UNIT II: Handling Client Request: Form DATA, Cookies and Session Tracking

Reading Form Data from Servlet, Example: Reading three parameter, Example: Reading all parameter, Filtering String for HTML –specific character, Benefits of cookies, Some problem with cookies, Sending and receiving cookies, Using cooking to detect first time visitors, Using cookies attributes, The need for session tracking, Session tracking basics, Session tracking API, Browser session Vs server sessions, A Servlets that shows per client access counts

READING FORM DATA FROM SERVLET:

Reading Form Data from Servlet is very easy. Servlets provides features that all data of form is parsed automatically.

For reading or handling form data servlet provides following functions:

1. **For Reading Single Values: getParameter**

   For reading single value from HTML Form servlets provides **request.getParameter**. **request.getParameterValues** is also used if the parameter value appears more than once, or you can call **request.getParameterNames** if you want a complete list of all parameters in the current request.

   Parameter names are case sensitive, for example:

```
request.getParameter("Param1"); and

request.getParameter("param1");
```

   are considered as different.

2. **For Reading Multiple Values: getParameterValues:**

   If the same parameter name might appear in the form data more than once, for that you can use getParameterValues (which returns an array of strings) instead of getParameter (which returns a single string corresponding to the first occurrence of the parameter).
   The return value of getParameterValues is null for nonexistent parameter names and is a one-element array when the parameter has only a single value.

   For multiselectable list boxes (i.e., HTML SELECT elements with the MULTIPLE attribute set;) you have to use **getParameterValues** function.

3. **Looking Up Parameter Names:getParameterNames and getParameterMap**

While reading HTML form, if the parameters names are not know them **getParameterNames and getParameterMap** can be used. It is very useful to get a full list of parameter names if the parameter names are not known.

Use getParameterNames to get this list in the form of an Enumeration, each entry of which can be cast to a String and used in a getParameter or getParameterValues call.

4. Reading Raw Form Data and Parsing Uploaded Files: getReader or getInputStream:

Rather than reading individual form parameters, you can access the query data directly by calling getReader or getInputStream on the HttpServletRequest and then using that stream to parse the raw input.
Note, however, that if you read the data in this manner, it is not guaranteed to be available with getParameter.
These functions are also used for reading uploaded files:

5. Reading Input in Multiple Character Sets: setCharacterEncoding

By default, request.getParameter reads input using the server's current character set. To change this default, we can use the setCharacterEncoding (For reading character in multiple language like Marathi, English, Japanese etc.) method of ServletRequest.

For example, to allow input in either English or Japanese, you can use the following.

```
request.setCharacterEncoding("JISAutoDetect");
String firstName = request.getParameter("firstName");
```

If input what to read the character in more than one character set? In such a case, you cannot simply call setCharacterEncoding with a normal character set name.

For that, here is an example that converts a parameter to Japanese:

```
String firstNameWrongEncoding = request.getParameter("firstName");
String firstName =new String(firstNameWrongEncoding.getBytes(), "Shift_JIS");
```

EXAMPLE: READING THREE PARAMETERS:

Following Programs presents a simple servlet called ThreeParams that reads form parameters named **param1, param2**, and **param3** and places their values in a bulleted list:

HTML Form for Reading from Servlets:

ThreeParamsForm.html

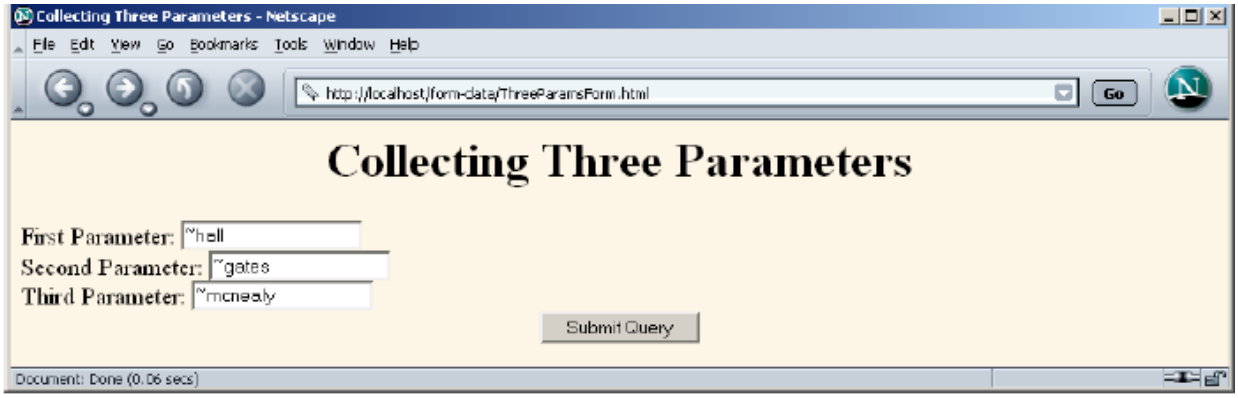
```
<HTML><HEAD><TITLE>Collecting Three Parameters</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Collecting Three Parameters</H1>
<FORM ACTION="servlet/ThreeParams" method =”post”>
First Parameter: <INPUT TYPE="TEXT" NAME="param1"><BR>
Second Parameter: <INPUT TYPE="TEXT" NAME="param2"><BR>
Third Parameter: <INPUT TYPE="TEXT" NAME="param3"><BR>
<CENTER><INPUT TYPE="SUBMIT"></CENTER>
</FORM>
</BODY></HTML>
```

Save the above **ThreeParamsForm.html** file to location:

C:\apache-tomcat-6.0.10\webapps\ROOT

Then URL Will Be:

http://localhost/ThreeParamsForm.html



ThreeParams.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
/* Simple servlet that reads three parameters from the form data.*/
public class ThreeParams extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String title = "Reading Three Request Parameters";
String param1=request.getParameter("param1");
String param2=request.getParameter("param2");
String param3=request.getParameter("param3");
out.println("<HTML>" +
"<HEAD><TITLE>" + title + "</TITLE></HEAD>" +
"<BODY BGCOLOR=\"#FDF5E6\">" +
"<H1 ALIGN=\"CENTER\">" + title + "</H1>" +
"<UL>" +
"<LI><B>param1</B>: "+ param1+
"<LI><B>param2</B>: "+ param2 +
"<LI><B>param3</B>: "+ param3 +
"</UL>" +
```

