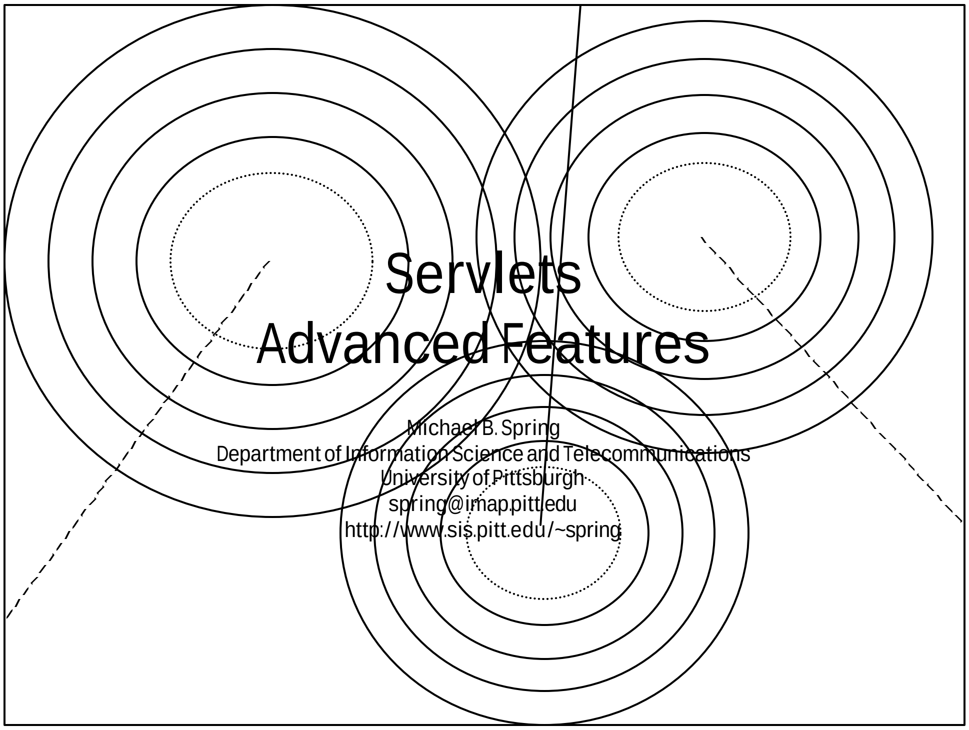
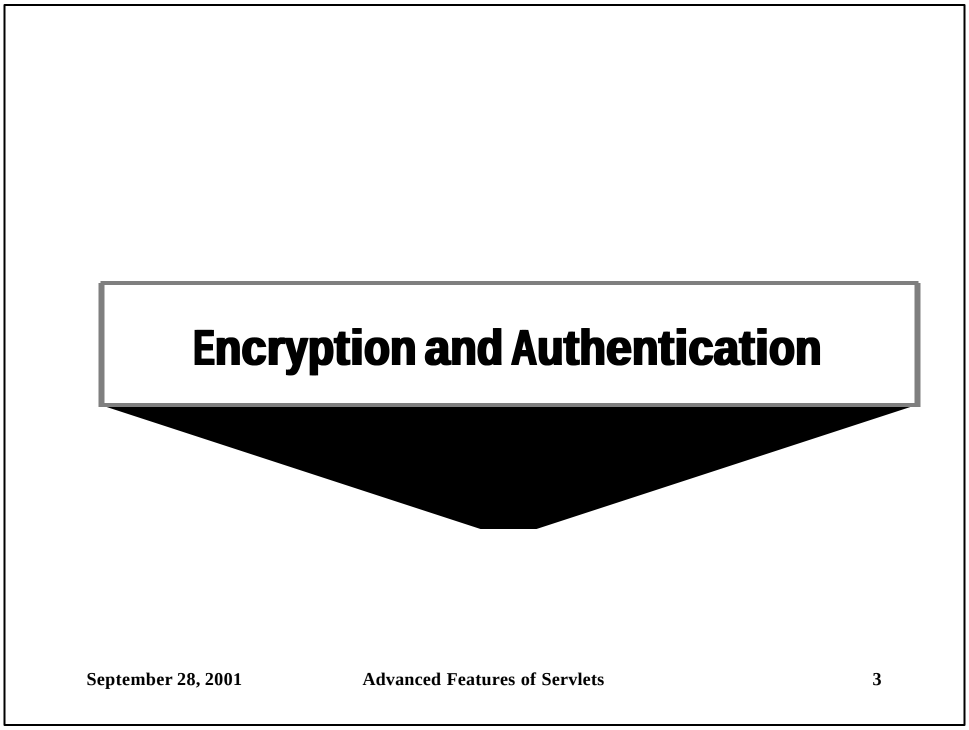


