COP 4610L: Applications in the Enterprise Spring 2005

Introduction to Servlet Technology – Part 4

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cop4610L/spr2005

School of Electrical Engineering and Computer Science
University of Central Florida

Session Tracking and Servlets

- Many e-businesses personalize users' browsing experiences, tailoring web pages to their users' individual preferences and allowing them to bypass irrelevant content.
- This is typically done by tracking the user's movement through the Internet and combining that data with information provided by the users themselves, such as billing information, interests and hobbies, among other things.
- Personalization of the Internet has become rather commonplace today with many sites even allowing their clients the ability to customize their homepage to fit individual user likes/needs (see MSN.com, CNN.com or numerous other sites).
- This increase in personalization of the Internet has also given rise to the problems of privacy invasion. What happens when the e-business to which you give your personal data sells or gives that data to another organization without your knowledge?

Session Tracking and Servlets (cont.)

- As we have discussed before, the request/response mechanism of the Internet is based on HTTP.
- Unfortunately, HTTP is a stateless protocol – it does not support persistent information that could help a web server determine that a request is from a particular client.
- As far as a web server is concerned, every request could be from the same client or every request could be from a different client. Thus, sites like MSN.com and CNN.com need a mechanism to identify individual clients.
- To help the server distinguish between clients, each client must identify itself to the server. There are a number of popular techniques for distinguishing between clients.
- Two common techniques are cookies and session tracking we'll look at both of these mechanisms. Two other techniques are hidden forms and URL-rewriting.

Cookies

- Cookies are a popular technique for customizing web pages. Browsers can store cookies on the user's computer for retrieval later in the same browsing session or in future browsing sessions.
- For example, cookies are used in on-line shopping applications to store unique identifiers for the users. When users add items to their on-line shopping carts or perform other tasks resulting in a request to the web server, the server receives cookies containing unique identifiers for each user. The server then uses the unique identifier to locate the shopping carts and perform any necessary processing.
- Cookies are also used to indicate the client's shopping preferences. When the servlet receives the client's nest communication, the servlet can examine the cookie(s) it sent to the client in a previous communication, identify the client's preferences and immediately display products of interest to that particular client.

Cookies (cont.)

- Cookies are text-based data that are sent by servlets (or other similar server-side technologies like JSPs and PHP that we will see later) as part of responses to clients.
- Every HTTP-based interaction between a client and a server includes a header containing information about the request (when the communication is from the client to the server) or information about the response (when the communication is from the server to the client).
- When an `HTTPServlet` receives a request the header includes information such as the request type (e.g., `get` or `post`) and the cookies that are sent by the server to be stored on the client machine. When the server prepares its response, the header information includes any cookies the server wants to store on the client computer and other information such as the MIME type of the response.

Cookies (cont.)

- Depending on the maximum age of a cookie, the web browser either maintains the cookie for the duration of the browsing session (i.e., until the user closes the web browser) or stores the cookie on the client computer for future use.
- When a browser requests a resource from a server, cookies that were previously sent to the client by that server are returned to the server as part of the request formulated by the browser.
- Cookies are deleted automatically when they expire (i.e., reach their maximum age).
- Browsers that support cookies must be able to store a minimum of 20 cookies per web site and 300 cookies per user. Browsers may limit the cookie size to 4K.
- Each cookie stored on the client contains a domain. The browser sends a cookie only to the domain stored in the cookie.

Cookies (cont.)

- The next example shows how a cookie can be used to differentiate between first-time and repeat visitors to our servlet. To do this our servlet needs to 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.
- This example, will use a cookie for this purpose. The cookie will be named "RepeatVisitor".
- Recall that by default, a cookie exists only during the current browsing session. The cookie in this example is a persistent cookie with a lifetime of 1 minute (see the code on the next page).

RepeatVisitor Servlet

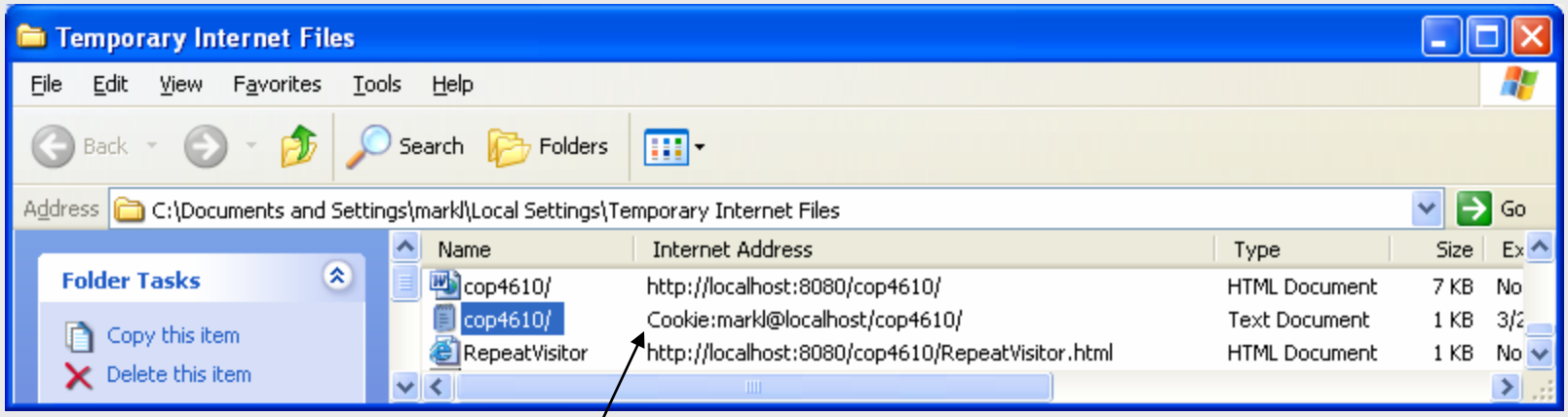
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Servlet that says "Welcome aboard" to first-time
//  visitors and "Welcome back" to repeat visitors.

