Session Management

In this lecture, you will learn

- HTTP is a stateless protocol, and the implications of this statelessness

- Techniques you can use to manage user sessions

HTTP Protocol is Stateless

- This simply means that the Hyper Text Transfer Protocol that is the backbone of the Web is unable to retain a memory of the identity of each client that connects to a Web site and therefore treats each request for a Web page as a unique and independent connection, with no relationship whatsoever to the connections that preceded it.

- For viewing statically generated pages the stateless nature of the HTTP protocol is not usually a problem because the page you view will be the same no matter what previous operations you had performed.

- However for applications such as shopping carts which accumulate information as you shop it is extremely important to know what has happened previously, for example what you have in your basket. What is needed for these applications is a way to "maintain state" allowing connections to be tracked so that the application can respond to a request based on what has previously taken place.

To see the problem more clearly, consider the following LoginServlet.

☐ Assuming that the form uses the POST method, the user information is captured in the doPost method, which does the authentication by calling the login method.

- ☐ If the login is successful, the information is displayed.
- ☐ If not, the login form is displayed again

```
public class LoginServlet extends HttpServlet
{
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    {
        String userName = request.getParameter("userName");
        String password = request.getParameter("password");

        if (login(userName, password))
                //login successful, display the information
        else
                //login failed, resend the login form
    }
}
```

Discussion:

- What if you have another servlet that also only allows authorized users to view the information?

User Authorization

- The second servlet does not know whether the same user has successfully logged in to the first servlet.

- Consequently, the user will be required to login again.

Is this practical?

- This is, of course, is not practical. Every time a user goes to request a protected servlet, the user has to login again even though all the servlets are part of the same application

- Fortunately, there are ways to get around this, using techniques for remembering a user's session.

- Once users have logged in, they don't have to login again. The application will remember them.

- This is called **Session Management**.

- Session management goes beyond simply remembering a user who successfully logged in.

- Anything that makes the application remember information that has been entered or requested by the user can be considered session management.

- Session management does not change the nature of HTTP statelessness – it simply provides a way around it.

How do you manage a user's session?

- By performing the following to servlets that need to remember a user's state:

1. When the user requests a servlet, in addition to sending the response, the servlet also sends a token or an identifier.

2. If the user does not come back with the next request for the same or a different servlet, that is fine, the token/identifier is sent back to the server

3. Upon encountering the token, the next servlet should recognize the identifier and can do a certain action based on the token.

4. When the servlet responds to the request, it also sends the same or a different token.

5. This goes on and on with all the servlets that need to remember a user's session.

There are 4 techniques for session management.

☐ They operate based on the same principle, although what is passed and how it is passed is different from one to another.

☐ The techniques are as follows:

1. URL rewriting

2. Hidden fields

3. Cookies

4. Session objects

☐ Which technique you use depend on what you need to do in your application.

1. URL Rewriting

- With URL Rewriting, you append a token or identifier to the URL of the next servlet. You can send parameter name/value pairs using the following format:

  url?name1=value1&name2=value2&…

- When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server.

- From a servlet, you can use the HttpServletRequest interface's getParameter method to obtain a parameter value

- For example, to obtain the value of the second parameter, you write the following:

  request.getParameter(name2);

The use of URL rewriting is easy.

- When using this technique, however, you need to consider several thing:

  I. The number of characters that can be passed in a URL is limited. Typically, a browser can pass up to 2,000 characters

  II. The value that you pass can be seen in the URL. Sometimes this is not desirable. For example, some people prefer their password not to appear on the URL