Servlets
Advanced Features

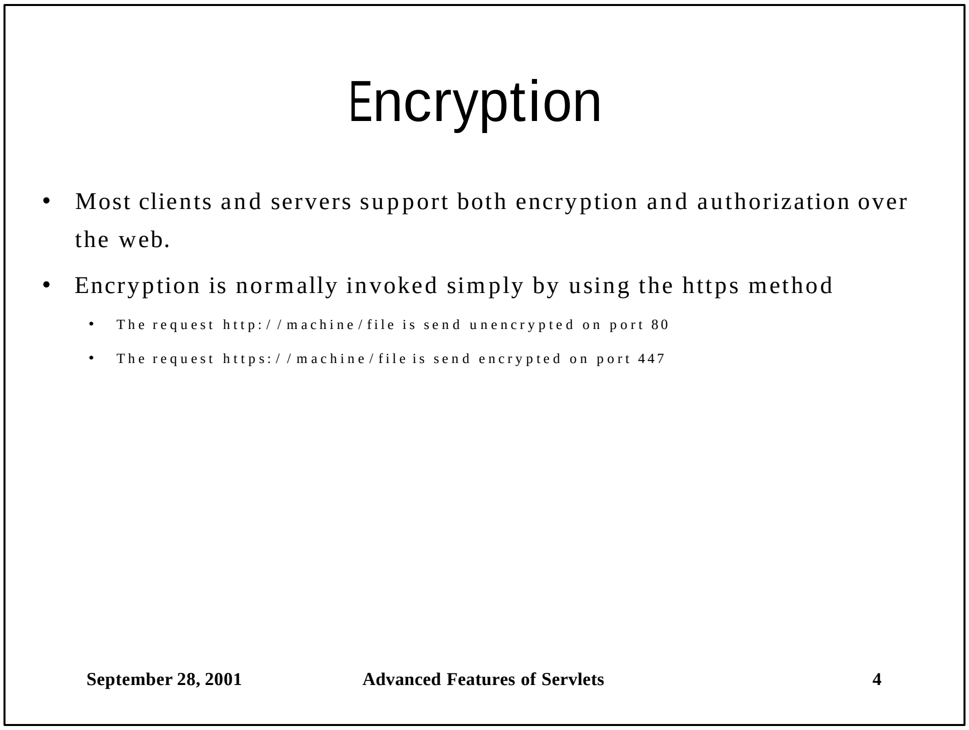
Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring

Overview

- Encryption and secure sockets
- Authentication and authorization
- Cookies
- State and session information



Encryption and Authentication



Encryption

- Most clients and servers support both encryption and authorization over the web.
- Encryption is normally invoked simply by using the https method
  - The request http://machine/file is send unencrypted on port 80
  - The request https://machine/file is send encrypted on port 447

Authentication

- Authentication is determined by server requirements to have an "Authorization" header prepended to messages
  - The client requests a document that is secured by server directive or by program constraint
  - The server sends a 401 response to the client
  - The client gathers the username and password, encrypts them, and sends them to the server along with the request. The information is sent in the authorization header
  - The server accepts or rejects the authorization information.
- The client includes the same authorization header in all subsequent calls to the same server