```
"</BODY></HTML>");

}

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException

{

        doGet(request, response);

}

}
```

Compile & Copy the .class file of this servlets to:

C:\apache-tomcat-6.0.10\webapps\ROOT\WEB-INF\classes

Then URL Will Be:

http://localhost/servlet/ThreeParams

EXAMPLE: READING ALL PARAMETER:

This example shows how to read all parameter from html form without knowing the names of parameter that are send to servlet.

First, the servlet looks up all the parameter names with the getParameterNames method of HttpServletRequest. This method returns an **Enumeration** that contains the parameter names in an unspecified order.

Next, the servlet loops down the Enumeration in the standard manner, using **hasMoreElements** to determine when to stop and using **nextElement** to get each parameter name.

nextElement returns an Object, the servlet casts the result to a String and passes that to getParameter.

The source code for the servlet & corresponding html form is shown as follows:

Reg.html

```
<HTML>

<HEAD>

<TITLE>Registration </TITLE>

</HEAD>

<BODY>
<H1 align="center">Registration Form </H1>

<FORM action="servlet/AllParams" method="post">

<TABLE border="1" align="center" width="70%">

<TR><TD>Full Name</TD><TD><Input Type="text" name="Name"></TD></TR>

<TR><TD>Address</TD><TD><TEXTAREA name="Address"></TEXTAREA></TD></TR>

<TR><TD>City</TD><TD><SELECT


name="City"><OPTION>Pune</OPTION><OPTION>Mumbai </OPTION><OPTION> Delhi


</OPTION></SELECT></TD></TR>

<TR><TD>Gender</TD><TD> <input type="Radio" name="Gender" value="Male">Male <Input


Type="Radio" name="gender1" value="Female">Female</TD></TR>

<TR><TD>Language Known</TD><TD> <Input Type="checkbox" name="Lang1" value=


"Hindi">Hindi <Input Type="checkbox" name= "Lang2" value="Marathi"> Marathi<Input
```

```
Type="checkbox" name="Lang3" value= "English"> English</TD></TR>
<TR><TD>Mobile No.</TD><TD><Input Type="text" name="mobile"></TD></TR>
<TR><TD>E-mail Address</TD><TD><Input Type="text" name="email"></TD></TR>
<TR><TD></TD><TD><Input Type="submit" value="Submit"></TD></TR>
<TR><TD></TD><TD><Input Type="Button" value="Cancel"></TD></TR>
</TABLE>
</FORM>
</BODY>
```

Save the above **Reg.html** file to location:

C:\apache-tomcat-6.0.10\webapps\ROOT

Then URL Will Be:

http://localhost/Reg.html

AllParams.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class AllParams extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
```

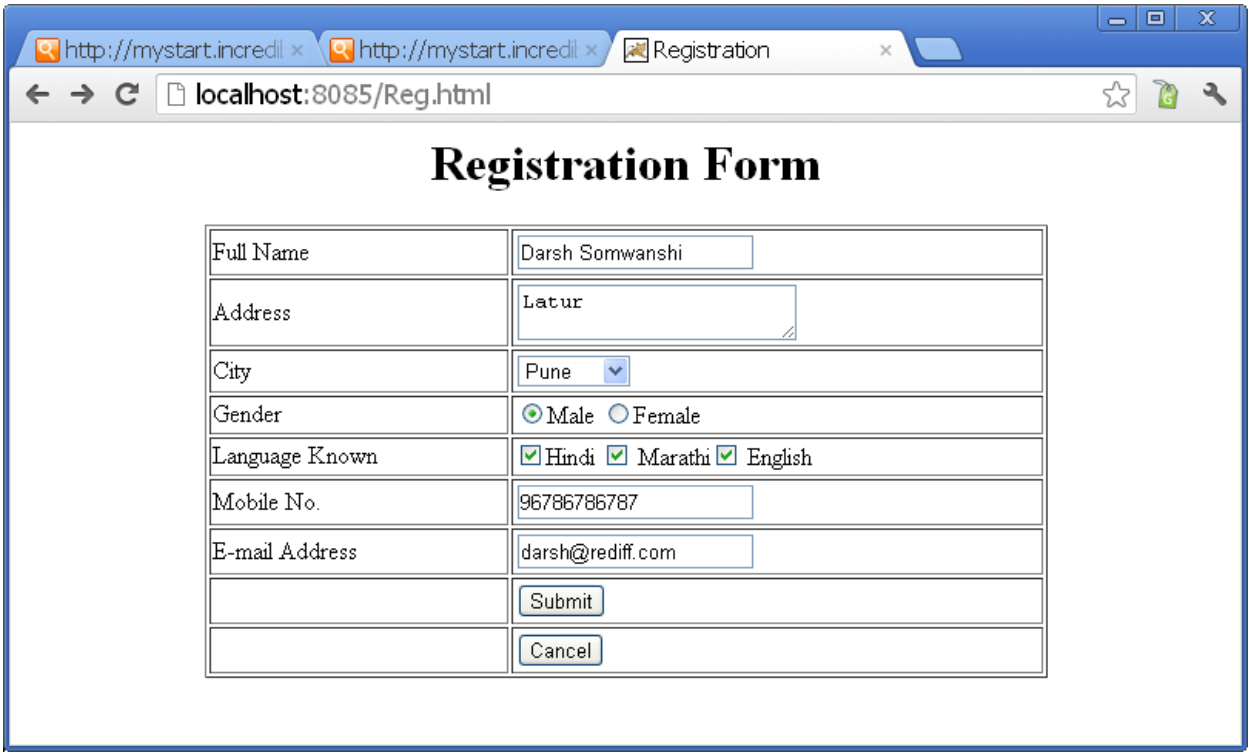
Compile & Copy the .class file of this servlets to:

