Core Servlets and JavaServer Pages / 2e
Volume 1: Core Technologies
Marty Hall •Larry Brown

Session Tracking

Agenda

- Implementing session tracking from scratch
- Using basic session tracking
- Understanding the session-tracking API
- Differentiating between server and browser sessions
- Encoding URLs
- Storing immutable objects vs. storing mutable objects
- Tracking user access counts
- Accumulating user purchases
- Implementing a shopping cart
- Building an online store

Session Tracking

- **HTTP is a stateless protocol**
  - Each time a client retrieves a Web page, the client opens a separate connection to the Web server
  - The server does not automatically maintain contextual information about the client
  - When clients at on-line store add item to their shopping cart, how does server know what's already in cart?
  - When clients decide to proceed to checkout, how can server determine which previously created cart is theirs?

Rolling Your Own Session Tracking: Cookies

- **Use cookies to store an ID for a shopping session**
  - Each subsequent connection looks up the current session ID
  - Use the ID to extract information about that session from a lookup table on the server
- **Here’s how it could be done:**

```
String sessionID = makeUniqueString();
HashMap sessionInfo = new HashMap();
HashMap globalTable = findTableStoringSessions();
globalTable.put(sessionID, sessionInfo);
Cookie sessionCookie =
  new Cookie("JSESSIONID", sessionID);
sessionCookie.setPath("/");
response.addCookie(sessionCookie);
```

  - The server could use the **`globalTable`** hash table to associate a session ID from the **`JSESSIONID`** cookie with the **`sessionInfo`** hash table of user-specific data

Rolling Your Own Session Tracking: Cookies

- **This is a widely accepted solution**
- **Servlets have a higher-level API that handles all this plus the following:**
  - Extracting the cookie that stores the session identifier from the other cookies
  - Setting appropriate expiration time for cookie
  - Determining when idle sessions have expired and reclaiming them
  - Associating the hash tables with each request
  - Generating the unique session identifiers

Rolling Your Own Session Tracking: URL-Rewriting

- **Idea**
  - Client appends some extra data on the end of each URL that identifies the session
  - Server associates that identifier with data it has stored about that session
  - e.g., http://host/path/file.html;jsessionid=a1234
    - a1234 is the ID that uniquely identifies the table of data associated with that user
- **Advantage**
  - Works even if cookies are disabled or unsupported
- **Disadvantages**
  - Must encode all URLs that refer to your own site
  - All pages must be dynamically generated
  - Fails for bookmarks and links from other sites

Rolling Your Own Session Tracking: Hidden Form Fields

- **Idea:**

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

- **Advantage**
  - Works even if cookies are disabled or unsupported
- **Disadvantages**
  - Lots of tedious processing
  - All pages must be dynamically generated by a form submissions

Session Tracking Basics

- **Access the session object**
  - Call request.getSession to get HttpSession object
    - This is a hashtable associated with the user
- **Look 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.
- **Store information in a session.**
  - Use setAttribute with a key and a value.
- **Discard session data.**
  - Call removeAttribute discards a specific value.
  - Call invalidate to discard an entire session.

Accessing the Session Object of the Current Request

- **Session objects are of type `HttpServlet`, but they are basically hash tables that can store arbitrary user objects (each associated with a key)**

- **Look up the `HttpSession` object by calling the `getSession` method of `HttpServletRequest`**

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

  – Must be called before you send any document content to the client

- **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**
  – If no session ID is found in an incoming cookie or attached URL information, the system creates a new, empty session
  – If cookies are being used, the system creates an outgoing cookie named JSESSIOND with a unique value representing the session ID

Accessing the Session Object of the Current Request

- **Use `getSession()` or `getSession(true)` to add data to the session, regardless of whether data were there or not.**
  - It creates a new session if no session already exists
- **To just view a session, use `getSession(false)` which returns null if no session already exists for the current client**

```
HttpSession session = request.getSession(false);
if (session==null) printMessageSayingCartIsEmpty():
else extractCartAndPrintContents(session);
```

Looking Up Session Information

- **HttpSession objects live on the server, they don’t go back and forth over the network**
- **Sessions automatically associated with client via cookies or URL-rewriting**
  - Use request.getSession to get session
    - Behind the scenes, the system looks at cookie or URL extra info and sees if it matches the key to some previously stored session object. If so, it returns that object. If not, it creates a new one, assigns a cookie or URL info as its key, and returns that new session object.
- **Hashtable-like mechanism lets you store arbitrary objects inside session**
  - setAttribute stores values
  - getAttribute retrieves values

Information Associated with a Session

```
HttpSession session = request.getSession();
SomeClass value =
  (SomeClass)session.getAttribute("someID");
if (value == null) {
 value = new SomeClass(...);
 session.setAttribute("someID", value);
}
doSomethingWith(value);
```

- In most cases, you have a specific attribute name in mind and want to find the value already associated with that name.
- To discover all attribute names, use getAttributesNames which returns an Enumeration
- To specify information use setAttribute, it replaces any previous value

```
session.setAttribute("someID", value);
```

- removeAttribute removes a value without supplying a replacement

Information Associated with a Session

- **Discarding Session Data**
  - Remove only the data your servlet created using removeAttribute
  - Delete the whole session in the current a Web application using invalidate
    - All of the user's session data is lost, not just the session data that your servlet or JSP page created
  - Log the user out and delete all sessions belonging to the user using logout

