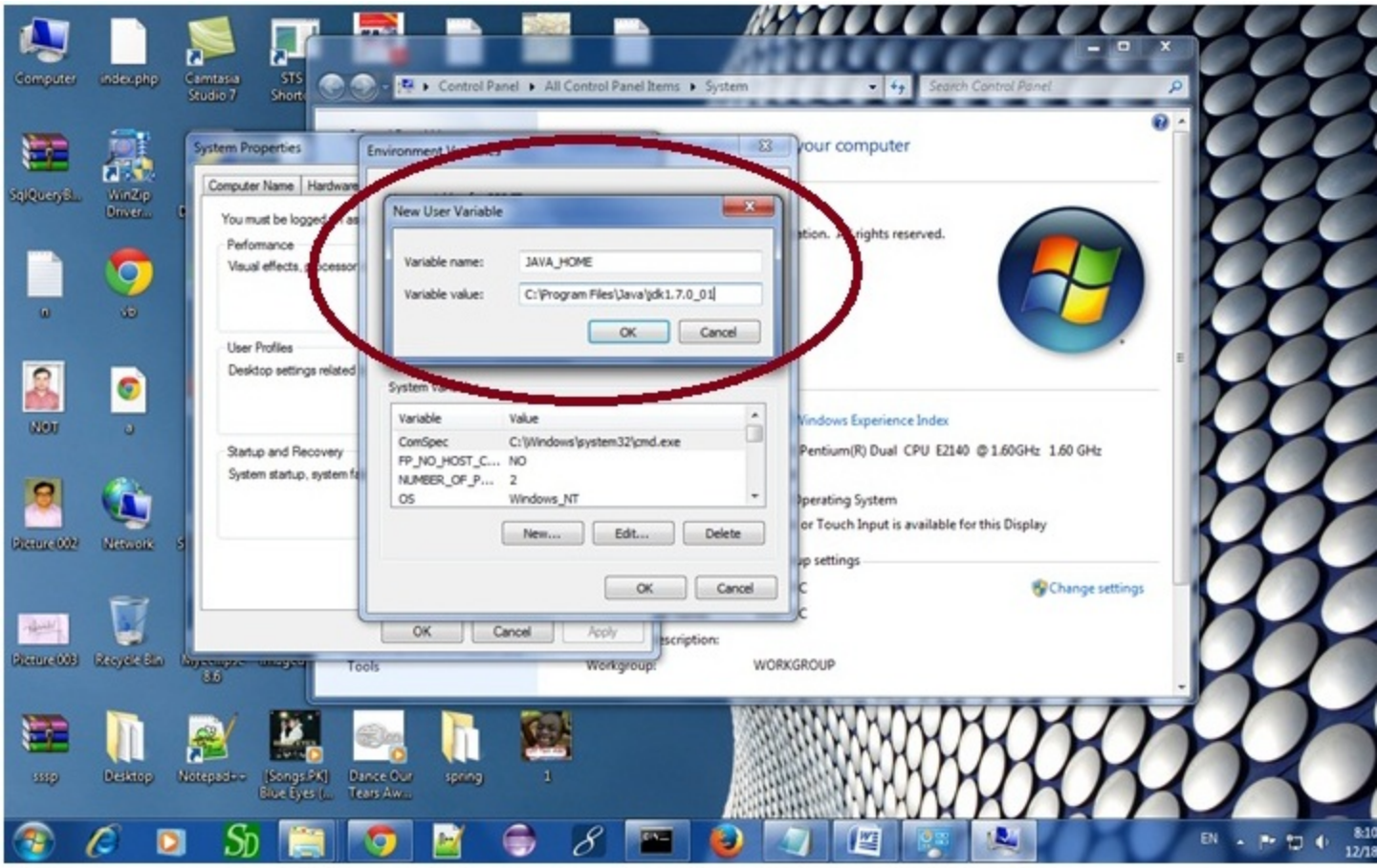

Control Panel ▸ All Control Panel Items ▸ System
Search Control Panel
Control Panel Home
Device Manager
Remote settings
System protection
Advanced system settings
View basic information about your computer
Windows edition
Windows 7 Ultimate
Copyright © 2009 Microsoft Corporation. All rights reserved.
System
Rating:
Windows Experience Index
Processor: Intel(R) Pentium(R) Dual CPU E2140 @ 1.60GHz 1.60 GHz
Installed memory (RAM): 2.00 GB
System type: 32-bit Operating System