Cookies

Cookies

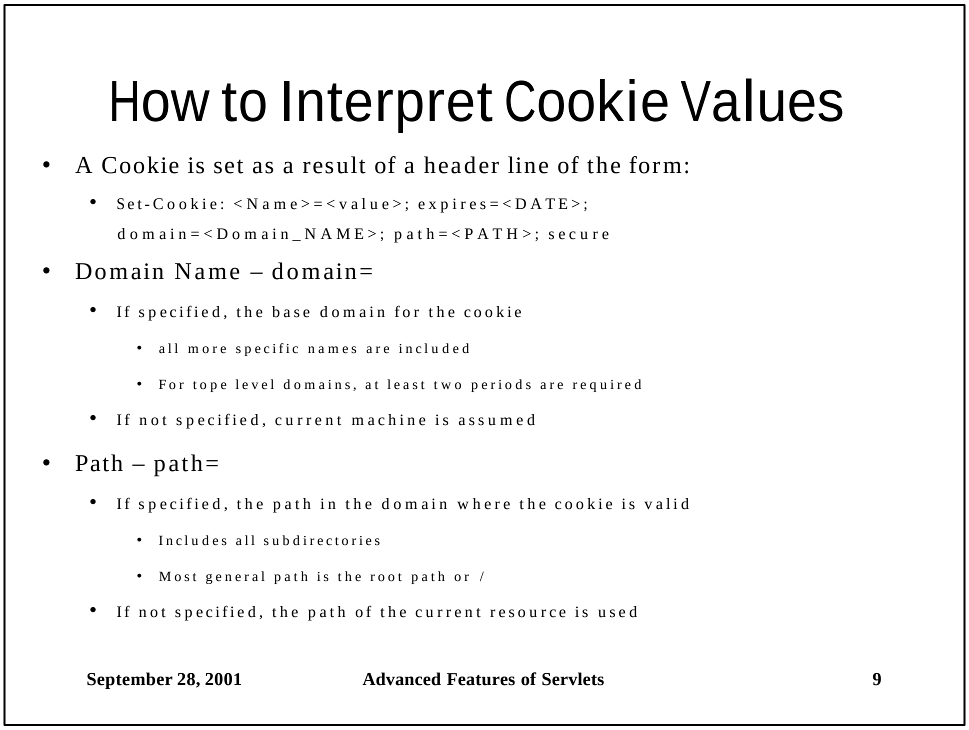
- Cookies have become a major mechanism for maintaining state information
- Prior to cookies, there was no standard way to maintain state information between sessions.
- By convention, most servers and clients have mechanisms that maintain state information within a session.
  - The authentication mechanism is one example
  - The client sends the authorization header again and again until either the client is shut down or it moves to another server.
- Cookies carry this one step further by providing a mechanism to save information between sessions.

What is a Cookie

- A cookie is information maintained in a file(s) on the client machine.
  - This means you might have different cookies for you desktop and laptop
  - There is currently no server side storage mechanisms
  - Lightweight Directory Access Protocol (LDAP) might provide this in the future
- A few examples of Cookies from netscape include:

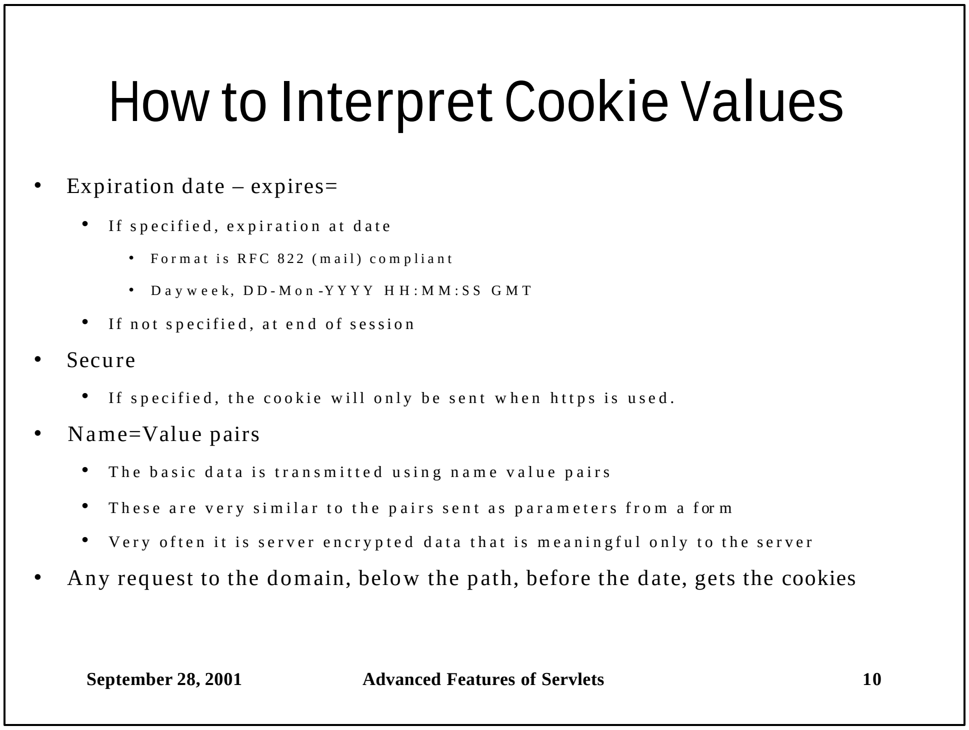
```
.netzip.com TRUE / FALSE 964670047 custid 108242
.dttus.com TRUE / FALSE 2051222109 SITESERVER ID=8345dc839e0b2af
.ap-adcenter.net TRUE / FALSE 2145801861 NGID a7d8dc18-732-93723-1
.tripod.com TRUE / FALSE 972087899 CookieStatus COOKIE_OK
.foxtrot.com TRUE / FALSE 1293839878 RMID 888e7444380f7580
```