HttpSession Methods

- **getAttribute**
  - Extracts a previously stored value from a session object. Returns null if no value is associated with given name.
- **setAttribute**
  - Associates a value with a name. he object supplied to setAttribute implemnets the HttpSessionBindingListener interface, the object’s valueBound method is called after it is stored in the session. If the previous value implements the listener, its valueUnbound method is called.
- **removeAttribute**
  - Removes values associated with name; invokes valueUnbound if listener is implemented
- **getAttributeNames**
  - Returns names of all attributes in the session.
- **getId**
  - Returns the unique identifier generated for each sessin
- **getCreationTime**
  - Returns time at which session was first created (pass to Date constructor)

HttpSession Methods

- **isNew**
  - Returns true if the *client* i.e., the browser (not the *page)* has never seen the session; returns false for preexisting sessions. Can be misleading – false value shows that the user has visited the Web application before but not that they visited your servlet or JSP page
- **getLastAccessedTime**
  - Returns time at which session was last sent from client
- **getMaxInactiveInterval, setMaxInactiveInterval**
  - Gets or sets the amount of time session should go without access before being invalidated. Negative value means session should never time out
- **invalidate**
  - Invalidates current session and unbinds all objects associated with it. Sessions are associated with clients not with individual servlets or JSP pages. Invalidating a session might destroy data that another servlet is using.

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**
- **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 server, however, does not know that the browser was closed and thus the server maintains the session in memory until the inactive interval has been exceeded**
- **Logout and invalidate tell the server that your session is completed**

A Servlet that Shows Per-Client Access Counts

- **This servlet shows basic information about a client's session**
  - When client connects, servlet uses **`request.getSession`** either to retrieve the existing session or create a new one (if there is no session)
  - Then it looks for an attribute called **`accessCount`** of type **`Integer`**
    - If not found, it uses 0 as the number of previous accesses
    - Because **`Integer`** is an immutable data structure, a new Integer is allocated on each request and **`setAttribute`** is used to replace the old object
  - This value is incremented and associated with the session by **`setAttribute`**
  - Finally, a small table of session information is printed

A Servlet that Shows Per-Client Access Counts

```
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";
      //immutable object, instantiate new one
      accessCount =
        new Integer(accessCount.intValue() + 1);
    }
    session.setAttribute("accessCount", accessCount);
```

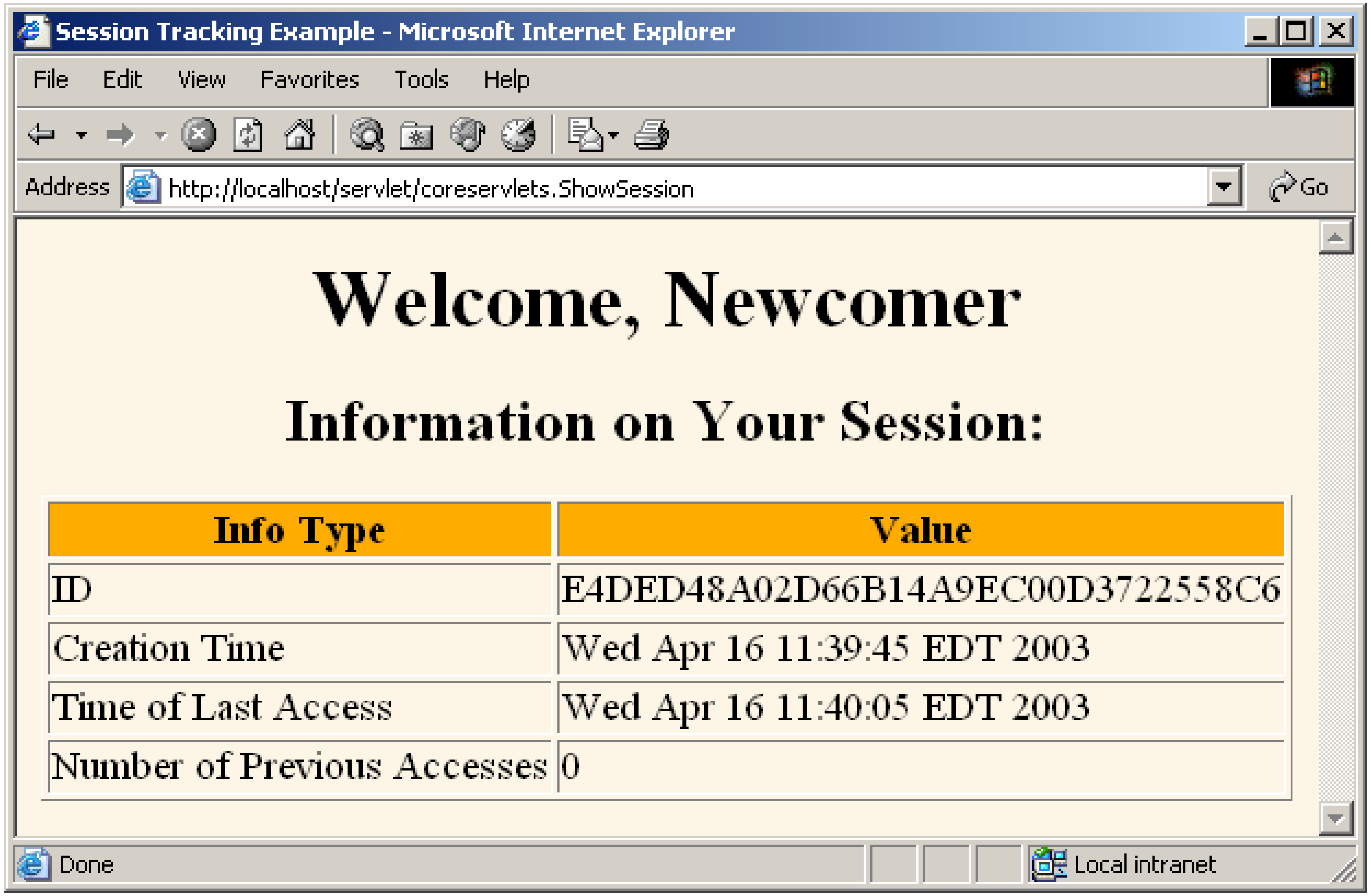
A Servlet that Shows Per-Client Access Counts

```
PrintWriter out = response.getWriter();
String title = "Session Tracking Example";
String docType =
"<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">\n";
 out.println(docType + "<HTML>\n" +
 "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
 "<BODY BGCOLOR=\"#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" +
```