Pen and Touch: No Pen or Touch Input is available for this Display
Computer name, domain, and workgroup settings
Computer name: SSSIT-PC
Change settings
Full computer name: SSSIT-PC
Computer description:
Workgroup: WORKGROUP
See also
Action Center
Windows Update
Performance Information and Tools



Click on the new tab of user variable or system variable:



Write JAVA_HOME in variable name and paste the path of jdk folder in variable value:



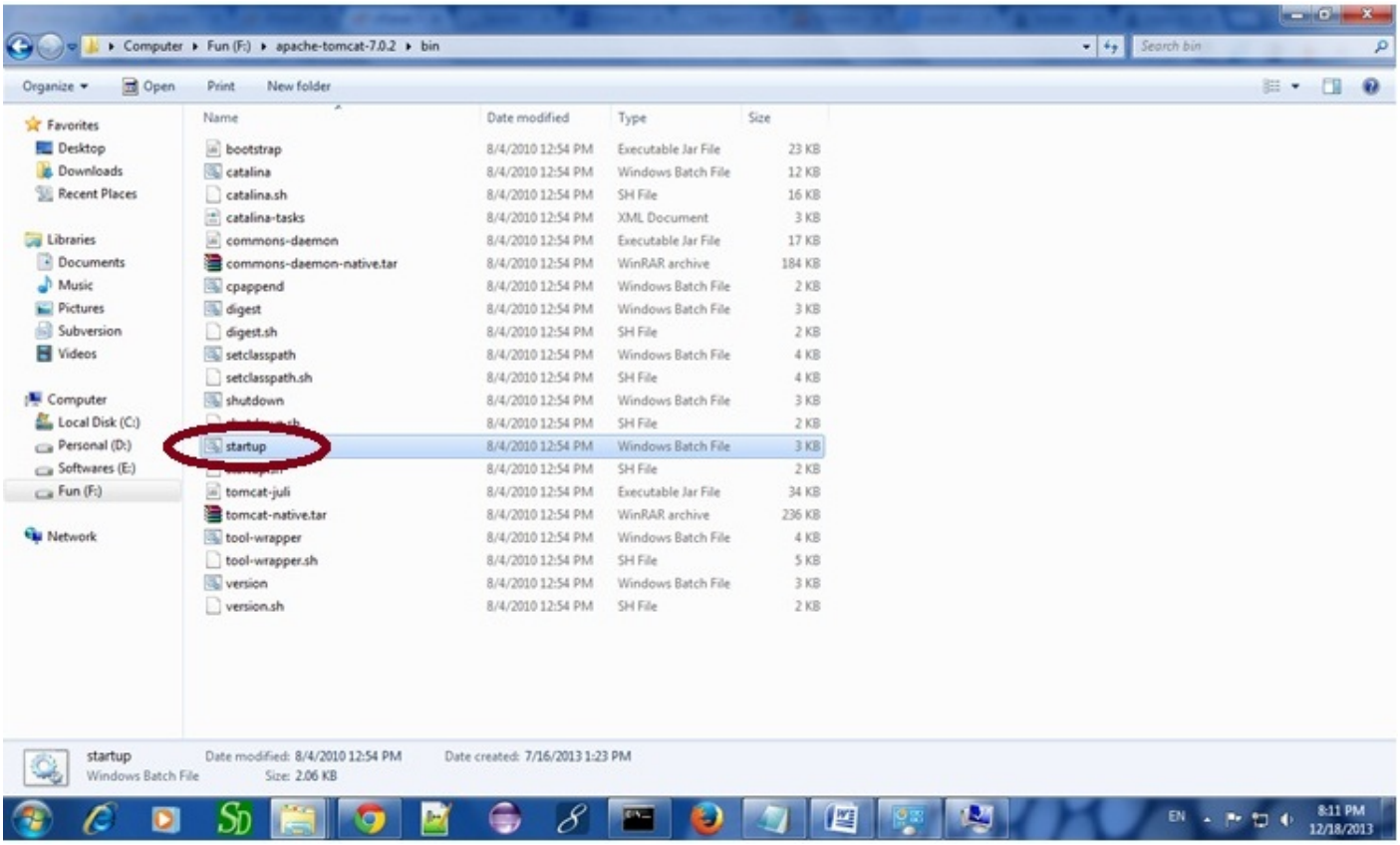
There must not be semicolon (;) at the end of the path.