public class RepeatVisitor extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    boolean newbie = 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")) &&
            // Could omit test and treat cookie name as a flag
            (c.getValue().equals("yes"))) {
          newbie = false;
          break;
        }
      }
    }
    String title;
```

If the cookie name is "RepeatVisitor" and the value of the cookie is "yes" then the visitor has been here before.

RepeatVisitor Servlet (cont.)

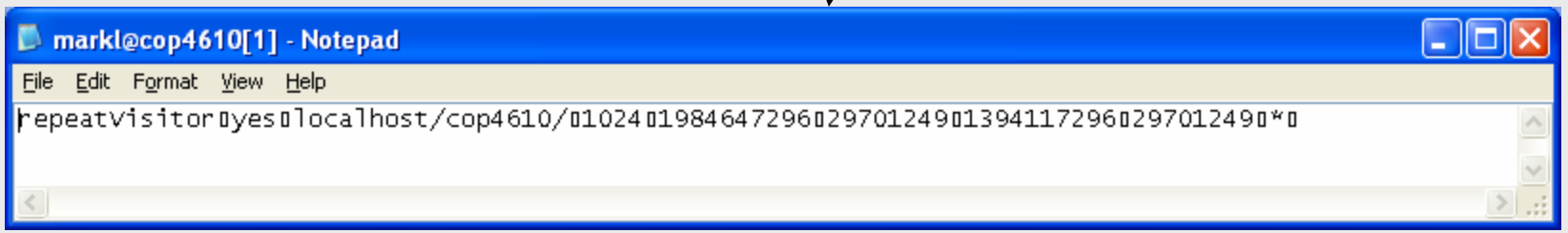
```
if (newbie) {
    Cookie returnVisitorCookie = new Cookie("repeatVisitor", "yes");
    // returnVisitorCookie.setMaxAge(60*60*24*365); // 1 year
     returnVisitorCookie.setMaxAge(60); //cookie expires in 1 minute
     response.addCookie(returnVisitorCookie);
     title = "Welcome Aboard";
} else {  title = "Welcome Back";  }
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println ("<body bgcolor=white background=images/background.jpg
lang=EN-US link=blue vlink=blue >");
    out.println ("<body style='tab-interval:.5in'>");
    out.println ("<font size = 5>");
    out.println ("<br>");
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                "</BODY></HTML>");
  }
}
```

Set cookie expiration date and send it to the client.



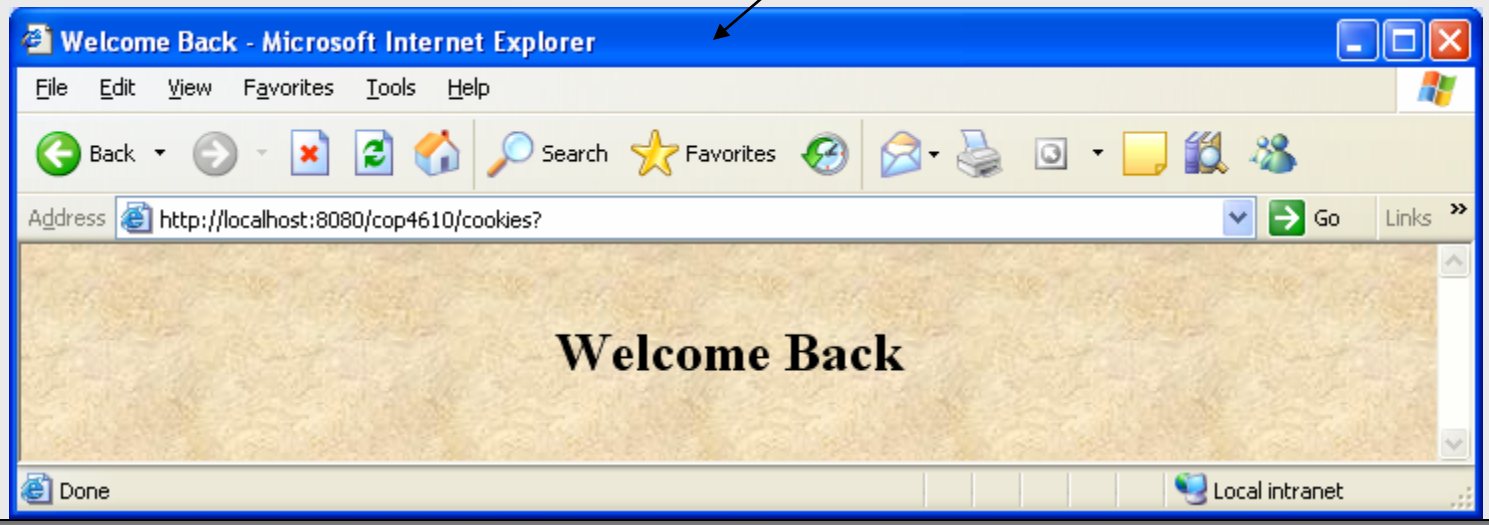
This is the cookie written by the server "localhost".

This is the contents of the cookie written by the RepeatVisitor servlet



First visit by a client to the RepeatVisitor servlet

Subsequent visit by a client to the RepeatVisitor servlet (within 1 minute)



Using Cookies Attributes

- Before adding the cookie to the outgoing headers, you can ser various characteristics of the cookie by using the following `set` methods.
- Although each `set` method has a corresponding `get` method to retrieve the attribute value, note that the attributes are part of the header sent from the server to the browser; they are **not** part of the header returned by the browser to the server.
- Except for `name` and `value`, the cookie attributes apply only to outgoing cookies from the server to the client; they are not set on cookies that come from the browser to the server. This means that these attributes are not available in the cookies that you get by means of `request.getCookies`.
- A brief description of the methods for setting and getting cookie attribute values are shown on the next page.

setComment(string) getComment()	Specify or look up a comment associated with the cookie.
setDomain(string) getDomain()	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.
setMaxAge(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 and will not be stored on disk. Specifying a value of 0 instructs the browser to delete the cookie.
getName()	Retrieves the name of the cookie. The name and the value are the two pieces of information which are the most important.
setPath(string) getPath()	Sets or retrieves the path to which the cookie applies. If you do not specify a path, the browser returns the cookie only to URLs in or below the directory containing the page that sent the cookie.
setSecure(boolean) getSecure()	Sets or retrieves the boolean value which indicates whether the cookie should only be sent over encrypted connections. The default is false.
setValue(string) getValue()	Sets or retrieves the value associated with the cookie.