  III. You need to encode certain characters – such as & and ? Characters and white spaces – that you append to a URL.

Example
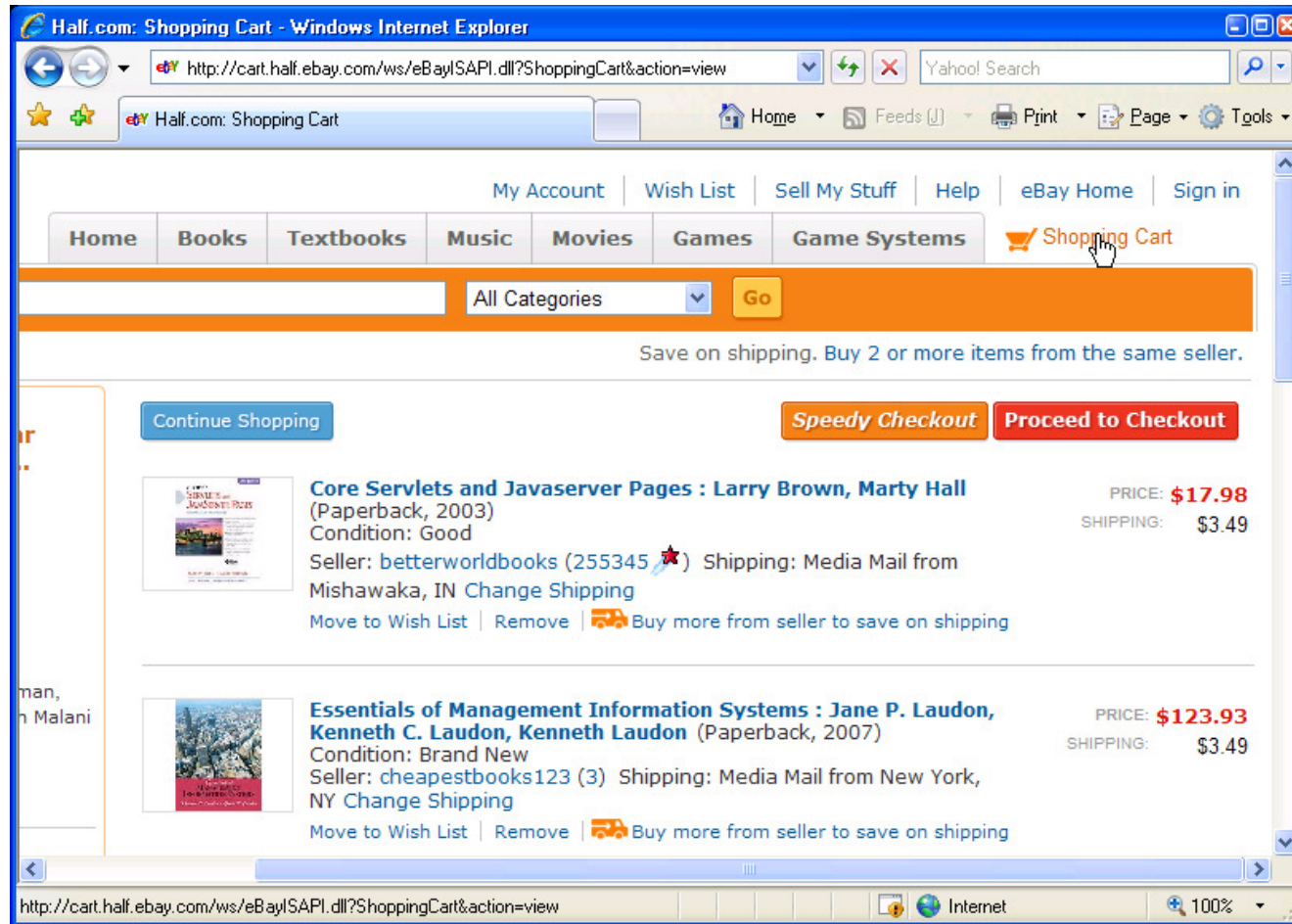