A Servlet that Shows Per-Client Access Counts

```
    "<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>");
  }
}
```

A Servlet that Shows Per-Client Access Counts: Result 1

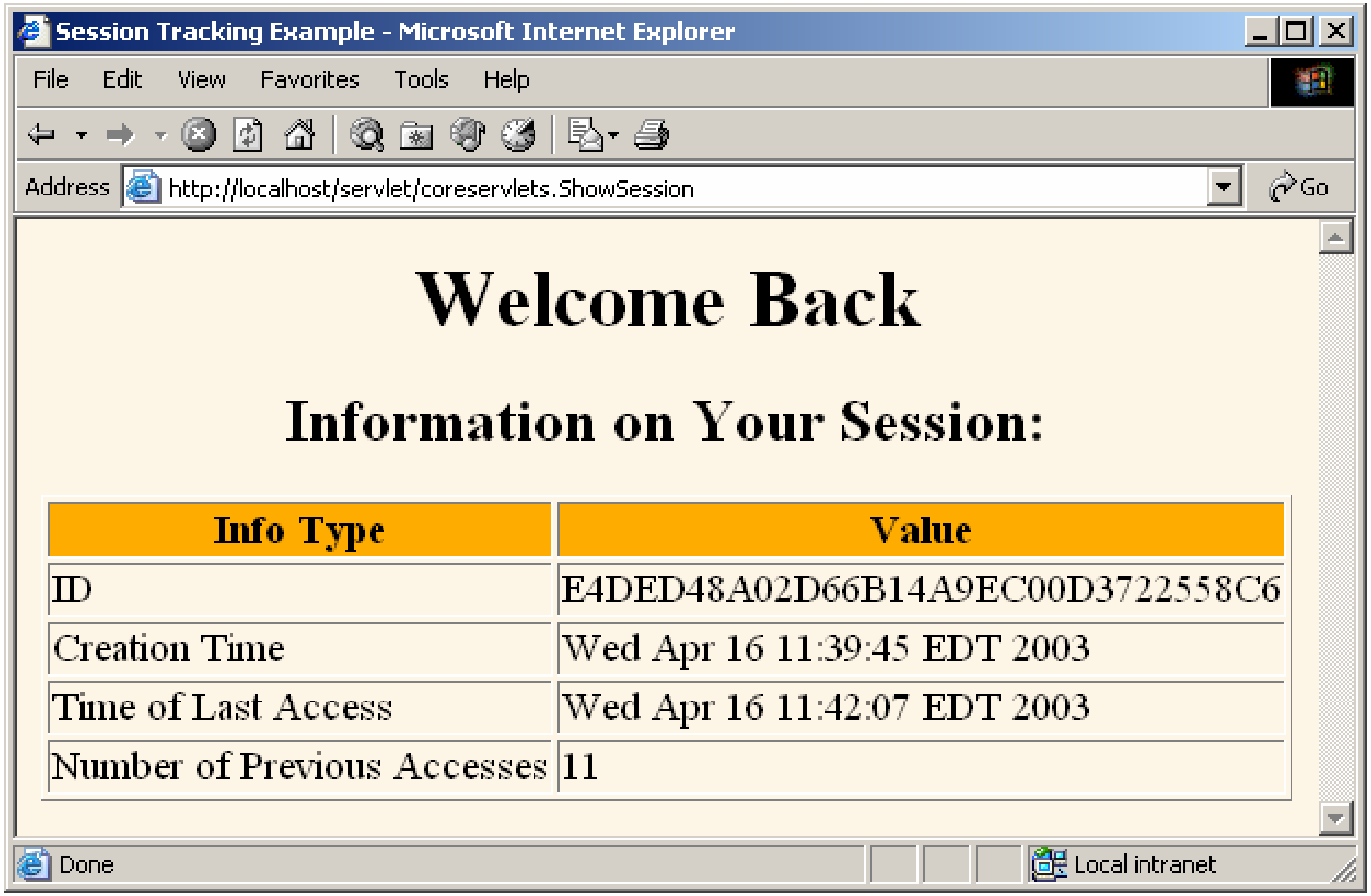

Session Tracking Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/servlet/coreservlets.ShowSession
Go
Done
Local intranet