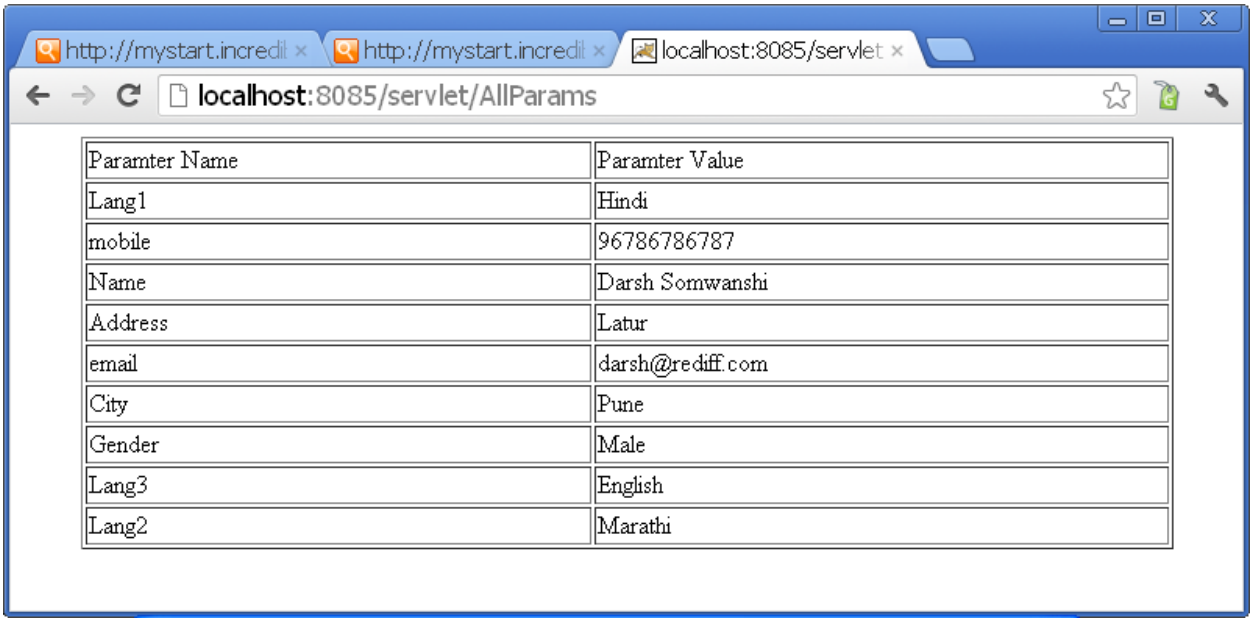
C:\apache-tomcat-6.0.10\webapps\ROOT\WEB-INF\classes

Then URL Will Be:

http://localhost/servlet/AllParams

```
PrintWriter out = response.getWriter();
String title = "Reading All Request Parameters";
out.println("<table border=1 align=center width=700>");
out.println("<tr><td>Paramter Name</td><td>Paramter Value</td></tr>");
Enumeration paramNames = request.getParameterNames();
while(paramNames.hasMoreElements())
{
String paramName = (String)paramNames.nextElement();
String paramValue =request.getParameter(paramName);
out.print("<TR><TD>" + paramName + "</TD>");
out.print("<TD>" + paramValue + "</TD></tr>");
}
out.println("</table>");
}
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
        doGet(request, response);
}
}
```


http://mystart.incredi
http://mystart.incredi
Registration
localhost:8085/Reg.html
Registration Form
Full Name
Darsh Somwanshi
Address
Latur
City
Pune
Gender
Male
Female
Language Known
Hindi
Marathi
English
Mobile No.
96786786787
E-mail Address
darsh@rediff.com
Submit
Cancel


http://mystart.incredi
http://mystart.incredi
localhost:8085/servlet
localhost:8085/servlet/AllParams
Paramter Name
Paramter Value
Lang1
Hindi
mobile
96786786787
Name
Darsh Somwanshi
Address
Latur
email
darsh@rediff.com
City
Pune
Gender
Male
Lang3
English
Lang2
Marathi

FILTERING STRINGS FOR HTML-SPECIFIC CHARACTERS:

When Generating HTML that contain characters like **<, >, "(Double Quote), and & (ampersand)** using servlet, we have to simply use <, >, " and & respectively because these are the standard HTML character. These are processed by web browser to interpret or display the result of those characters.

If we have not made such substitutions or changes, results will not be displayed properly.

In most cases, it is easy to note the special characters and use the standard HTML replacements manually.

However, there are two cases in which it is not so easy to make this substitution manually.

The first case in which manual conversion is difficult, when the string is derived from a program or another source in which it is already in some standard format.

Going through manually and changing all the special characters can be tedious in such a case, but forgetting to convert even one special character can result in your Web page having missing or improperly formatted sections.

The second case in which manual conversion fails is when the string is derived from HTML form data.

Here, the conversion must be performed at runtime, since the query data or data given by user is not known at compile time. If the user accidentally or deliberately enters HTML tags, the generated Web page will contain specious HTML tags and can have completely unpredictable results (the HTML specification tells browsers what to do with legal HTML; it says nothing about what they should do with HTML containing illegal syntax).

If you read request parameters and display their values in the resultant page, you should filter out the special HTML characters.

For Filtering & replacing character for HTML Specific Character we have to use **StringBuffer** class in java.

Following Program Demonstrate Reading string from HTML form that may contains HTML Specific Character & filtering these character to display proper output.