- Very often, the value saved to the client is an ID to a DBMS entry on the server with more information



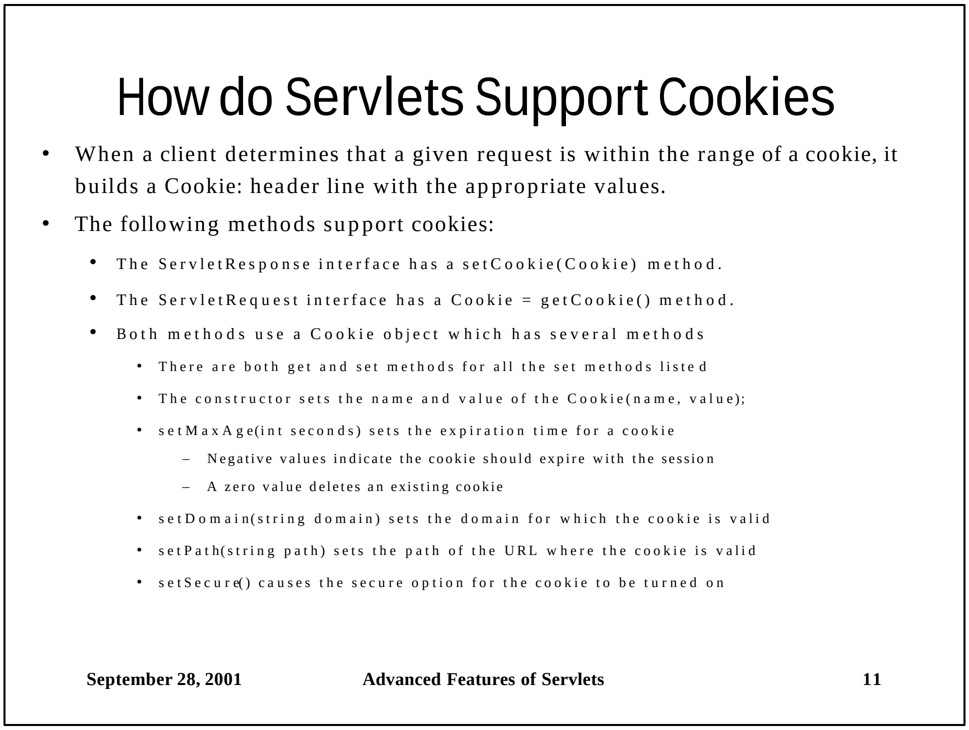
How to Interpret Cookie Values

- A Cookie is set as a result of a header line of the form:
  - Set-Cookie: <Name>=<value>; expires=<DATE>; domain=<Domain_NAME>; path=<PATH>; secure
- Domain Name – domain=
  - If specified, the base domain for the cookie
    - all more specific names are included
    - For tope level domains, at least two periods are required
  - If not specified, current machine is assumed
- Path – path=
  - If specified, the path in the domain where the cookie is valid
    - Includes all subdirectories
    - Most general path is the root path or /
  - If not specified, the path of the current resource is used