Differentiating Session Cookies From Persistent Cookies

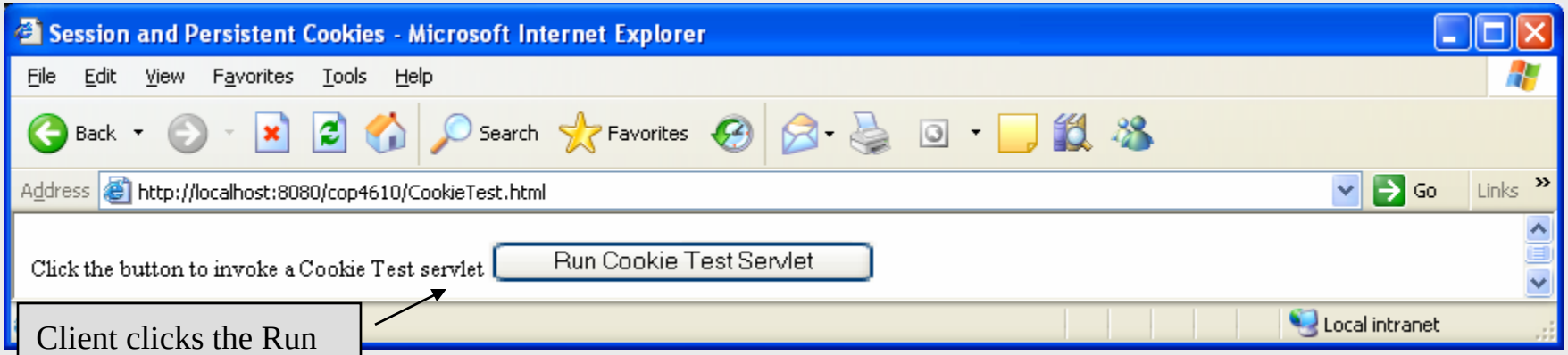
- The next example illustrates the use of cookie attributes by contrasting the behavior of cookies with and without a maximum age.
- This servlet called CookieTest, performs two tasks

1. First the servlet sets six outgoing cookies. Three have no explicit age (i.e., they have a negative value by default), meaning that they will apply only to the current browsing session – until the client restarts the browser. The other three cookies use `setMaxAge` to stipulate that the browser should write them to disk and that they should persist for the next 15 minutes (15 * 60 = 900 seconds), regardless of whether the client restarts the browser or not.
2. Second, the servlet uses request.getCookies to find all the incoming cookies and display their names and values in an HTML table.

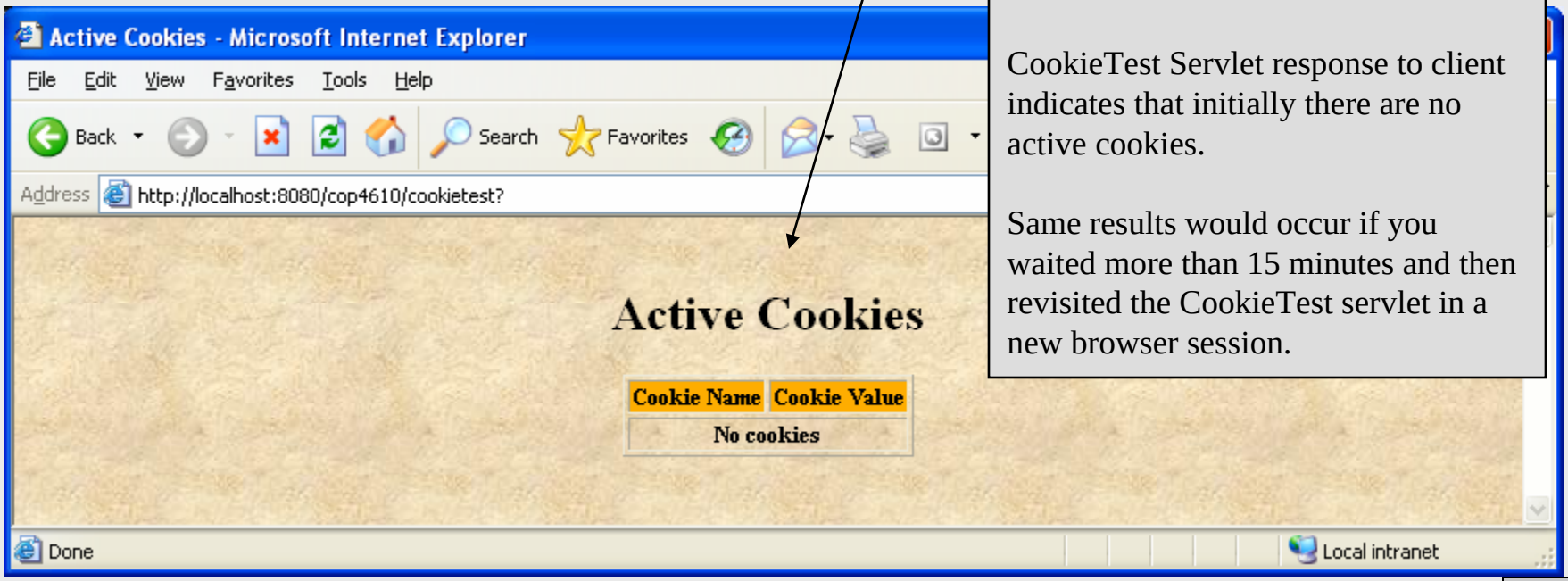
CookieTest Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
//  Creates a table of the cookies associated with
//  the current page. Also sets six cookies: three
//  that apply only to the current session
//  (regardless of how long that session lasts)
//  and three that persist for an hour (regardless
//  of whether the browser is restarted).
public class CookieTest extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    for(int i=0; i<3; i++) {
      // Default maxAge is -1, indicating cookie
      // applies only to current browsing session.
      Cookie cookie = new Cookie("Session-Cookie-" + i,
                                 "Cookie-Value-S" + i);
      response.addCookie(cookie);
      cookie = new Cookie("Persistent-Cookie-" + i,
                          "Cookie-Value-P" + i);
      // Cookie is valid for 15 minutes, regardless of whether
      // user quits browser, reboots computer, or whatever.
      cookie.setMaxAge(900); //cookie expires in 15 minutes
      response.addCookie(cookie);
    }
```


```
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String docType ="<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
          "Transitional//EN\">\n";
        String title = "Active Cookies";
        out.println ("<body bgcolor=white background=images/background.jpg
    lang=EN-US link=blue vlink=blue >");
        out.println ("<body style='tab-interval:.5in'>");
        out.println ("<font size = 5>");  out.println ("<br>");
        out.println(docType +"<HTML>\n" + "<HEAD><TITLE>" + title +
                             "</TITLE></HEAD>\n" + "<BODY
                        BGCOLOR=\"#FDF5E6\">\n" +
                    "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                    "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
                    "<TR BGCOLOR=\"#FFAD00\">\n" + "  <TH>Cookie Name\n" +
                    "  <TH>Cookie Value");
        Cookie[] cookies = request.getCookies();
        if (cookies == null) {
          out.println("<TR><TH COLSPAN=2>No cookies");
        } else {
          Cookie cookie;
          for(int i=0; i<cookies.length; i++) {
            cookie = cookies[i];
            out.println("<TR>\n" +
                        "  <TD>" + cookie.getName() + "\n" +
                        "  <TD>" + cookie.getValue());
          }  }   out.println("</TABLE></BODY></HTML>");
      }  }
```