StringFilter.htm

```
<html>

<head>

<title>Enter Program in C</title>
```

```
</head>
<body>
<form method="POST" action="servlet/StringFilter">
 <div align="center">
  <center>
  <h1>Enter Program in C/C++ or HTML for Filtering HTML Specific Character</h1>
  <table border="1" width="70%">
   <tr>
    <td width="100%">
    <p align="center"><b><font size="4">Submit Your Program Here..</font></b></td>
   </tr>
   <tr>
    <td width="100%">
    <p align="center"><textarea rows="11" name="S1" cols="70"></textarea></td>
   </tr>
   <tr>
    <td width="100%">
    <p align="center"><input type="submit" value="Submit" name="B1"><input type="reset"
value="Reset" name="B2"></td>
   </tr>
  </table>
  </center>
 </div>
</form>
</body>
```

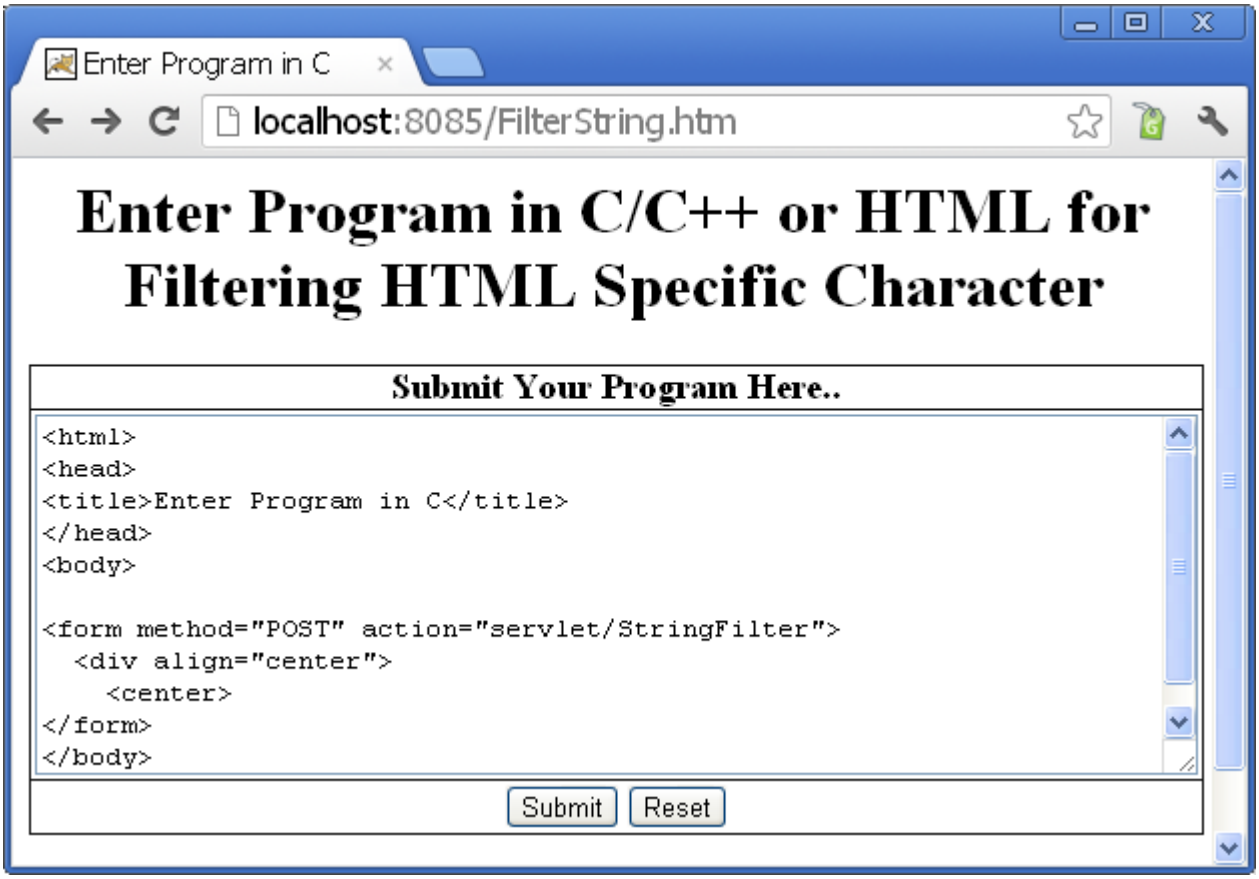
```
</html>
```

StringFilter.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class StringFilter extends HttpServlet
{
   public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws IOException, ServletException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html>");
      out.println("<head>");
      out.println("<title>String Filtering</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>String Filtering Servlet for Html Specific Character </h1>");
            String str=request.getParameter("S1");
            StringBuffer br=new StringBuffer(str.length());
            int len =str.length();
            for(int i=0;i<len;i++)
            {
                  char ch=(char)str.charAt(i);
                  switch(ch)
```

```
                {
                        case '<':
                        br.append("&lt;");
                        break;
                        case '>':
                        br.append("&gt;");
                        break;
                        case '"':
                        br.append("&quot;");
                        break;
                        case '&':
                        br.append("&amp;");
                        break;
                        default:
                        br.append(ch);
                }
          }
          out.println(br);
          out.println("</body>");
        out.println("</html>");
  }
public void doPost(HttpServletRequest request, HttpServletResponse response)
  throws IOException, ServletException
  {
          doGet(request,response);
```