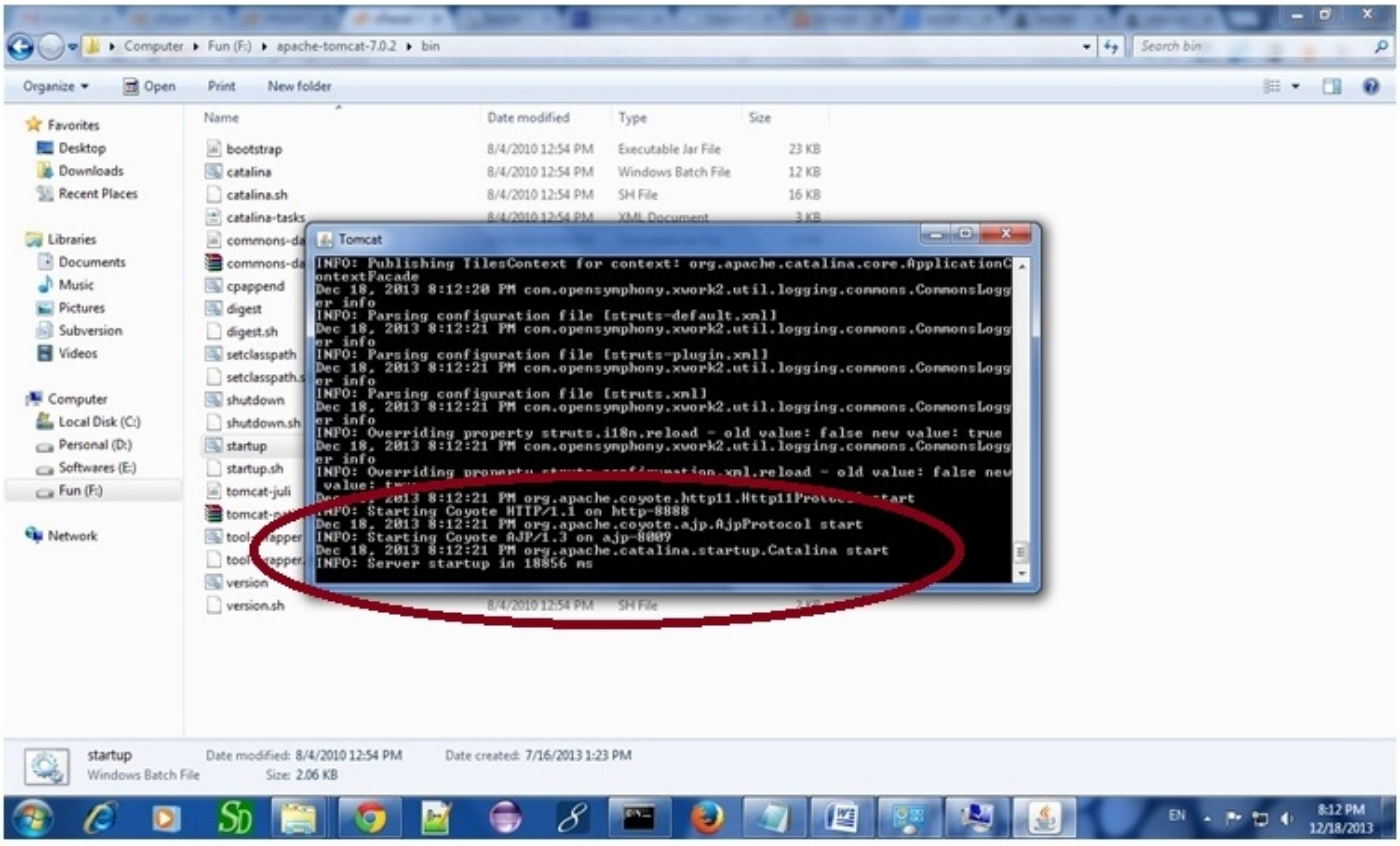
After setting the JAVA_HOME double click on the startup.bat file in apache tomcat/bin.