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

A Servlet that Shows Per-Client Access Counts: Result 2


Session Tracking Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/servlet/coreservlets.ShowSession
Go
Done
Local intranet

Welcome Back

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 11:42:07 EDT 2003
Number of Previous Accesses	11

Accumulating a List of User Data

- **HttpSession data can also be stored in a mutable data structure (such as an array, List, etc.) or an application specific data structure that has writable instance variables**
  - **`setAttribute`** is called only when the object is first allocated

```
HttpSession session = request.getSession();
SomeMutableClass value =
   SomeMutableClass)session.getAttribute(“someIdentifier");
     if (value == null) {
       value = new SomeMutableClass(…);
       session.setAttribute(“someIdentifier", value);
     }
      value.updateInternalState(…);
       doSomethingWith(value);
```

Accumulating a List of User Data

- **The following example is a simplified version of a shopping cart**
  - A basic list of items that each user has purchased is maintained
  - An ArrayList is used to store the items purchased by each user
  - The servlet:
    - finds or creates the session
    - inserts the newly purchased item into the list (not saved as a cookie)
    - Outputs a bulleted list of the items in the “cart”
    - The code that outputs the ArrayList is synchronized even though the need for this is very rare (if the same user submits two purchases in rapid succession)

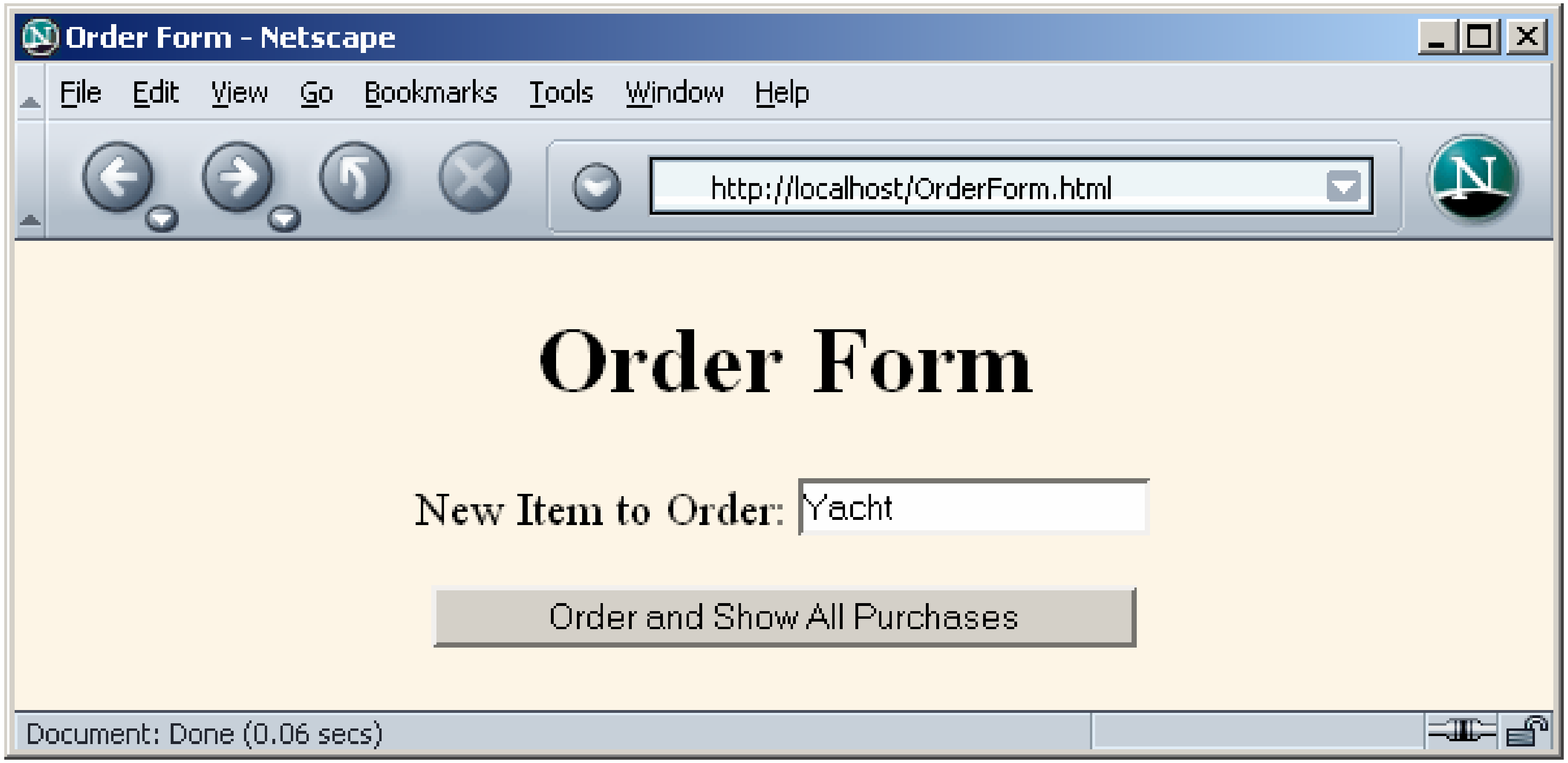
Accumulating a List of User Data

```
public class ShowItems extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    HttpSession session = request.getSession();
    ArrayList previousItems =
      (ArrayList)session.getAttribute("previousItems");
    if (previousItems == null) {
      previousItems = new ArrayList();
      session.setAttribute("previousItems",
                           previousItems);
    }
```