```
  }

}
```


Enter Program in C
localhost:8085/FilterString.htm
Enter Program in C/C++ or HTML for Filtering HTML Specific Character
Submit Your Program Here..
<html>
<head>
<title>Enter Program in C</title>
</head>
<body>
<form method="POST" action="servlet/StringFilter">

</form>
</body>
Submit
Reset

BENEFITS OF COOKIES

Cookies are small bits of textual information that a Web server sends to a browser and that the browser later returns unchanged when visiting the same Web site or domain.

By letting the server read information it sent the client previously, the site can provide visitors with a number of conveniences such as presenting the site the way the visitor previously customized it or letting identifiable visitors in without their having to reenter a password.

There are four typical ways in which cookies can add value to your site. These are:

1. **Identifying a user during an e-commerce session.** Many online shopping sites provide facility to use a shopping cart or basket to put item into it, and then continue shopping.
   HTTP connection is usually closed after each page is sent, when a user selects a new item to add to the cart, how does the store know that it is the same user who put the previous item in the cart? This problem can be solved using cookies.

   Cookies create a small file on user's machine & stores are the items selected by user & then can be accessed from any page of website.

2. **Remembering usernames and passwords.** Cookies let a user log in to a site automatically, providing a significant convenience for users of unshared computers.

   Many large sites require you to register to use their services, but it is inconvenient to remember and enter the username and password each time you visit. Cookies are a good alternative for low-security sites. When a user registers, a cookie containing a unique user ID is sent to him. When the client reconnects at a later date, the user ID is returned automatically, the server looks it up, determines it belongs to a registered user that chose auto-login, and permits access without an explicit username and password.
   The site might also store the user's address, credit card number, and so forth in a database and use the user ID from the cookie as the key to retrieve the data. This approach prevents the user from having to reenter the data each time.

3. **Customizing sites.** Sites can use cookies to remember user preferences.
   Many sites let you customize the look of the main page. They might let you pick which weather report you want to see, what stock symbols should be displayed, what sports results you care about (yes, the Orioles are still losing), how search results should be displayed, and so on. Since it would be inconvenient for you to have to set up your page each time you visit their site, they use cookies to remember what you wanted.
   For simple settings, the site could accomplish this customization by storing the page settings directly in the cookies.
4. **Focusing advertising.** Cookies let the site remember which topics interest certain users and show advertisements relevant to those interests.
   With cookies, we can identify user interests by remembering user previous searches. This approach enables you to show directed ads on visits to users.

SOME PROBLEMS WITH COOKIES:

Providing convenience to the user and added value to the site owner is the purpose behind cookies. Cookies are not a serious security problem. Cookies are never interpreted or executed in any way and thus cannot be used to insert viruses or attack your system. And browsers generally only accept 20 cookies per site and 300 cookies total, and since browsers can limit each cookie to 4 kilobytes, cookies cannot be used to fill up someone's disk or launch other denial-of-service attacks.
But using Cookies is difficult to secure your privacy.
First, some people don't like the fact that search engines can remember what they previously searched for.
A second privacy problem occurs when sites depends on cookies for overly sensitive or important data. For example, some of the big online bookstores use cookies to remember you registration information and let you order without reentering much of your personal

information. In this case if someone using your computer, all your important information is known to him.

Because of that we need to remember following points while using Cookies:

1. Due to real and perceived privacy problems, some users turn off cookies. So, even when you use cookies to give added value to a site, whenever possible your site shouldn't depend on them.
2. You should be careful not to use cookies for particularly sensitive information, since this would open users up to risks if somebody accessed the user's computer or cookie files.
3. Cookies can be deleted.

SENDING AND RECEIVING COOKIES

To send cookies to the client, a servlet should use the Cookie constructor to create one or more cookies with specified names and values, set any optional attributes with **cookie.setXxx**, and insert the cookies into the HTTP response headers with response.addCookie.

To read incoming cookies, a servlet should call **request.getCookies**, which returns an array of Cookie objects corresponding to the cookies the browser has associated with your site (null if there are no cookies in the request).
The servlet should then loop down this array calling **getName** on each cookie until it finds the one whose name matches the name it was searching for, then call **getValue** on that Cookie to see the value associated with the name.

Sending Cookies to the Client:

Sending cookies to the client involves three steps:

1. **Creating a Cookie object**. You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.
   Neither the name nor the value should contain white space or any of the following characters: [ ] ( ) = , " / ? @ : ;
   For example, to create a cookie named userID with a value a1234, you would use the following.
   Cookie c = new Cookie ("userID", "a1234");

2. **Setting the maximum age**. If you want the browser to store the cookie on disk instead of just keeping it in memory, you use setMaxAge to specify how long (in seconds) the cookie should be valid.
   If you create a cookie and send it to the browser, by default a cookie that is stored in the browser's memory and deleted when the user quits the browser. If you want the browser to store the cookie on disk, use setMaxAge with a time in seconds, as below.

```
c.setMaxAge(60*60*24*7); // One week
```

3. **Placing the Cookie into the HTTP response headers**.
   You use **response.addCookie** to accomplish this.
   To send the cookie, use addCookie method of HttpServletResponse as follows:

```
Cookie userCookie = new Cookie("user", "uid1234");
userCookie.setMaxAge(60*60*24*365); // Store cookie for 1 year
response.addCookie(userCookie);
```

Reading Cookies from the Client:

To read the cookies that come back from the client, you should perform the following two tasks,

1. **Call request.getCookies.**
   To obtain the cookies that were sent by the browser, you call getCookies on the HttpServletRequest.
   This call returns an array of Cookie objects corresponding to the values that came in on the Cookie.
   If the request contains no cookies, getCookies should return **null**.

   Cookie[] cookies = request.getCookies();

2. **Loop Down the Cookie Array**
   Loop down the array, calling getName on each one until you find the cookie of interest. You then typically call getValue and use the value as per your requirement. Remember that cookies are specific to your host (or domain), not your servlet (or JSP page). So, although your servlet might send a single cookie, you could get many irrelevant cookies back. Once you find the cookie of interest, you typically call getValue on it and finish with some processing specific to the resultant value.

   For example:

```
String cookieName = "userID";
Cookie[] cookies = request.getCookies();
if (cookies != null)
{
for(int i=0; i<cookies.length; i++)
{
Cookie cookie = cookies[i];
if (cookieName.equals(cookie.getName()))
{
doSomethingWith(cookie.getValue());
```