Note: There are two types of tomcat available:

1. Apache tomcat that needs to extract only (no need to install)
2. Apache tomcat that needs to install

It is the example of apache tomcat that needs to extract only.


Computer ▸ Fun (F:) ▸ apache-tomcat-7.0.2 ▸ bin
Search bin
Organize
Open
Print
New folder
Favorites
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Subversion
Videos
Computer
Local Disk (C:)
Personal (D:)
Softwares (E:)
Fun (F:)
Network
Name
Date modified
Type
Size
bootstrap 8/4/2010 12:54 PM Executable Jar File 23 KB
catalina 8/4/2010 12:54 PM Windows Batch File 12 KB
catalina.sh 8/4/2010 12:54 PM SH File 16 KB
catalina-tasks 8/4/2010 12:54 PM XML Document 3 KB
commons-daemon 8/4/2010 12:54 PM Executable Jar File 17 KB
commons-daemon-native.tar 8/4/2010 12:54 PM WinRAR archive 184 KB
cpappend 8/4/2010 12:54 PM Windows Batch File 2 KB
digest 8/4/2010 12:54 PM Windows Batch File 3 KB
digest.sh 8/4/2010 12:54 PM SH File 2 KB
setclasspath 8/4/2010 12:54 PM Windows Batch File 4 KB
setclasspath.sh 8/4/2010 12:54 PM SH File 4 KB
shutdown 8/4/2010 12:54 PM Windows Batch File 3 KB
8/4/2010 12:54 PM SH File 2 KB
startup 8/4/2010 12:54 PM Windows Batch File 3 KB
8/4/2010 12:54 PM SH File 2 KB
tomcat-juli 8/4/2010 12:54 PM Executable Jar File 34 KB
tomcat-native.tar 8/4/2010 12:54 PM WinRAR archive 236 KB
tool-wrapper 8/4/2010 12:54 PM Windows Batch File 4 KB
tool-wrapper.sh 8/4/2010 12:54 PM SH File 5 KB
version 8/4/2010 12:54 PM Windows Batch File 3 KB
version.sh 8/4/2010 12:54 PM SH File 2 KB
startup
Windows Batch File
Date modified: 8/4/2010 12:54 PM
Size: 2.06 KB
Date created: 7/16/2013 1:23 PM
EN
8:11 PM
12/18/2013



Now server is started successfully.