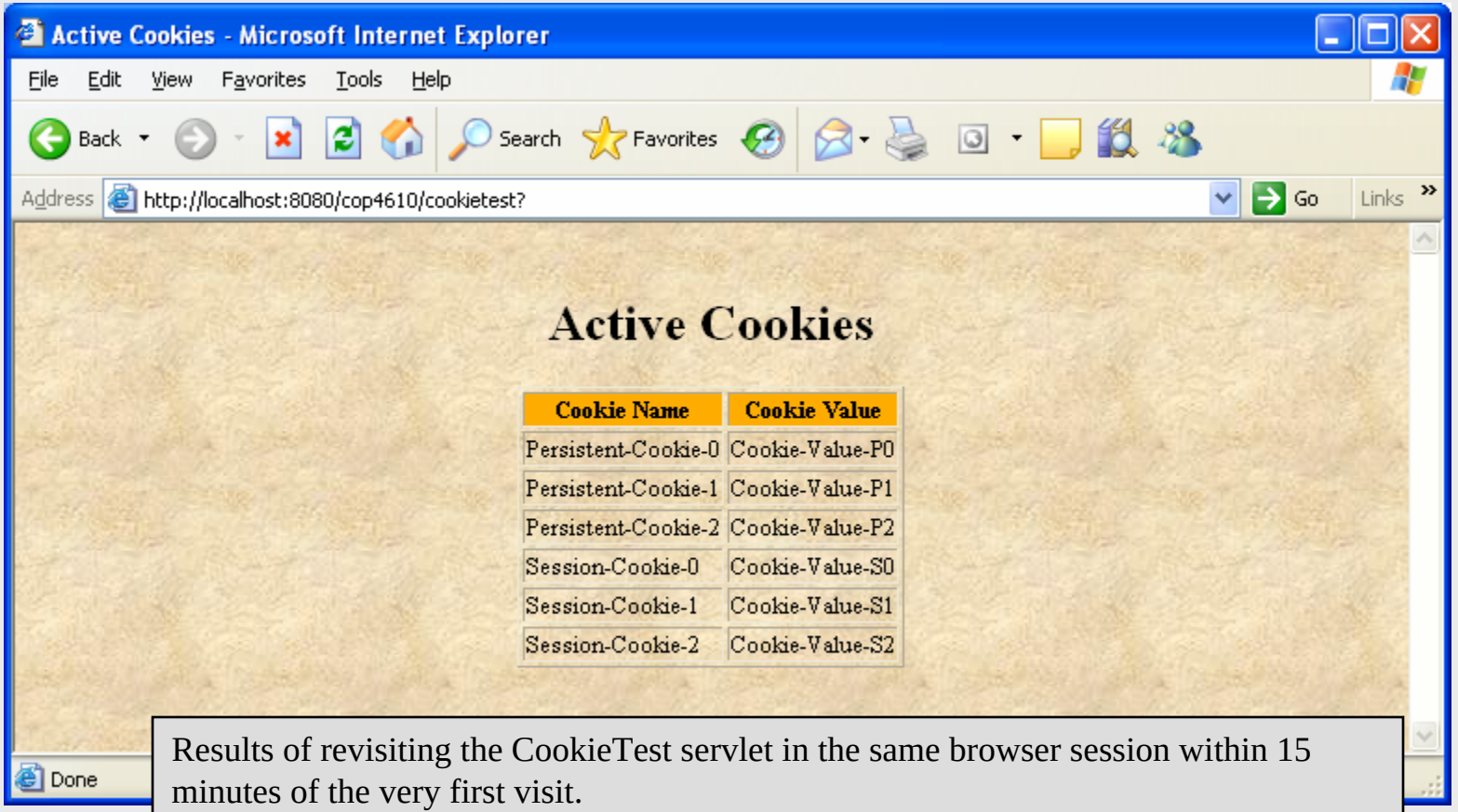
Client clicks the Run Cookie Test Servlet button from HomePage