```
}
}
}
```

USING COOKIES TO DETECT FIRST-TIME VISITORS:

Suppose that, at your site, you want to display an attractive banner to first-time visitors, telling them to register. But, you don't want to display the same banner to return visitors.

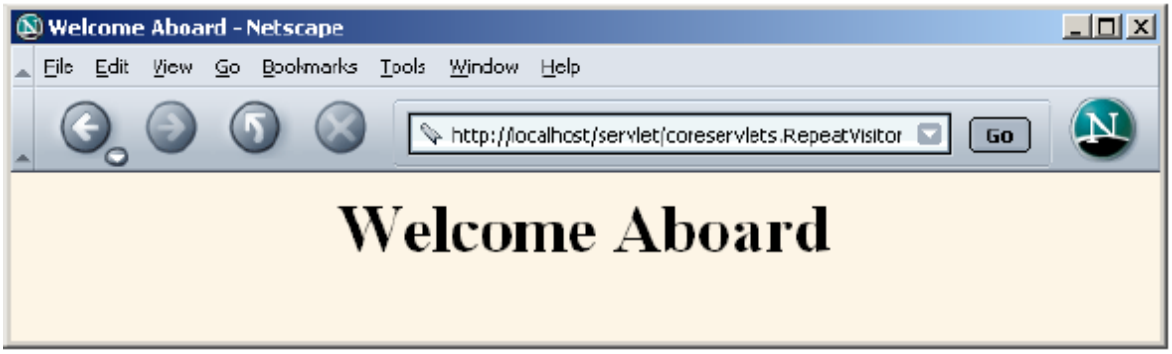
A cookie is the perfect way to differentiate first-timers from repeat visitors.

Check for the existence of a uniquely named cookie; if it is there, the client is a repeat visitor.
If the cookie is not there, the visitor is a newcomer, and you should set the cookie to detect latter.
Following program uses cookies to detect first time visitors:

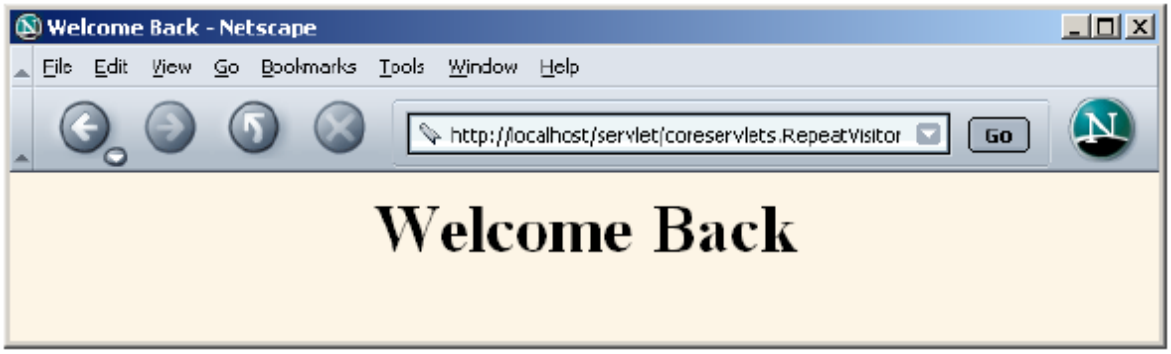
RepeatVisitor.java

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

public class RepeatVisitor extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
boolean newuser = true;
Cookie[] cookies = request.getCookies();
if (cookies != null)
{
for(int i=0; i<cookies.length; i++)
{
Cookie c = cookies[i];
if ((c.getName().equals("repeatVisitor")) && (c.getValue().equals("yes")))
{
newuser = false;
break;
}
}
}
```

```
String title;
if (newuser)
{
Cookie returnVisitorCookie =
new Cookie("repeatVisitor", "yes");
returnVisitorCookie.setMaxAge(60*60*24*365); // 1 year
response.addCookie(returnVisitorCookie);
title = "Welcome Aboard";
}
else
{
title = "Welcome Back";
}
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<HTML>" +
"<HEAD><TITLE>" + title + "</TITLE></HEAD>" +
"<BODY BGCOLOR=\"#FDF5E6\">" +
"<H1 ALIGN=\"CENTER\">" + title + "</H1>" +
"</BODY></HTML>");
}
}
```



First Visit of User



Second & subsequent visit of user.

USING COOKIE ATTRIBUTES:

For using cookie you can set various characteristics of the cookie by using the following setXxx methods, where Xxx is the name of the attribute you want to specify.

Each setXxx method has a corresponding getXxx method to retrieve the attribute value.

1. **public void setComment(String comment)**
   public String getComment()

   These methods specify or look up a comment associated with the cookie. The comment is used purely for informational purposes on the server; it is not sent to the client.
2. **public void setDomain(String domainPattern)**
   public String getDomain()
   These methods set or retrieve the domain to which the cookie applies. Normally, the browser returns cookies only to the exact same hostname that sent the cookies. For example, cookies sent from a servlet at **cocsit.org.in** would not normally get returned by the browser to pages at **onlineexam.cocsit.org.in** If the site wanted this to happen, the servlets could specify **cookie.setDomain(".cocsit.org.in ")**.
3. **public void setMaxAge(int lifetime)**
   public int getMaxAge()
   These methods tell how much time (in seconds) should elapse before the cookie expires. A negative value, which is the default, indicates that the cookie will last only for the current browsing session (i.e., until the user quits the browser) and will not be stored on disk.

4. **public String getName()**
   The getName method retrieves the name of the cookie. The name and the value are the two pieces you handle in web pages. The name is supplied to the Cookie constructor, there is no setName method; you cannot change the name once the cookie is created.
5. **public void setPath(String path)**
   public String getPath()

   These methods set or get the path to which the cookie applies. If you don't specify a path, the browser returns the cookie only to URLs in or below the directory containing the page that sent the cookie.
   To specify that a cookie apply to all URLs on your site, use cookie.setPath("/").

6. **public void setSecure(boolean secureFlag)**
   public boolean getSecure()
   This pair of methods sets or gets the boolean value indicating whether the cookie should only be sent over encrypted (i.e., SSL) connections. The default is false; the cookie should apply to all connections.

7. **public void setValue(String cookieValue)**
   public String getValue()

   The setValue method specifies the value associated with the cookie; getValue looks it up.
   The cookie value is supplied to the Cookie constructor, setValue is typically reserved for cases when you change the values of incoming cookies and then send them back out.

THE NEED FOR SESSION TRACKING:

Session is a time span between user login in and logout on a site or time span between the sites started in browsers & closes the browser.

HTTP is a "stateless" protocol: each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server does not automatically maintain related information about the client. There is no built-in support for maintaining related information. This lack of context causes a number of difficulties. For example, when clients at an online store add an item to their shopping carts, how does the server know what's already in the carts? Similarly, when clients decide to proceed to checkout, how can the server determine which previously created shopping carts are theirs?