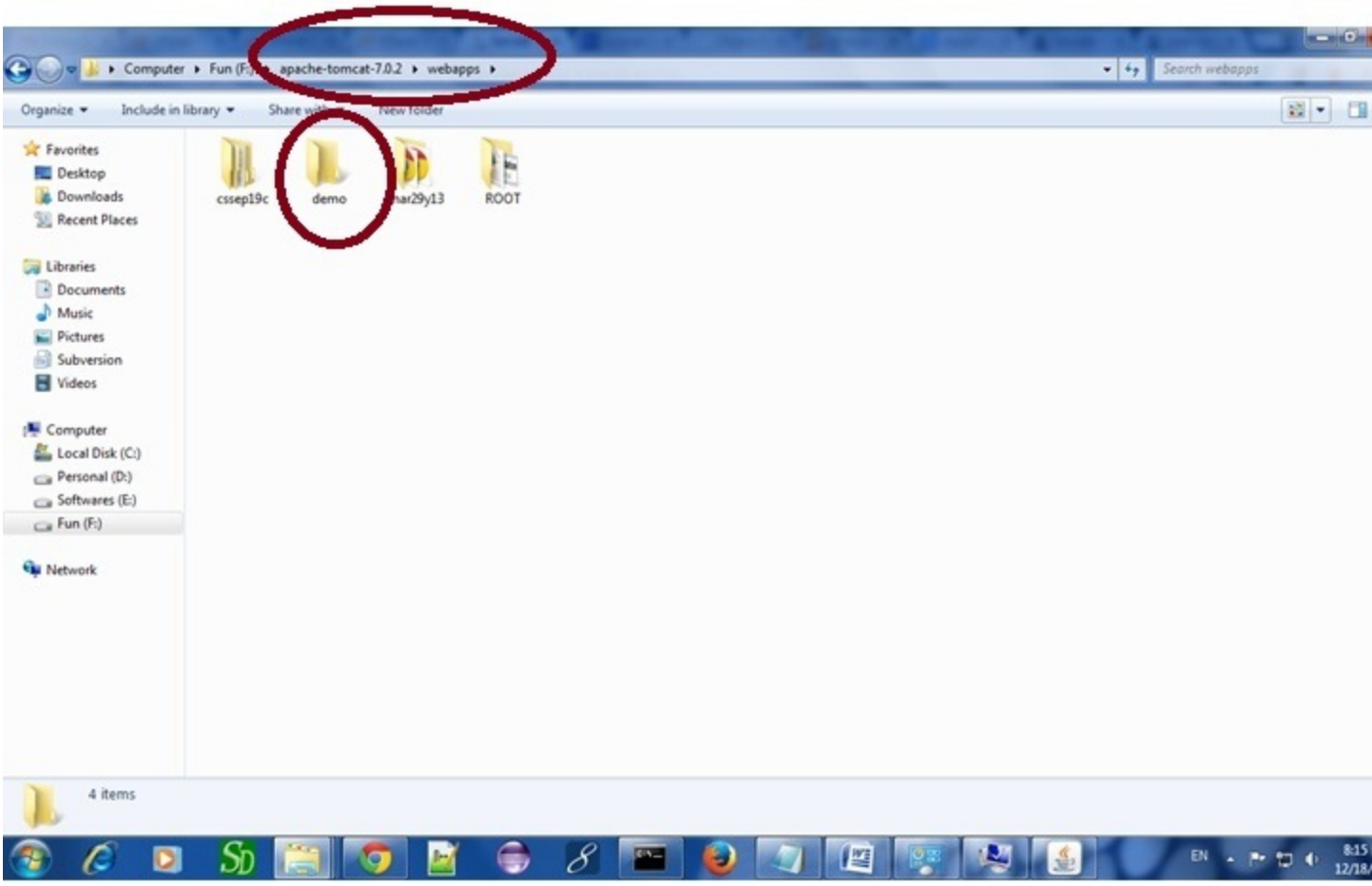
2) How to change port number of apache tomcat

Changing the port number is required if there is another server running on the same system with same port number.Suppose you have installed oracle, you need to change the port number of apache tomcat because both have the default port number 8080.

Open **server.xml file** in notepad. It is located inside the **apache-tomcat/conf** directory . Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. Let us replace it by 9999 and save this file.

5) How to deploy the servlet project

Copy the project and paste it in the webapps folder under apache tomcat.



But there are several ways to deploy the project. They are as follows:

- By copying the context(project) folder into the webapps directory
- By copying the war folder into the webapps directory
- By selecting the folder path from the server
- By selecting the war file from the server

Here, we are using the first approach.

You can also create war file, and paste it inside the webapps directory. To do so, you need to use jar tool to create the war file. Go inside the project directory (before the WEB-INF), then write:

1. projectfolder> jar cvf myproject.war *

Creating war file has an advantage that moving the project from one location to another takes less time.

6) How to access the servlet

Open broser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:

1. http://localhost:9999/demo/welcome

HANDLING CLIENT REQUEST THROUGH SERVLET

When a browser requests for a web page, it sends lot of information to the web server which cannot be read directly because this information travel as a part of header of HTTP request. You can check HTTP Protocol for more information on this.