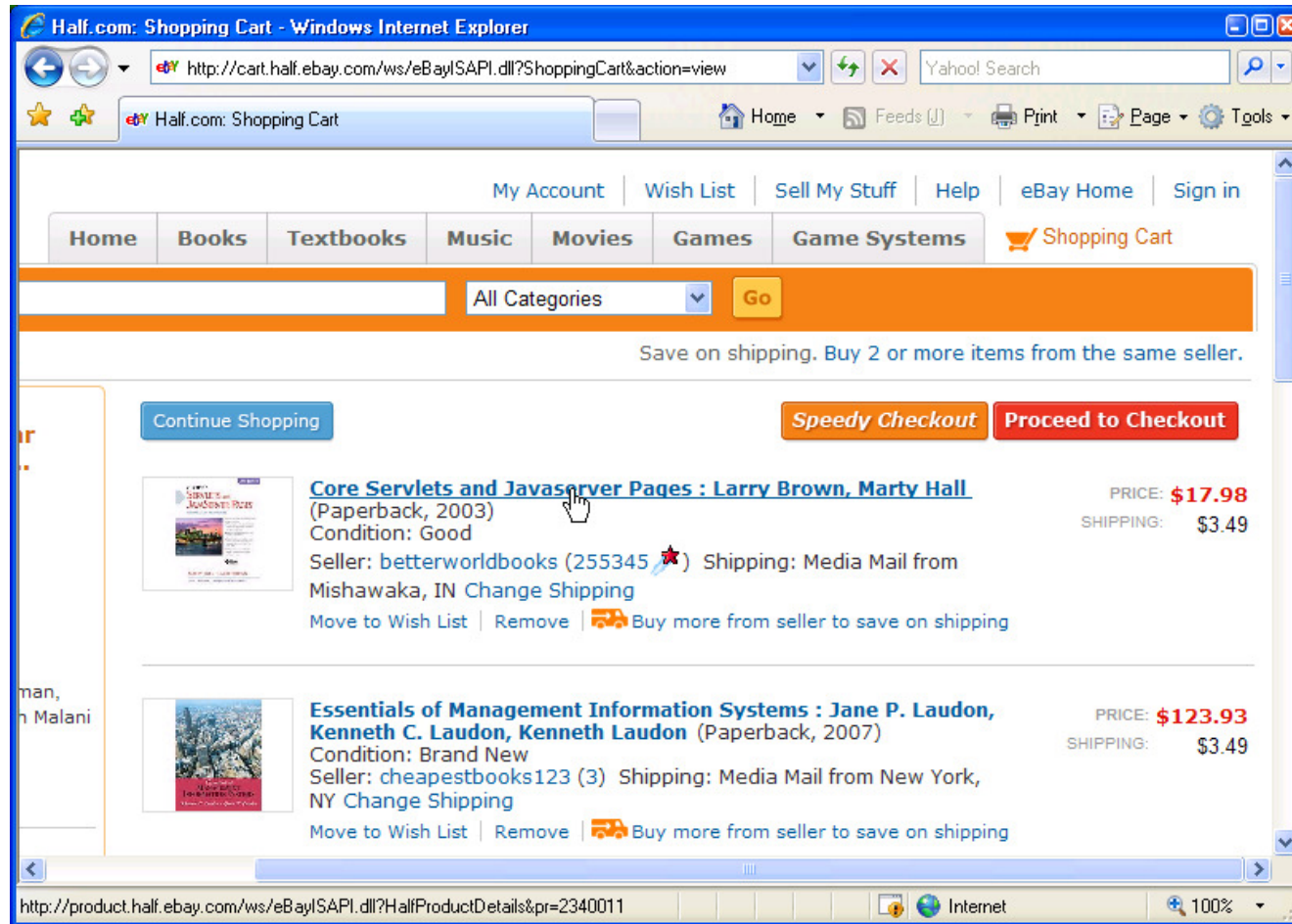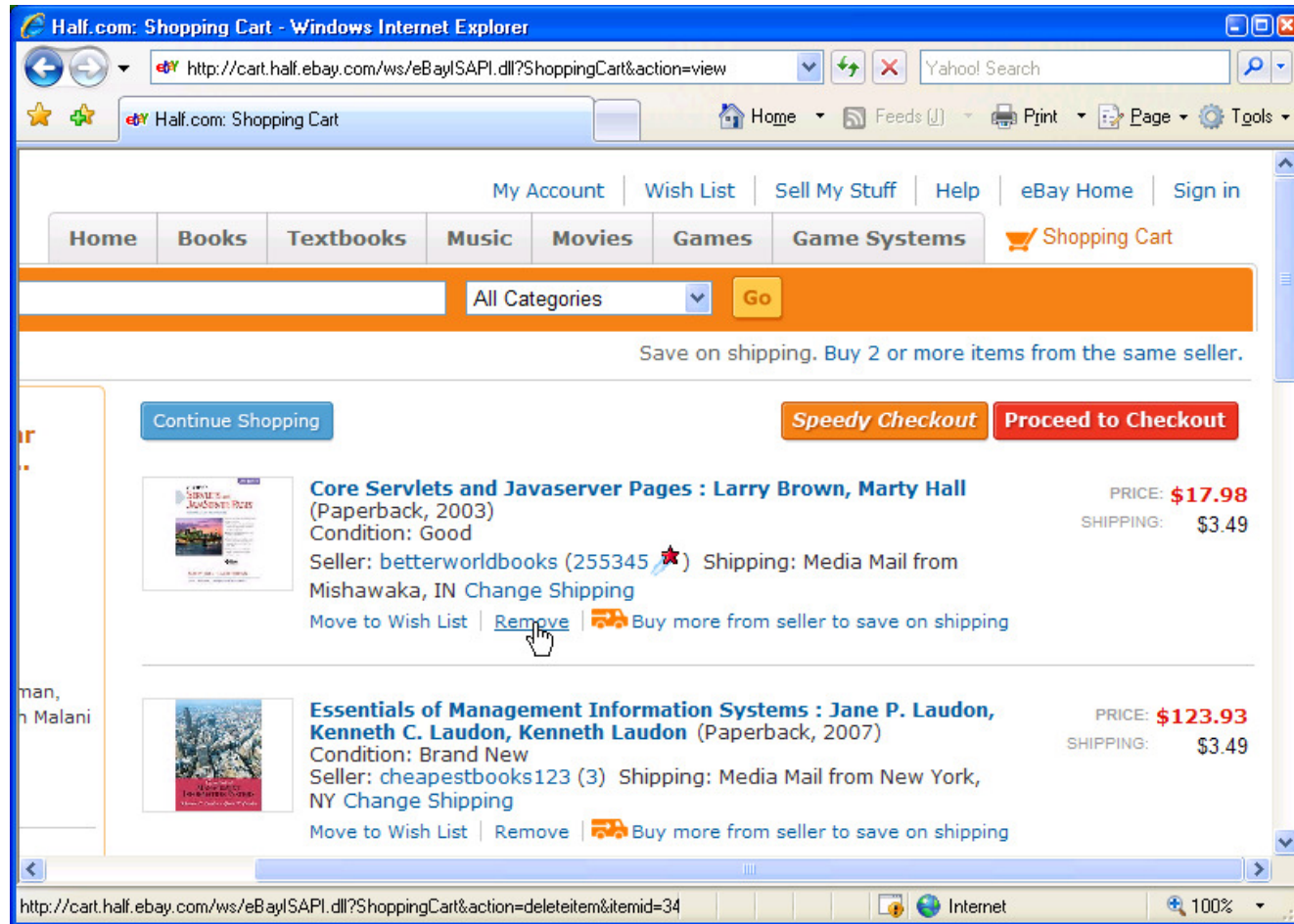
In-class exercise