There are three typical solutions to this problem: cookies, URL rewriting, and hidden form fields.

Cookies

You can use cookies to store an ID for a shopping session; with each subsequent connection, you can look up the current session ID and then use that ID to extract information about that session from a lookup table on the server machine. So, there would really be two tables: one that associates session IDs with user tables, and the user tables themselves that store user-specific data.

Using cookies in this manner is an excellent solution and is the most widely used approach for session handling, but for following reason session tracking is needed:

- Extracting the cookie that stores the session identifier from the other cookies.
- Determining when idle sessions have expired, and reclaiming them.

- Generating the unique session identifiers.

URL Rewriting

The client appends or attaches some extra data on the end of each URL. That data identifies the session, and the server associates that identifier with user-specific data it has stored.
For example, with http://localhost/path/file.html;jsessionid=a1234, the session identifier is attached as jsessionid=a1234, so a1234 is the ID that uniquely identifies the table of data associated with that user.

URL rewriting is a moderately good solution for session tracking and even has the advantage that it works when browsers don't support cookies or when the user has disabled them. However, if you implement session tracking yourself, URL rewriting has the same drawback as do cookies, namely, that the server-side program has a lot of straightforward but tedious processing to do.

Hidden Form Fields

HTML forms can have an entry that looks like the following:
<INPUT TYPE="HIDDEN" NAME="session" VALUE="a1234">
This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data. This hidden field can be used to store information about the session but has the major disadvantage that it only works if every page is dynamically generated by a form submission. Clicking on a regular
(<A HREF...>) hypertext link does not result in a form submission, so hidden form fields cannot support general session tracking, only tracking within a specific series of operations such as checking out at a store.

Session Tracking in Servlets

Servlets provide an outstanding session-tracking solution: the HttpSession API. This high-level interface is built on top of cookies or URL rewriting. All servers are required to support session tracking with cookies, and most have a setting by which you can globally switch to URL rewriting.

SESSION TRACKING BASICS:

Using sessions in servlets involves four basic steps:

1. Accessing the session object associated with the current request. Call request.getSession to get an HttpSession object, which is a simple table for storing user-specific data.

   You look up the HttpSession object by calling the getSession method of HttpServletRequest, as below.
   HttpSession session = request.getSession();

Behind the scenes, the system extracts a user ID from a cookie or attached URL data, then uses that ID as a key into a table of previously created HttpSession objects. But this is all done transparently to the programmer: you just call getSession.

2. Looking up information associated with a session. Call getAttribute on the HttpSession object, cast the return value to the appropriate type, and check whether the result is null.
   You can use **session.getAttribute("key")** to look up a previously stored value. The return type is Object, so you must do a typecast to whatever more specific type of data was associated with that attribute name in the session. The return value is null if there is no such attribute, so you need to check for null before calling methods on objects associated with sessions.

   Example:

```
HttpSession session = request.getSession();
SomeClass value =
(SomeClass)session.getAttribute("someIdentifier");
if (value == null) { // No such object already in session
value = new SomeClass(...);
session.setAttribute("someIdentifier", value);
}
doSomethingWith(value);
```

3. Storing information in a session. To specify information, use setAttribute. with a key and a value.

```
HttpSession session = request.getSession();
SomeClass value =
(SomeClass)session.getAttribute("someIdentifier");
if (value == null) { // No such object already in session
value = new SomeClass(...);
session.setAttribute("someIdentifier", value);
}
doSomethingWith(value);
```

4. Discarding session data. Call removeAttribute to discard a specific value. Call invalidate to discard an entire session. Call logout to log the client out of the Web server and invalidate all sessions associated with that user.

THE SESSION-TRACKING API:

Session-Tracking API (Application programming Interface) is a series or summary of the methods available in the HttpSession class used to perform operation of session values as per the requirement.

These API Are as follows:

1. **public Object getAttribute(String name)**
This method extracts a previously stored value from a session object. It returns null if no value is associated with the given name.

2. **public Enumeration getAttributeNames()**
This method returns the names of all attributes in the session.

3. **public void setAttribute(String name, Object value)**
This method set a value with a name.
4. **public void removeAttribute(String name)**
This method removes any values associated with the designated name.
5. **public void invalidate()**
This method invalidates or cancels the session and unbinds all objects associated with it.
6. **public void logout()**
This method logs the client out of the Web server and invalidates or cancels all sessions associated with that client. The scope of the logout is the same as the scope of the authentication.

7. **public String getId()**
This method returns the unique identifier generated for each session. It is useful for debugging or logging or, in rare cases, for programmatically moving values out of memory and into a database.

8. **public boolean isNew()**
This method returns true if the client (browser) has never seen the session, usually because the session was just created rather than being referenced by an incoming client request. It returns false for preexisting sessions.

9. **public long getCreationTime()**
This method returns the time in milliseconds since midnight, January 1, 1970 (GMT) at which the session was first built. To get a value useful for printing, pass the value to the Date constructor.

10. **public long getLastAccessedTime()**
This method returns the time in milliseconds since midnight, January 1, 1970

(GMT) at which the session was last accessed by the client.

11. public int getMaxInactiveInterval()
public void setMaxInactiveInterval(int seconds)

These methods get or set the length of time, in seconds, that a session should go without access before being automatically invalidated or canceled. A negative value specifies that the session should never time out.

BROWSER SESSION VS SERVER SESSIONS:

Browser session is a session created by browser of client machine. **Server session** is a session created by server. In **Server session** the session information are stored at server side.

By default, session-tracking is based on cookies that are stored in the browser's memory, not written to disk. Thus, unless the servlet explicitly reads the incoming cookie, sets the maximum age and path, and sends it back out, quitting the browser results in the session being broken: the client will not be able to access the session again because it is browser session. The problem, however, is that the server does not know that the browser was closed and thus the server has to maintain the session in memory until the inactive interval has been exceeded so it is called server session.

(Consider a physical shopping trip to a Wal-Mart store. You browse around and put some items in a physical shopping cart, then leave that shopping cart at the end of an aisle while you look for another item. A clerk walks up and sees the shopping cart.