Following is the important header information which comes from browser side and you would use very frequently in web programming −

Sr.No.	Header & Description
1	**Accept** This header specifies the MIME types that the browser or other clients can handle. Values of **image/png** or **image/jpeg** are the two most common possibilities.
2	**Accept-Charset** This header specifies the character sets the browser can use to display the information. For example ISO-8859-1.
3	**Accept-Encoding** This header specifies the types of encodings that the browser knows how to handle. Values of **gzip** or **compress** are the two most common possibilities.
4	**Accept-Language** This header specifies the client's preferred languages in case the servlet can produce results in more than one language. For example en, en-us, ru, etc
5	**Authorization** This header is used by clients to identify themselves when accessing password-

	protected Web pages.
6	**Connection** This header indicates whether the client can handle persistent HTTP connections. Persistent connections permit the client or other browser to retrieve multiple files with a single request. A value of **Keep-Alive** means that persistent connections should be used.
7	**Content-Length** This header is applicable only to POST requests and gives the size of the POST data in bytes.
8	**Cookie** This header returns cookies to servers that previously sent them to the browser.
9	**Host** This header specifies the host and port as given in the original URL.
10	**If-Modified-Since** This header indicates that the client wants the page only if it has been changed after the specified date. The server sends a code, 304 which means **Not Modified** header if no newer result is available.
11	**If-Unmodified-Since** This header is the reverse of If-Modified-Since; it specifies that the operation should succeed only if the document is older than the specified date.
12	**Referer** This header indicates the URL of the referring Web page. For example, if you are at Web page 1 and click on a link to Web page 2, the URL of Web page 1 is included in the Referrer header when the browser requests Web page 2.
13	**User-Agent** This header identifies the browser or other client making the request and can be used to return different content to different types of browsers.

Methods to read HTTP Header

There are following methods which can be used to read HTTP header in your servlet program. These methods are available with *HttpServletRequest* object

Sr.No.	Method & Description
1	**Cookie[] getCookies()** Returns an array containing all of the Cookie objects the client sent with this request.
2	**Enumeration getAttributeNames()** Returns an Enumeration containing the names of the attributes available to this request.
3	**Enumeration getHeaderNames()** Returns an enumeration of all the header names this request contains.
4	**Enumeration getParameterNames()** Returns an Enumeration of String objects containing the names of the parameters contained in this request
5	**HttpSession getSession()** Returns the current session associated with this request, or if the request does not have a session, creates one.
6	**HttpSession getSession(boolean create)** Returns the current HttpSession associated with this request or, if if there is no current session and value of create is true, returns a new session.
7	**Locale getLocale()** Returns the preferred Locale that the client will accept content in, based on the Accept-Language header.
8	

	Object getAttribute(String name) Returns the value of the named attribute as an Object, or null if no attribute of the given name exists.
9	**ServletInputStream getInputStream()** Retrieves the body of the request as binary data using a ServletInputStream.
10	**String getAuthType()** Returns the name of the authentication scheme used to protect the servlet, for example, "BASIC" or "SSL," or null if the JSP was not protected.
11	**String getCharacterEncoding()** Returns the name of the character encoding used in the body of this request.
12	**String getContentType()** Returns the MIME type of the body of the request, or null if the type is not known.
13	**String getContextPath()** Returns the portion of the request URI that indicates the context of the request.
14	**String getHeader(String name)** Returns the value of the specified request header as a String.
15	**String getMethod()** Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
16	**String getParameter(String name)** Returns the value of a request parameter as a String, or null if the parameter does not exist.
17	**String getPathInfo()** Returns any extra path information associated with the URL the client sent when

	it made this request
18	**String getProtocol()** Returns the name and version of the protocol the request.
19	**String getQueryString()** Returns the query string that is contained in the request URL after the path.
20	**String getRemoteAddr()** Returns the Internet Protocol (IP) address of the client that sent the request.
21	**String getRemoteHost()** Returns the fully qualified name of the client that sent the request.
22	**String getRemoteUser()** Returns the login of the user making this request, if the user has been authenticated, or null if the user has not been authenticated.
23	**String getRequestURI()** Returns the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.
24	**String getRequestedSessionId()** Returns the session ID specified by the client.
25	**String getServletPath()** Returns the part of this request's URL that calls the JSP.
26	**String[] getParameterValues(String name)** Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.
27	**boolean isSecure()**

	Returns a Boolean indicating whether this request was made using a secure channel, such as HTTPS.
28	**int getContentLength()** Returns the length, in bytes, of the request body and made available by the input stream, or -1 if the length is not known.
29	**int getIntHeader(String name)** Returns the value of the specified request header as an int.
30	**int getServerPort()** Returns the port number on which this request was received.