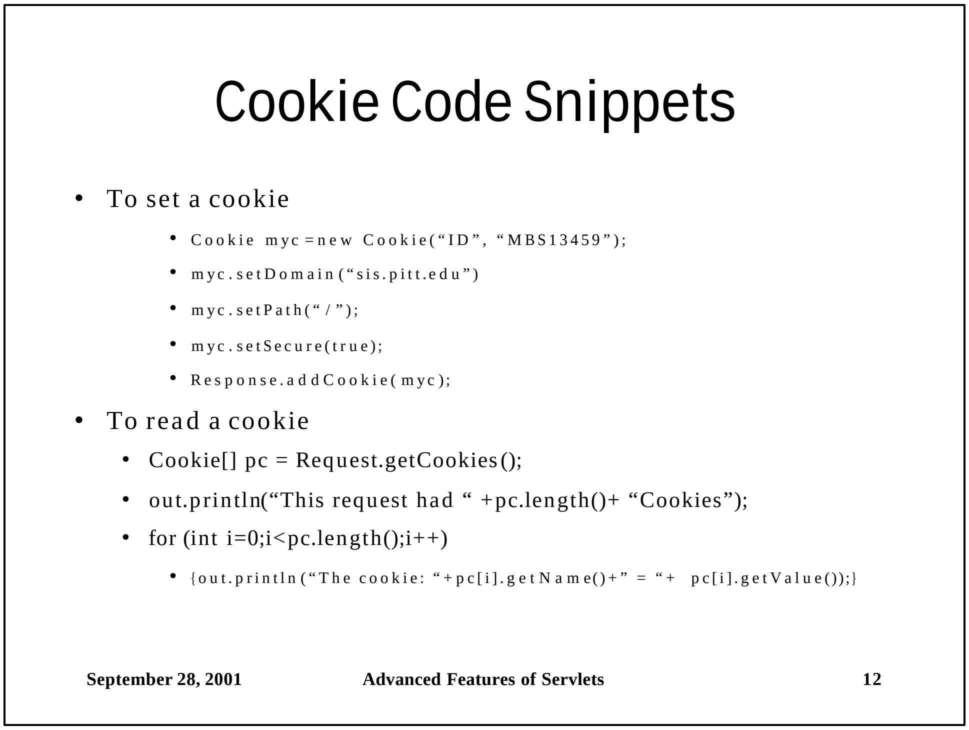
How to Interpret Cookie Values

- Expiration date – expires=
  - If specified, expiration at date
    - Format is RFC 822 (mail) compliant
    - Dayweek, DD-Mon-YYYY HH:MM:SS GMT
  - If not specified, at end of session
- Secure
  - If specified, the cookie will only be sent when https is used.
- Name=Value pairs
  - The basic data is transmitted using name value pairs
  - These are very similar to the pairs sent as parameters from a form
  - Very often it is server encrypted data that is meaningful only to the server
- Any request to the domain, below the path, before the date, gets the cookies



How do Servlets Support Cookies

- When a client determines that a given request is within the range of a cookie, it builds a Cookie: header line with the appropriate values.
- The following methods support cookies:
    - The ServletResponse interface has a setCookie(Cookie) method.
    - The ServletRequest interface has a Cookie = getCookie() method.
    - Both methods use a Cookie object which has several methods
        - There are both get and set methods for all the set methods listed
        - The constructor sets the name and value of the Cookie(name, value);
        - setMaxAge(int seconds) sets the expiration time for a cookie
            - – Negative values indicate the cookie should expire with the session
            - – A zero value deletes an existing cookie
        - setDomain(string domain) sets the domain for which the cookie is valid
        - setPath(string path) sets the path of the URL where the cookie is valid
        - setSecure() causes the secure option for the cookie to be turned on



Cookie Code Snippets

- To set a cookie
    - Cookie myc =new Cookie(“ID”, “MBS13459”);
    - myc.setDomain(“sis.pitt.edu”)
    - myc.setPath(“/”);
    - myc.setSecure(true);
    - Response.addCookie(myc);
- To read a cookie
    - Cookie[] pc = Request.getCookies();
    - out.println(“This request had “ +pc.length()+ “Cookies”);
    - for (int i=0;i<pc.length();i++)
        - {out.println(“The cookie: “+pc[i].getName()+” = “+ pc[i].getValue());}

State and Session Management

State and Session Management

- The http protocol is stateless
  - Each request to a server is independent of all other requests
  - This makes the server very robust
  - It limits the capability of the protocol to support complex operations
- There have been numerous efforts to provide support for state management
  - Hidden variables in forms and cookies are two examples of efforts to provide state information
- The definition of session is more elusive – in client server systems, frequently it is the length of time a connection is open between two partners
  - In the http world, physical sessions last only as long as the time it takes for a server to respond to a single request
  - We introduce the notion of virtual connections to overcome this and stipulate that a session is a set of connections which share some ID

State and Session Management Approach 1

- We can use the fact that CGI programs are recognized within a pathname to attach information to any URL.
- For example, given the following URL,
  - http://mysite/cgi/prog.cgi/hello/howareyou/xys.dat
  - Given that the server has CGI enabled and set correctly
  - If the file prog.cgi exists in the directory cgi and is executable
  - The string /hello/howareyou/xys.dat will be placed in the environment variable PATH_INFO
  - The program prog.cgi will be executed
- While this allows state information to be passed, it is very insecure and requires programmatic handling of all pages