Can he reshelf the items in it? No—you are probably still shopping and will come back for the cart soon. What if you realize that you have lost your wallet, so you get in your car and drive home? Can the clerk reshelf the items in your shopping cart now? Again, no—the clerk presumably does not know that you have left the store.

So, what can the clerk do? He can keep an eye on the cart, and if nobody has touched it for some period of time, he can then conclude that it is abandoned and take the items out of it. The only exception is if you brought the cart to him and said "I'm sorry, I left my wallet at home, so I have to leave.")

If you quit your browser, the session will effectively break. But the server does not know that you quit your browser. So, the server still has to wait for a period of time to see if the session has been timeout.

Sessions automatically become inactive when the amount of time between client accesses exceeds the interval specified by getMaxInactiveInterval. When this happens, objects stored in the HttpSession object are removed.

A SERVLET THAT SHOWS PER-CLIENT ACCESS COUNTS:

Following Programs shows a simple servlet that shows basic information about the client's session. When the client connects, the servlet uses request.getSession either to retrieve the existing session or, if there is no session, to create a new one. The servlet then looks for an attribute called accessCount of type Integer. If it cannot find incremented and associated with the session by setAttribute.

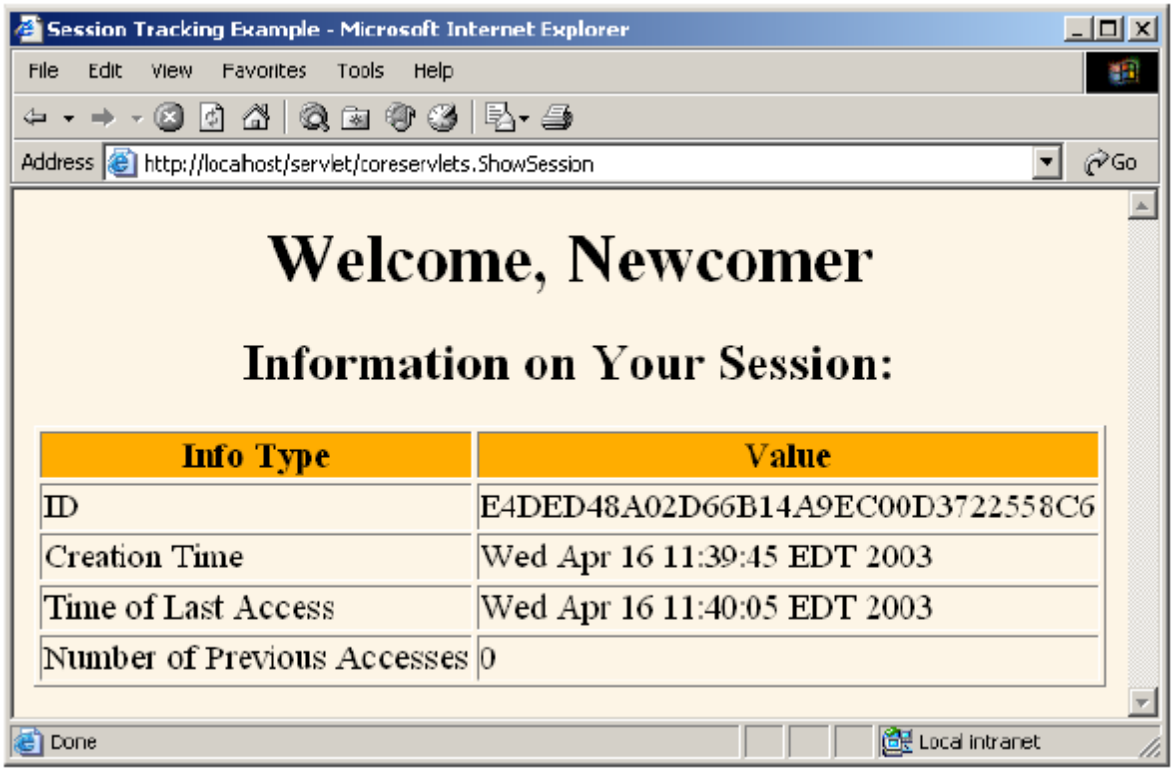
Finally, the servlet prints a small HTML table showing information about the session.

ShowSession.java

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

public class ShowSession extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
HttpSession session = request.getSession();

String heading;
Integer accessCount =(Integer)session.getAttribute("accessCount");
if (accessCount == null)
{
    accessCount = new Integer(0);
    heading = "Welcome, Newcomer";
}
else
{
    heading = "Welcome Back";
    accessCount = new Integer(accessCount.intValue() + 1);
}

session.setAttribute("accessCount", accessCount);
```

```
PrintWriter out = response.getWriter();
String title = "Session Tracking Example";
String docType =
out.println("<HTML>" +
"<HEAD><TITLE>" + title + "</TITLE></HEAD>" +
"<BODY BGCOLOR=\"#FDF5E6\">" +
"<CENTER>" +
"<H1>" + heading + "</H1>" +
"<H2>Information on Your Session:</H2>" +
"<TABLE BORDER=1>" +
"<TR BGCOLOR=\"#FFAD00\">" +
"<TH>Info Type<TH>Value" +
"<TR>" +
"<TD>ID" +
"<TD>" + session.getId() +
"<TR>\n" +
"<TD>Creation Time" +
"<TD>" + new Date (session.getCreationTime()) +
"<TR>" +
"<TD>Time of Last Access" +
"<TD>" + new Date (session.getLastAccessedTime()) +
"<TR>" +
"<TD>Number of Previous Accesses" +
"<TD>" + accessCount +
"</TABLE> +
"</CENTER></BODY></HTML>");
}
}
```

http://localhost/servlet/ShowSession


Session Tracking Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/servlet/coreservlets.ShowSession
Go
Welcome, Newcomer
Information on Your Session:
Info Type
Value
ID
E4DED48A02D66B14A9EC00D3722558C6
Creation Time
Wed Apr 16 11:39:45 EDT 2003
Time of Last Access
Wed Apr 16 11:40:05 EDT 2003
Number of Previous Accesses
0
Done
Local intranet