HTTP Header Request Example

Following is the example which uses **getHeaderNames()** method of HttpServletRequest to read the HTTP header information. This method returns an Enumeration that contains the header information associated with the current HTTP request.

Once we have an Enumeration, we can loop down the Enumeration in the standard manner, using *hasMoreElements()* method to determine when to stop and using *nextElement()* method to get each parameter name

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class DisplayHeader extends HttpServlet {

   // Method to handle GET method request.
   public void doGet(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "HTTP Header Request Example";
```

```
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n"+
         "<body bgcolor = \"#f0f0f0\">\n" +
         "<h1 align = \"center\">" + title + "</h1>\n" +
         "<table width = \"100%\" border = \"1\" align =
\"center\">\n" +
         "<tr bgcolor = \"#949494\">\n" +
         "<th>Header Name</th><th>Header Value(s)</th>\n"+
         "</tr>\n"
      );

      Enumeration headerNames = request.getHeaderNames();

      while(headerNames.hasMoreElements()) {
         String paramName = (String)headerNames.nextElement();
         out.print("<tr><td>" + paramName + "</td>\n");
         String paramValue = request.getHeader(paramName);
         out.println("<td> " + paramValue + "</td></tr>\n");
      }
      out.println("</table>\n</body></html>");
   }

   // Method to handle POST method request.
   public void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

Now calling the above servlet would generate the following result −

HTTP Header Request Example

Header Name	Header Value(s)
accept	*/*
accept-language	en-us

user-agent	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; InfoPath.2; MS-RTC LM 8)
accept-encoding	gzip, deflate
host	localhost:8080
connection	Keep-Alive
cache-control	no-cache

As discussed in the previous chapter, when a Web server responds to an HTTP request, the response typically consists of a status line, some response headers, a blank line, and the document. A typical response looks like this −