Accumulating a List of User Data (Continued)

```
  String newItem = request.getParameter("newItem");
  PrintWriter out = response.getWriter();
  …
  synchronized(previousItems) {
    if ((newItem != null) &&
        (!newItem.trim().equals(""))) {
      previousItems.add(newItem);
    }
    if (previousItems.size() == 0) {
      out.println("<I>No items</I>");
    } else {
      out.println("<UL>");
      for(int i=0; i<previousItems.size(); i++) {
        out.println("<LI>" +
                     (String)previousItems.get(i));
      }
      out.println("</UL>");
    }
  }
  out.println("</BODY></HTML>");
}
```

Accumulating a List of User Data: Front End


Order Form - Netscape
File Edit View Go Bookmarks Tools Window Help

http://localhost/OrderForm.html

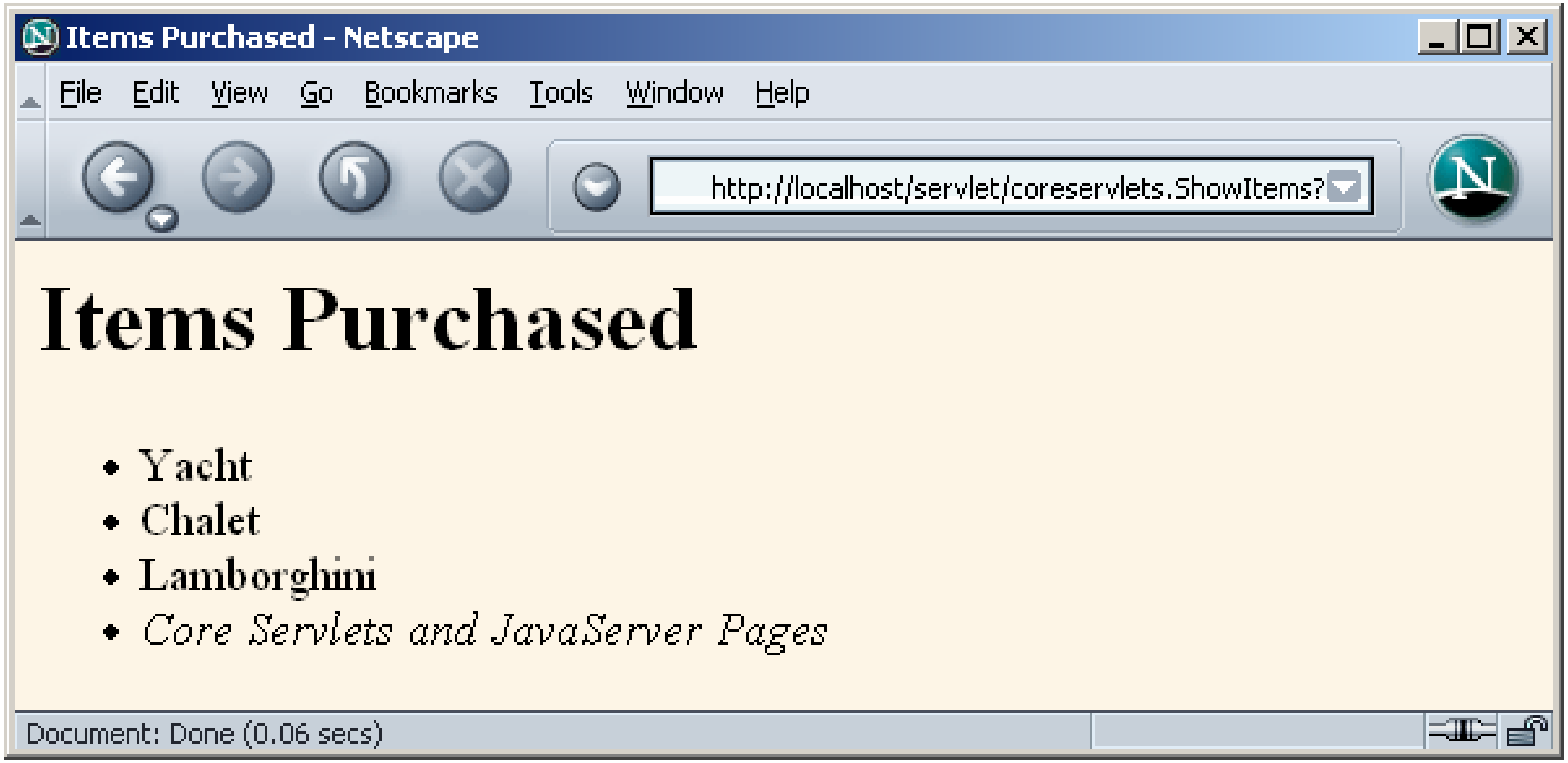
Order Form

New Item to Order: Yacht

Order and Show All Purchases

Document: Done (0.06 secs)

Accumulating a List of User Data: Result


Items Purchased - Netscape
File Edit View Go Bookmarks Tools Window Help
http://localhost/servlet/coreservlets.ShowItems?