Results of initial visit to the CookieTest servlet.

CookieTest Servlet response to client indicates that initially there are no active cookies.

Same results would occur if you waited more than 15 minutes and then revisited the CookieTest servlet in a new browser session.

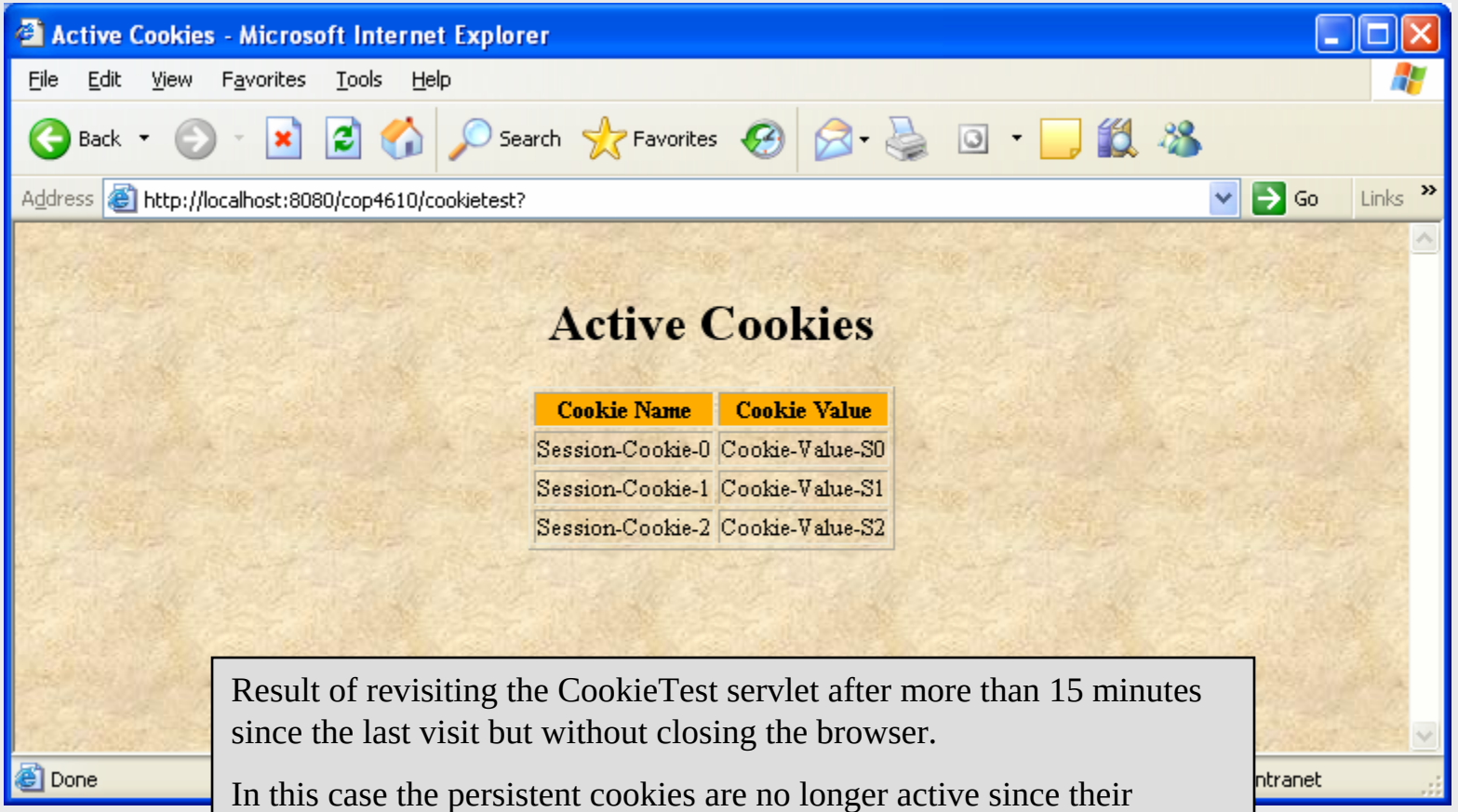


Results of revisiting the CookieTest servlet in the same browser session within 15 minutes of the very first visit.

After client runs the servlet the first time 3 persistent cookies and 3 session cookies are created. At this point there are six cookies active. Notice that the original cookie that was created in the last example (RepeatVisitor) is not active since its 1 minute lifetime has elapsed. If you set its lifetime to be longer or execute the CookieTest servlet within 1 minute of the RepeatVisitor cookie creation it will also appear in this list as well as the list from the previous page.

Results of revisiting the CookieTest servlet using a new browser session within 15 minutes of the first visit (in the earlier browser session).

Notice that the only cookies which are now active are the persistent cookies (the ones with the 15 minute lifetime).



Result of revisiting the CookieTest servlet after more than 15 minutes since the last visit but without closing the browser.

In this case the persistent cookies are no longer active since their maximum age has been exceeded. The session cookies are still active since the browser session has not terminated.

Modifying Cookie Values

- In the previous examples, we've sent a cookie to the user only on the first visit to the servlet. Once the cookie had a value, we never changed it.
- This approach of a single cookie value is quite common since cookies frequently contain nothing but unique user identifiers: all the real user data is stored in a database – the user identifier is merely the database key.