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
   (Blank Line)
<!doctype ...>
<html>
   <head>...</head>
   <body>
      ...
   </body>
</html>
```

The status line consists of the HTTP version (HTTP/1.1 in the example), a status code (200 in the example), and a very short message corresponding to the status code (OK in the example).

Following is a summary of the most useful HTTP 1.1 response headers which go back to the browser from web server side and you would use them very frequently in web programming −

Sr.No.	Header & Description
1	**Allow** This header specifies the request methods (GET, POST, etc.) that the server supports.

2	**Cache-Control** This header specifies the circumstances in which the response document can safely be cached. It can have values **public**, **private** or **no-cache** etc. Public means document is cacheable, Private means document is for a single user and can only be stored in private (non-shared) caches and nocache means document should never be cached.
3	**Connection** This header instructs the browser whether to use persistent in HTTP connections or not. A value of **close** instructs the browser not to use persistent HTTP connections and **keepalive** means using persistent connections.
4	**Content-Disposition** This header lets you request that the browser ask the user to save the response to disk in a file of the given name.
5	**Content-Encoding** This header specifies the way in which the page was encoded during transmission.
6	**Content-Language** This header signifies the language in which the document is written. For example en, en-us, ru, etc
7	**Content-Length** This header indicates the number of bytes in the response. This information is needed only if the browser is using a persistent (keep-alive) HTTP connection.
8	**Content-Type** This header gives the MIME (Multipurpose Internet Mail Extension) type of the response document.
9	**Expires** This header specifies the time at which the content should be considered out-of-date and thus no longer be cached.

10	**Last-Modified** This header indicates when the document was last changed. The client can then cache the document and supply a date by an **If-Modified-Since** request header in later requests.
11	**Location** This header should be included with all responses that have a status code in the 300s. This notifies the browser of the document address. The browser automatically reconnects to this location and retrieves the new document.
12	**Refresh** This header specifies how soon the browser should ask for an updated page. You can specify time in number of seconds after which a page would be refreshed.
13	**Retry-After** This header can be used in conjunction with a 503 (Service Unavailable) response to tell the client how soon it can repeat its request.
14	**Set-Cookie** This header specifies a cookie associated with the page.

Methods to Set HTTP Response Header

There are following methods which can be used to set HTTP response header in your servlet program. These methods are available with *HttpServletResponse* object.

Sr.No.	Method & Description
1	**String encodeRedirectURL(String url)** Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
2	**String encodeURL(String url)** Encodes the specified URL by including the session ID in it, or, if encoding is not

	needed, returns the URL unchanged.
3	**boolean containsHeader(String name)** Returns a Boolean indicating whether the named response header has already been set.
4	**boolean isCommitted()** Returns a Boolean indicating if the response has been committed.
5	**void addCookie(Cookie cookie)** Adds the specified cookie to the response.
6	**void addDateHeader(String name, long date)** Adds a response header with the given name and date-value.
7	**void addHeader(String name, String value)** Adds a response header with the given name and value.
8	**void addIntHeader(String name, int value)** Adds a response header with the given name and integer value.
9	**void flushBuffer()** Forces any content in the buffer to be written to the client.
10	**void reset()** Clears any data that exists in the buffer as well as the status code and headers.
11	**void resetBuffer()** Clears the content of the underlying buffer in the response without clearing headers or status code.
12	**void sendError(int sc)**

	Sends an error response to the client using the specified status code and clearing the buffer.
13	**void sendError(int sc, String msg)** Sends an error response to the client using the specified status.
14	**void sendRedirect(String location)** Sends a temporary redirect response to the client using the specified redirect location URL.
15	**void setBufferSize(int size)** Sets the preferred buffer size for the body of the response.
16	**void setCharacterEncoding(String charset)** Sets the character encoding (MIME charset) of the response being sent to the client, for example, to UTF-8.
17	**void setContentLength(int len)** Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.
18	**void setContentType(String type)** Sets the content type of the response being sent to the client, if the response has not been committed yet.
19	**void setDateHeader(String name, long date)** Sets a response header with the given name and date-value.
20	**void setHeader(String name, String value)** Sets a response header with the given name and value.
21	**void setIntHeader(String name, int value)** Sets a response header with the given name and integer value

22	**void setLocale(Locale loc)** Sets the locale of the response, if the response has not been committed yet.
23	**void setStatus(int sc)** Sets the status code for this response

HTTP Header Response Example

You already have seen setContentType() method working in previous examples and following example would also use same method, additionally we would use **setIntHeader()** method to set **Refresh** header.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class Refresh extends HttpServlet {

   // Method to handle GET method request.
   public void doGet(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      // Set refresh, autoload time as 5 seconds
      response.setIntHeader("Refresh", 5);

      // Set response content type
      response.setContentType("text/html");

      // Get current time
      Calendar calendar = new GregorianCalendar();
      String am_pm;
      int hour = calendar.get(Calendar.HOUR);
      int minute = calendar.get(Calendar.MINUTE);
      int second = calendar.get(Calendar.SECOND);

      if(calendar.get(Calendar.AM_PM) == 0)
         am_pm = "AM";
      else
         am_pm = "PM";
```

```
      String CT = hour+":"+ minute +":"+ second +" "+ am_pm;

      PrintWriter out = response.getWriter();
      String title = "Auto Refresh Header Setting";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n"+
         "<body bgcolor = \"#f0f0f0\">\n" +
         "<h1 align = \"center\">" + title + "</h1>\n" +
         "<p>Current Time is: " + CT + "</p>\n"
      );
   }

   // Method to handle POST method request.
   public void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

Now calling the above servlet would display current system time after every 5 seconds as follows. Just run the servlet and wait to see the result −

Auto Refresh Header Setting

Current Time is: 9:44:50 PM