Items Purchased

- Yacht
- Chalet
- Lamborghini
- *Core Servlets and JavaServer Pages*

Document: Done (0.06 secs)

An On-Line Bookstore

- **Session tracking code stays the same as in simple examples**
- **Shopping cart class is relatively complex**
  - Identifies items by a unique catalog ID
  - Does not repeat items in the cart
    - Instead, each entry has a count associated with it
    - If count reaches zero, item is deleted from cart
- **The first section of code describes the building pages that display items for sale**
  - The code for each display page lists the page title and the identifiers of the items listed on the page
  - Pages built automatically by methods in the parent class, based on the item description stored in the catalog

An On-Line Bookstore

- **The second section of code shows the page the handle the orders**
  - Order handling uses session tracking to associate a shopping cart with each user and permits the user to modify orders of previously selected items
- **The third section of code presents the implementation of the shopping cart, the data structures representing individual items and orders and the catalog**

CatalogPage.java

- Base class for pages showing catalog entries.

- Servlets that extend this class must specify the catalog entries that they are selling and the page title before the servlet is ever accessed.
-
- This is done by putting calls to setItems and setTitle in init.

CatalogPage.java

- setItems() method
  - **Given an array of item IDs, look them up in the Catalog and put their corresponding CatalogItem entry into the items array.**
  - **The CatalogItem contains a short description, a long description, and a price, using the item ID as the unique key.**
  - **Servlets that extend CatalogPage must call this method (usually from init) before the servlet is accessed.**

CatalogPage.java

```
public abstract class CatalogPage extends HttpServlet {
 private CatalogItem[] items;
 private String[] itemIDs;
 private String title;
 protected void setItems(String[] itemIDs) {
   this.itemIDs = itemIDs;
   items = new CatalogItem[itemIDs.length];
   for(int i=0; i<items.length; i++) {
      items[i] = Catalog.getItem(itemIDs[i]);
 }
}
```

CatalogPage.java

- doGet method
  - First displays the title
  - Then, for each catalog item, put its short description in a level-two (H2) heading with the price in parentheses and long description below.
  - Below each entry, put an order button that submits info to the OrderPage servlet for the associated catalog entry.

doGet method()

```
public void doGet(HttpServletRequest request,
   HttpServletResponse response)
    throws ServletException, IOException {
    if (items == null) {
      response.sendError(response.SC_NOT_FOUND, "Missing
      Items.");
       return;
    }
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML
      4.0 " + "Transitional//EN\">\n";
    out.println(docType +
    "<HTML>\n" +
    "<HEAD><TITLE>" + title +
    "</TITLE></HEAD>\n" +
    "<BODY BGCOLOR=\"#FDF5E6\">\n" +
    "<H1 ALIGN=\"CENTER\">" + title + "</H1>");
```

CatalogPage.java

```
    CatalogItem item;
    for(int i=0; i<items.length; i++) {
        out.println("<HR>");
        item = items[i];
        // Show error message if subclass lists item ID not in the
          catalog.
        if (item == null) {
         out.println("<FONT COLOR=\"RED\">“
         + "Unknown item ID " + itemIDs[i] + "</FONT>");
        }
        else {
        out.println();
        String formURL = "/servlet/coreservlets.OrderPage";
        // Pass URLs that reference own site through encodeURL.
        formURL = response.encodeURL(formURL);
        out.println ("<FORM ACTION=\"" + formURL + "\">\n" +
         "<INPUT TYPE=\"HIDDEN\" NAME=\"itemID\" " + " VALUE=\"" +
          item.getItemID() + "\">\n" +
          "<H2>" + item.getShortDescription() +
          " ($" + item.getCost() + ")</H2>\n" + item.getLongDescription()
          + "\n" +  "<P>\n<CENTER>\n" +
          "<INPUT TYPE=\"SUBMIT\" " +
             "VALUE=\"Add to Shopping Cart\">\n" +
          "</CENTER>\n <P>\n </FORM>");
        }
    }
    out.println("<HR>\n</BODY></HTML>");
 }
}
```