- But what if you would like to periodically change the value of a cookie?
- To replace a previous cookie value, send the same cookie name with a different cookie value. If you actually use the incoming `Cookie` objects, don't forget to do `response.addCookie`; merely calling `setValue` is not sufficient.

Modifying Cookie Values (cont.)

- You also need to reapply any relevant cookie attributes by calling `setMaxAge`, `setPath`, etc. – remember that cookie attributes are not specified for incoming cookies.
- Reapplying these attributes means that reusing the incoming `Cookie` object saves you very little, so many developers don't bother to use the incoming `Cookie` object.
- The next example illustrates modifying a cookie value by maintaining a count of the number of times your web browser visits a servlet named `ClientAccessCounts`.
- The code for ClientAccessCounts is shown on the next page with some results shown on the following pages.

ClientAccessCounts Servlet

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

  // Servlet that prints per-client access counts.

  public class ClientAccessCounts extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
      String countString =
        CookieUtilities.getCookieValue(request, "accessCount", "1");
      int count = 1;
      try {
        count = Integer.parseInt(countString);
      } catch(NumberFormatException nfe) { }
      Cookie c =
        new Cookie("accessCount",
                   String.valueOf(count+1));
      c.setMaxAge(900);
      response.addCookie(c);
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Access Count Servlet";
```

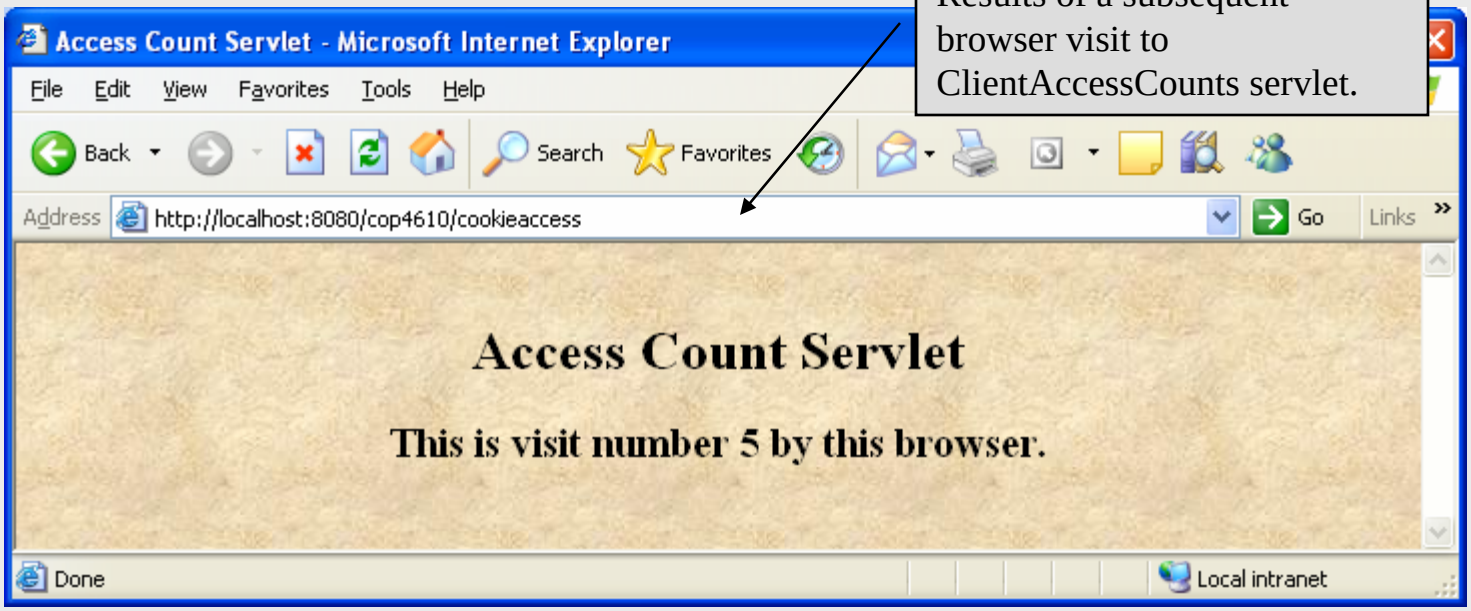
Read the current cookie value.

Create a new cookie with value 1 greater than current cookie value.