State and Session Management Approach 2

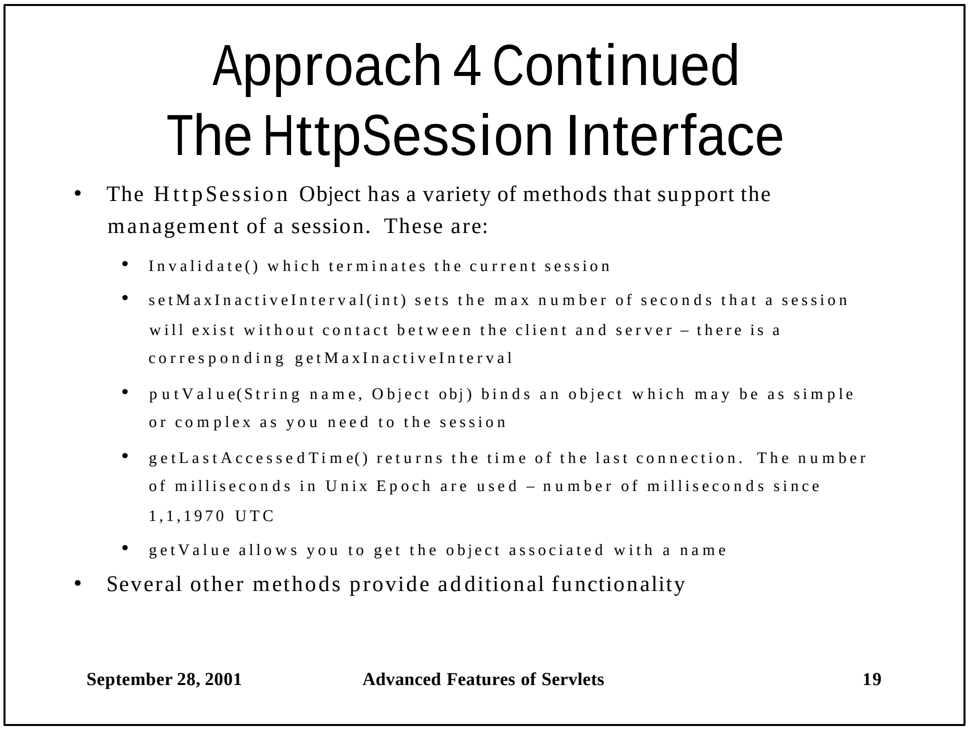
- The CGI mechanism can be used more directly to transfer data to server side program
- For standard URLs, the URL can be rewritten to include the state/session information
  - http://myserver/prog.cgi?name=value
- Unfortunately, this does not work for forms which construct the name value pairs based on the action.
  - In the case of forms, hidden variables can be used to cause the client to build the correct URL
  - <INPUT TYPE=“HIDDEN” NAME=“ID” VALUE=“1234”>
- Like approach 1, this approach suffers from the need to programmatically handle all pages and from the fact that data is visible to the curious

State and Session Management Approach 3

- The Cookie mechanism can be used
- As demonstrated, the process is very simple and will serve to meet most needs for state information
- Because it uses header info, it will thwart the casually curious and when used with https, it has good security.
- A minor problem with cookies is that not all browsers support cookies – although this is an increasingly small set
- A major problem with cookies is that they can be turned off by users for any of a number of good or bad reasons. Thus, you as a developer cannot be sure that state information is available.

State and Session Management Approach 4

- This approach uses a simplified cookie approach, with URL rewriting as a fallback, and allowing additional attributes to be stored on the server.
- An HttpSession object is provided that may be accessed via the HttpServletRequest:
  - HttpSession myses = request.getSession(true)
  - In this example, the boolean asks that the session be created if it doesn't exist
- Additional request methods allow the designer to determine the session id, whether it is valid, and how the information was stored on the client – cookies or rewritten url



Approach 4 Continued
The HttpSession Interface

- The HttpSession Object has a variety of methods that support the management of a session. These are:
  - Invalidate() which terminates the current session
  - setMaxInactiveInterval(int) sets the max number of seconds that a session will exist without contact between the client and server – there is a corresponding getMaxInactiveInterval
  - putValue(String name, Object obj) binds an object which may be as simple or complex as you need to the session
  - getLastAccessedTime() returns the time of the last connection. The number of milliseconds in Unix Epoch are used – number of milliseconds since 1,1,1970 UTC
  - getValue allows you to get the object associated with a name
- Several other methods provide additional functionality