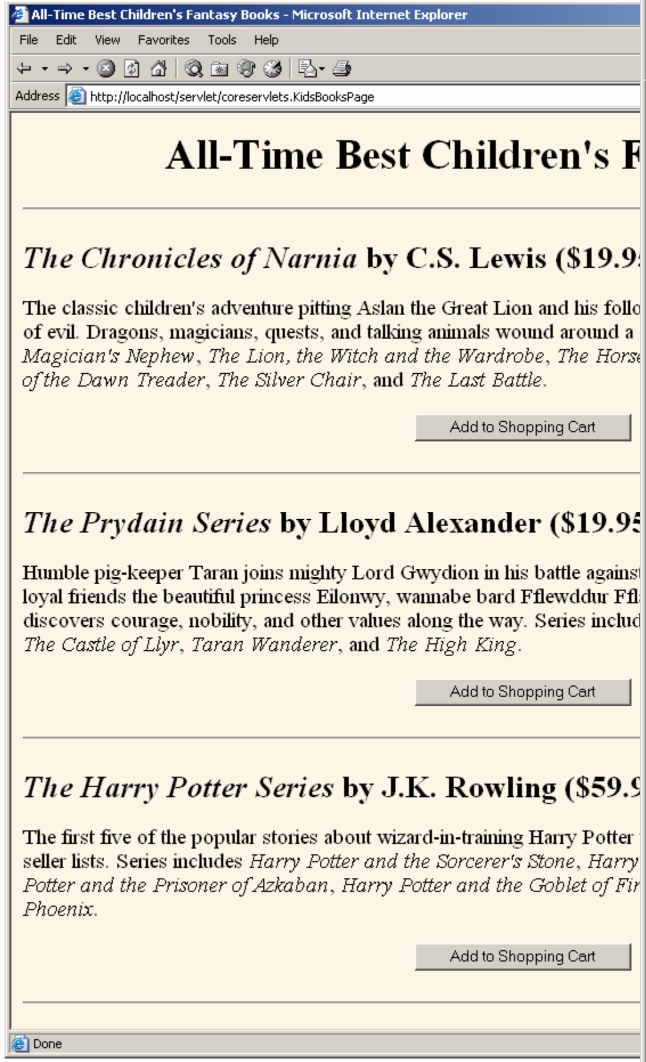
The catalogs

```
/** A specialization of the CatalogPage servlet that displays a
  page
   selling computer or children’s books.
   Orders are sent to the OrderPage servlet.
*/

 public class TechBooksPage extends CatalogPage {
  public void init()
  { String[] ids = { "hall001", "hall002" };
    setItems(ids);
    setTitle("All-Time Best Computer Books");
  }
 }

public class KidsBooksPage extends CatalogPage {
 public void init()
 { String[] ids = { "lewis001", "alexander001", "rowling001" };
   setItems(ids);
   setTitle("All-Time Best Children's Fantasy Books");
 }
}
```

An On-Line Bookstore


All-Time Best Children's Fantasy Books - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/servlet/coreservlets.KidsBooksPage
Add to Shopping Cart
Done

All-Time Best Children's F

The Chronicles of Narnia by C.S. Lewis ($19.9

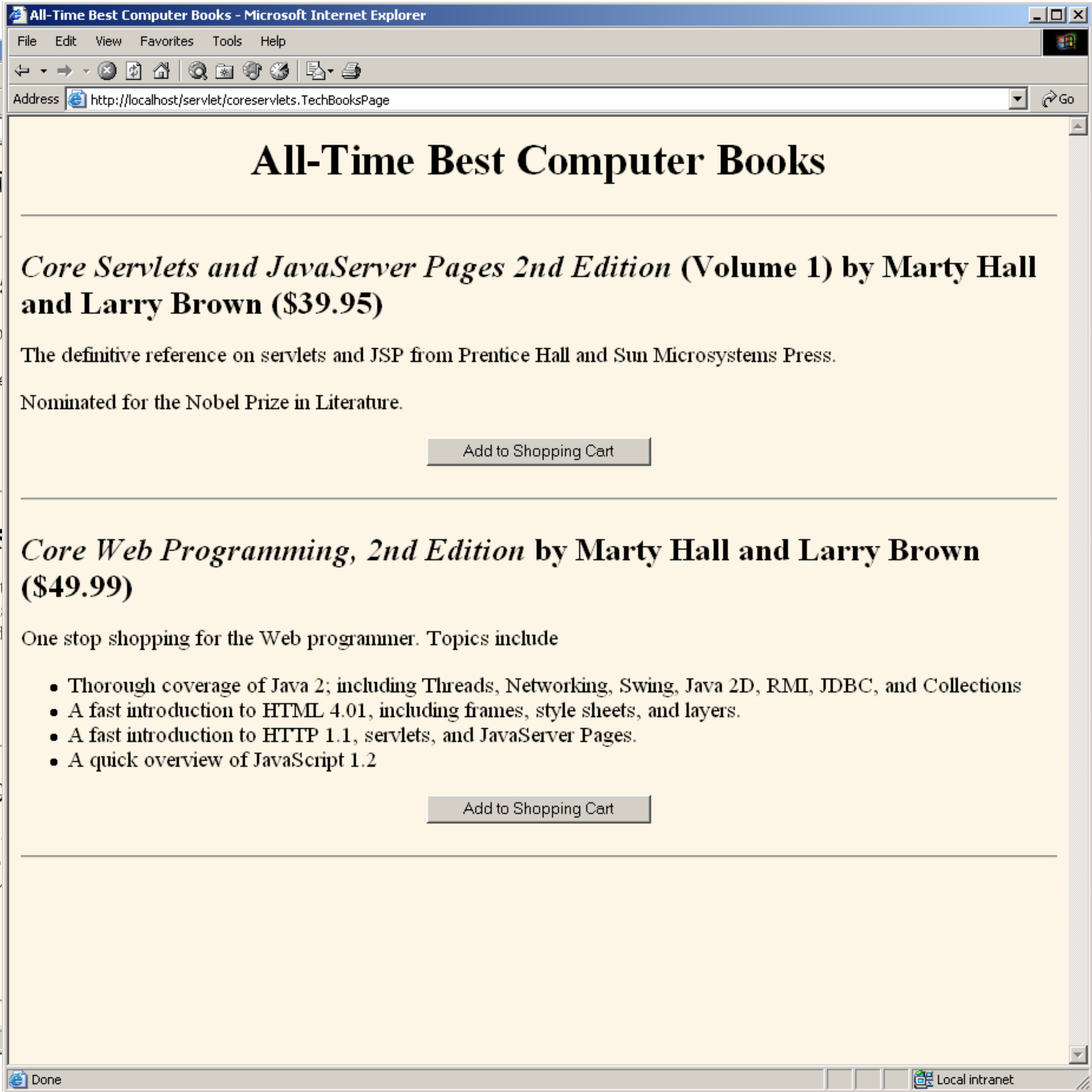
The classic children's adventure pitting Aslan the Great Lion and his follo of evil. Dragons, magicians, quests, and talking animals wound around a *Magician's Nephew*, *The Lion, the Witch and the Wardrobe*, *The Horse of the Dawn Treader*, *The Silver Chair*, and *The Last Battle*.