- Use URL Rewriting in the following web application:
  - Download from Blackboard under Course Materials

2. Hidden Fields

□ Another technique for managing user sessions is by passing a token as the value for an HTML hidden field. Unlike the URL rewriting, the value does not show on the URL but can still be read by viewing the HTML source code.

□ HTML forms have an entry that looks like the following:

        `<INPUT TYPE="HIDDEN" NAME="session" VALUE="...">`

This means that, when the form is submitted, the specified name and value are included in the GET or POST data. This can be used to store information about the session.

Disadvantages:
• it only works if every page is dynamically generated, since the whole point is that each session has a unique identifier.
• an HTML form is always required.

Example

In-class exercise

- Use Hidden Fields in the following web application:Use
  - Download from Blackboard under Course Materials

Cookies

□ The third technique that you can use to manage user sessions is by using cookies.

□ A cookie is a small piece of information that is passed back and forth in the HTTP request and response.

□ Even though a cookie can be created on the client side using some scripting language such as JavaScript, it is usually created by a server resource, such as a servlet.

□ The cookie sent by a servlet to the client will be passed back to the server when the client requests another page from the same application.

Creating Cookies

□ In servlet programming, a cookie is represented by the Cookie class in the javax.servlet.http package. You can create a cookie by calling the Cookie class constructor and passing two String objects: the name and value of the cookie.

□ For instance, the following code creates a cookie object called c1. The cookie has the name "myCookie" and a value of "secret":

Cookie c1 = new Cookie("myCookie", "secret");

□ You then can add the cookie to the HTTP response using the addCookie method of the HttpServletResponse interface:

response.addCookie(c1);

□ Note that because cookies are carried in the request and response headers, you must not add a cookie after an output has been written to the HttpServletResponse object. Otherwise, an exception will be thrown.

Example

□ The following example shows how you can create two cookies called userName and password.

```
Cookie c1 = new Cookie("userName", "yusuf");
Cookie c2 = new Cookie("password", "secret");
response.addCookie(c1);
response.addCookie(c2);
```

□ To retrieve cookies, you use the getCookies method of the HttpServletRequest interface. This method returns a Cookie array containing all cookies in the request. It is your responsibility to loop through the array to get the cookie you want, as follows:

```
Cookie[] cookies = request.getCookies();
int length = cookies.length;

for (int i=0; i<length; i++)
{
  Cookie cookie = cookies[i];
  out.println("<B>Cookie Name:</B> " + cookie.getName() + "<BR>");
  out.println("<B>Cookie Value:</B> " + cookie.getValue() + "<BR>");
}
```

Disadvantage of using Cookies

☐ The user can choose not to accept them.

☐ Even though browsers leave the factories with the cookie setting on, any user can (accidentally) change this setting.

Session Objects

- Session object, represented by the javax.servlet.http.HttpSession interface, is the easiest to use and the most powerful.
- For each user, the servlet can create an HttpSession object that is associated with that user only and can only be accessed by that particular user.
- The HttpSession object acts like a Hashtable into which you can store any number of key/object pairs.
- The HttpSession object is accessible from other servlets in the same application.
- To retrieve an object previously stored, you need only to pass the key.
- Unlike previous techniques, however, the server does not send any value.
- What it sends is simply a unique number called the *session identifier*.
- This session identifier is used to associate a user with a Session object in the server.
- Therefore, if there are 10 simultaneous users, 10 Session objects will be created in the server and each user can access only his or her own HttpSession object.