```
    String docType =
       "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
       "Transitional//EN\">\n";
     out.println ("<body bgcolor=white background=images/background.jpg
lang=EN-US link=blue vlink=blue >");
     out.println ("<body style='tab-interval:.5in'>");
     out.println ("<font size = 5>");
     out.println ("<br>");
     out.println(docType +
                 "<HTML>\n" +
                 "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                 "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                 "<CENTER>\n" +
                 "<H1>" + title + "</H1>\n" +
                 "<H2>This is visit number " +
                 count + " by this browser.</H2>\n" +
                 "</CENTER></BODY></HTML>");
  }
}
```


Results of initial browser visit to ClientAccessCounts servlet.

Results of a subsequent browser visit to ClientAccessCounts servlet.



Session Tracking

- As we mentioned before, 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 contextual information about the client.
- Even with servers that support persistent (keep-alive) HTTP connections and keep sockets open for multiple client requests that occur in rapid succession, there is no built-in support for maintaining contextual 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 of the previously created shopping carts are theirs?
- Servlets provide an outstanding session tracking solution: the `HttpSession` API. This high-level interface is built on top of cookies (and URL rewriting). All servers are required to support session tracking with cookies.

Session Tracking (cont.)

- Using sessions in servlets is straightforward and 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 hash table for storing user-specific data.
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.
3. Storing information in a session. Use `setAttribute` with a key and a 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.

Browser Sessions Vs. Server Sessions

- 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 `JSESSIONID` 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.
- The problem, however, is that the server does not know that the browser was closed and thus the server must maintain the session in memory until the inactive interval has been exceeded.
- To understand this problem consider the following scenario:

Browser Sessions Vs. Server Sessions

- Consider a physical shopping trip to your favorite store. You browse around and put some items into 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 they reshelve the items in it?
- No – you are probably still shopping and will come back for the cart soon.
- What if you realize that you left your wallet at home – so you go back home to get it. Can the clerk reshelve the items in the cart now?
- Again, no – the clerk presumably does not know that you have left the store.
- So, what can the clerk do? They can keep an eye on the cart, and if nobody claims the cart for some period of time, they can conclude that it is abandon and remove the items in it for reshelving.
- The only exception would be if you explicitly brought the cart to the clerk and told them that you left your wallet at home are have to leave.

Browser Sessions Vs. Server Sessions (cont.)

- The analogous situation in the servlet world is one in which the server is trying to decided if it can throw away your HttpSession object.
- Just because you are not currently using the session does not mean the server can throw it away. Maybe you will be back (submit a new request) soon.
- If you quit your browser, thus causing the browser-session-level cookies to be lost, the session is effectively broken. But as with the physical case of getting in your car and leaving, the server does not know that you quit your browser. So the server must still wait for a period of time to see if the session has been abandoned.
- 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.
- The one exception to the "server waits until the sessions time out" rule is if invalidate or logout is called. This is akin to your explicitly telling the clerk that you are leaving, so the server can immediately remove all the items from the session and destroy the session object.

A Servlet That Shows Per-Client Access Counts

- The last example in this section of notes is a 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 such an attribute, it uses the value of 0 as the number of previous accesses by the client. This value is then incremented and associated with the session by `setAttribute`.
- Finally, the servlet prints a small HTML table showing information about the session.
- Note that `Integer` is an immutable data structure: once built, it cannot be changed. That means that you have to allocate a new `Integer` object on each request, then use `setAttribute` to replace the old object.

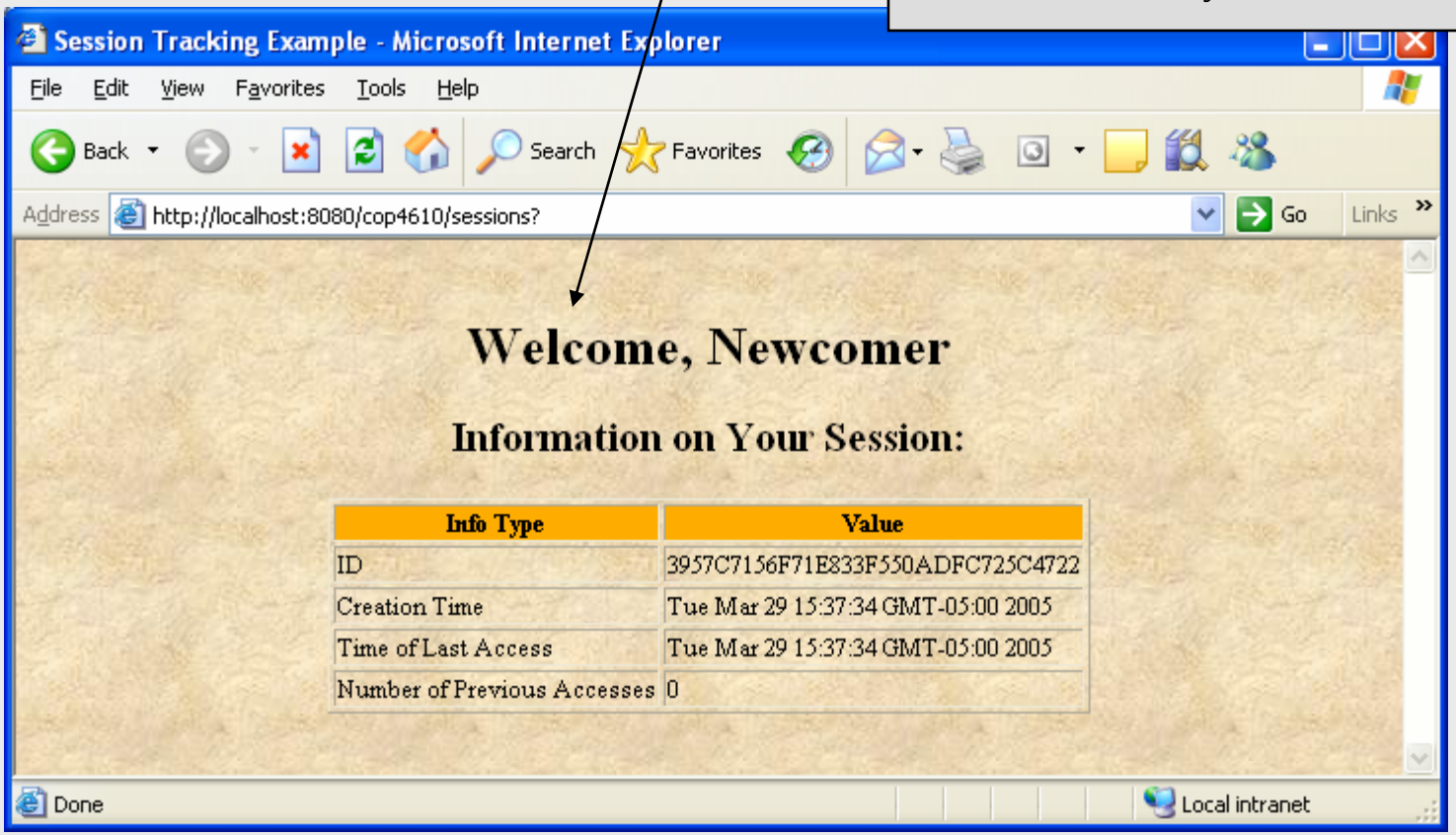
ShowSession Servlet