How session tracking works with the HttpSession object.
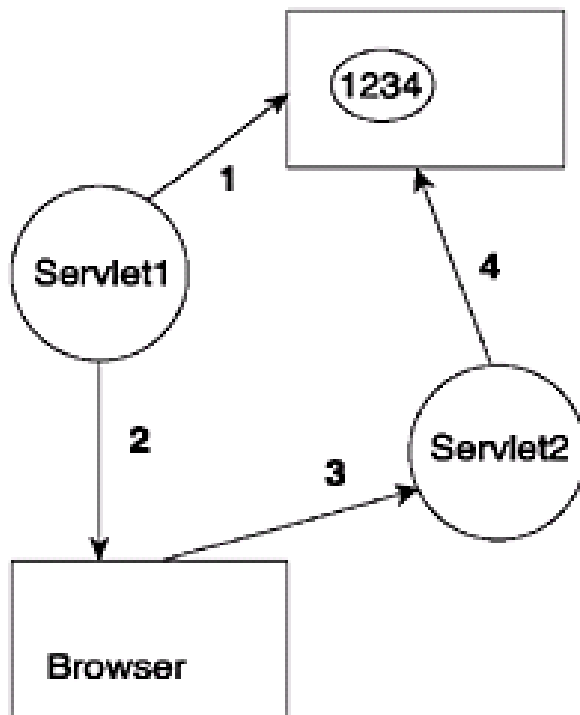
- An HttpSession object is created by a servlet called Servlet1. A session identifier is generated for this HttpSession object. In this example, the session identifier is 1234, but in reality, the servlet container will generate a longer random number that is guaranteed to be unique. The HttpSession object then is stored in the server and is associated with the generated session identifier. Also the programmer can store values immediately after creating an HttpSession.

- In the response, the servlet sends the session identifier to the client browser.

- When the client browser requests another resource in the same application, such as Servlet2, the session identifier is sent back to the server and passed to Servlet2 in the javax.servlet.http.HttpServletRequest object.

- For Servlet2 to have access to the HttpSession object for this particular client, it uses the getSession method of the javax.servlet.http.HttpServletRequest interface. This method automatically retrieves the session identifier from the request and obtains the HttpSession object associated with the session identifier.

Discussion

- What if the user never comes back after an HttpSession object is created?

- Then the servlet container waits for a certain period of time and removes that HttpSession object.

Problems?

- One worry about using Session objects is scalability.

- In some servlet containers, Session objects are stored in memory, and as the number of users exceeds a certain limit, the server eventually runs out of memory.

getSession method

- The getSession method of the javax.servlet.http.HttpServletRequest interface has two overloads. They are as follows:

    ```
    HttpSession getSession()
    HttpSession getSession(boolean create)
    ```

- The first overload returns the current session associated with this request, or if the request does not have a session identifier, it creates a new one.

- The second overload returns the HttpSession object associated with this request if there is a valid session identifier in the request. If no valid session identifier is found in the request, whether a new HttpSession object is created depends on the create value. If the value is true, a new HttpSession object is created if no valid session identifier is found in the request. Otherwise, the getSession method will return null.

The javax.servlet.http.HttpSession Interface

This interface has the following methods:

- getAttribute
- getAttributeNames
- getCreationTime
- getId
- getLastAccessedTime
- getMaxInactiveInterval
- getServletContext
- getSessionContext
- getValue
- getValueNames
- invalidate
- isNew
- putValue
- removeAttribute
- removeValue
- setAttribute
- setMaxInactiveInterval

In-class exercise

- Use Session Object in the following web application:
  - Download from Blackboard under Course Materials

Knowing Which Technique to Use

- Having learned the four techniques for managing user sessions, you may be wondering which one you should choose to implement.

Session Object

☐ Clearly, using Session objects is the easiest and you should use this if your servlet container supports swapping Session objects from memory to secondary storage.

☐ If you are using Tomcat 4 and later, this feature is available to you.

Cookies

- Using cookies is not as flexible as using Session objects.

- However, cookies are the way to go if you don't want your server to store any client-related information or if you want the client information to persist when the browser is closed.

Hidden Fields

- If you need to split a form into several smaller ones, however, using hidden fields is the cheapest and most efficient method.

- You don't need to consume server resources to temporarily store the values from the previous forms, and you don't need to rely on cookies.

- I would suggest hidden fields over Session objects, URL-rewriting, or cookies in this case.

URL Rewriting

I. The number of characters that can be passed in a URL is limited. Typically, a browser can pass up to 2,000 characters

II. The value that you pass can be seen in the URL. Sometimes this is not desirable. For example, some people prefer their password not to appear on the URL

III. You need to encode certain characters – such as & and ? Characters and white spaces – that you append to a URL.