```
// Servlet that uses session-tracking to keep per-client
//  access counts. Also shows other info about the session.

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);
    }
    // Integer is an immutable data structure. So, you
    // cannot modify the old one in-place. Instead, you
    // have to allocate a new one and redo setAttribute.
```


ShowSession Servlet

```
        session.setAttribute("accessCount", accessCount);
        PrintWriter out = response.getWriter();
        String title = "Session Tracking Example";
        String docType =
          "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
          "Transitional//EN\">\n";
        out.println ("<body bgcolor=white background=images/background.jpg
    lang=EN-US link=blue vlink=blue >");
        out.println ("<body style='tab-interval:.5in'>");
        out.println ("<font size = 5>");
        out.println ("<br>");
        out.println(docType +
                    "<HTML>\n" +
                    "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                    "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                    "<CENTER>\n" + "<H1>" + heading + "</H1>\n" +
                    "<H2>Information on Your Session:</H2>\n" +
                    "<TABLE BORDER=1>\n" +"<TR BGCOLOR=\"#FFAD00\">\n" +
                    "  <TH>Info Type<TH>Value\n" + "<TR>\n" + "  <TD>ID\n" +
                    "  <TD>" + session.getId() + "\n" + "<TR>\n" +
                    "  <TD>Creation Time\n" + "  <TD>" +
                    new Date(session.getCreationTime()) + "\n" +
                    "<TR>\n" + "  <TD>Time of Last Access\n" +
                    "  <TD>" +
                    new Date(session.getLastAccessedTime()) + "\n" +
                    "<TR>\n" + "  <TD>Number of Previous Accesses\n" +
                    "  <TD>" + accessCount + "\n" + "</TABLE>\n" +
                    "</CENTER></BODY></HTML>");
    } }
```


Client makes their first visit to the ShowSession servlet. Since no session exists for this client, one is created. Some of the details about this session are shown by the servlet.



The client returns to the ShowSession servlet. Since HttpSession utilize cookies from the client's browser, this means that the user has not terminated the browser in-between visits to this servlet.