The Prydain Series by Lloyd Alexander ($19.95

Humble pig-keeper Taran joins mighty Lord Gwydion in his battle agains loyal friends the beautiful princess Eilonwy, wannabe bard Fflewddur Ffl discovers courage, nobility, and other values along the way. Series includ *The Castle of Llyr*, *Taran Wanderer*, and *The High King*.

The Harry Potter Series by J.K. Rowling ($59.9

The first five of the popular stories about wizard-in-training Harry Potter seller lists. Series includes *Harry Potter and the Sorcerer's Stone*, *Harry Potter and the Prisoner of Azkaban*, *Harry Potter and the Goblet of Fir Phoenix*.


All-Time Best Computer Books - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/servlet/coreservlets.TechBooksPage
Go
Add to Shopping Cart
Done
Local intranet

All-Time Best Computer Books

Core Servlets and JavaServer Pages 2nd Edition (Volume 1) by Marty Hall and Larry Brown ($39.95)

The definitive reference on servlets and JSP from Prentice Hall and Sun Microsystems Press.

Nominated for the Nobel Prize in Literature.

Core Web Programming, 2nd Edition by Marty Hall and Larry Brown ($49.99)

One stop shopping for the Web programmer. Topics include

- Thorough coverage of Java 2; including Threads, Networking, Swing, Java 2D, RMI, JDBC, and Collections
- A fast introduction to HTML 4.01, including frames, style sheets, and layers.
- A fast introduction to HTTP 1.1, servlets, and JavaServer Pages.
- A quick overview of JavaScript 1.2

Handling the Orders

- The servlet uses session tracking to associate a shopping cart with each user
  - **Multiple threads are not needed as each user has a separate session but, if a user has multiple browser windows open and sends updates from more than one window in quick succession a problem could arise**

Handling the Orders

```
public class OrderPage extends HttpServlet {

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    HttpSession session = request.getSession();
    ShoppingCart cart;

    synchronized(session) {
    cart =
  (ShoppingCart)session.getAttribute("shoppingCart");
      // New visitors get a fresh shopping cart.
      // Previous visitors keep using their existing cart.
      if (cart == null) {
        cart = new ShoppingCart();
        session.setAttribute("shoppingCart", cart);
      }
```

Handling the Orders

```
    String itemID = request.getParameter("itemID");
    if (itemID != null) {
      String numItemsString = request.getParameter("numItems");
      if (numItemsString == null) {
// If request specified an ID but no number, then got here
// via an "Add Item to Cart" button on a catalog page.
        cart.addItem(itemID);
      } else {
// If request specified an ID and number, then got here
// via an "Update Order" button after changing the number of
// items in order. A value of 0 results in item being deleted from cart.
        int numItems;
        try {
          numItems = Integer.parseInt(numItemsString);
        } catch(NumberFormatException nfe) {
          numItems = 1;
        }
        cart.setNumOrdered(itemID, numItems);
      }
    }
  }
```

Handling the Orders

```
 // Whether or not the customer changed the order, show order status.
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 String title = "Status of Your Order";
 String docType =
  "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
"Transitional//EN\">\n";
 out.println(docType +"<HTML>\n" +
   "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
   "<BODY BGCOLOR=\"#FDF5E6\">\n" +
   "<H1 ALIGN=\"CENTER\">" + title + "</H1>");
 synchronized(session) {
   List itemsOrdered = cart.getItemsOrdered();
   if (itemsOrdered.size() == 0) {
     out.println("<H2><I>No items in your cart...</I></H2>");
   } else {
     // If there is at least one item in cart, show table of items
ordered.
     out.println
       ("<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
        "<TR BGCOLOR=\"#FFAD00\">\n" +
        "  <TH>Item ID<TH>Description\n" +
        "  <TH>Unit Cost<TH>Number<TH>Total Cost");
```

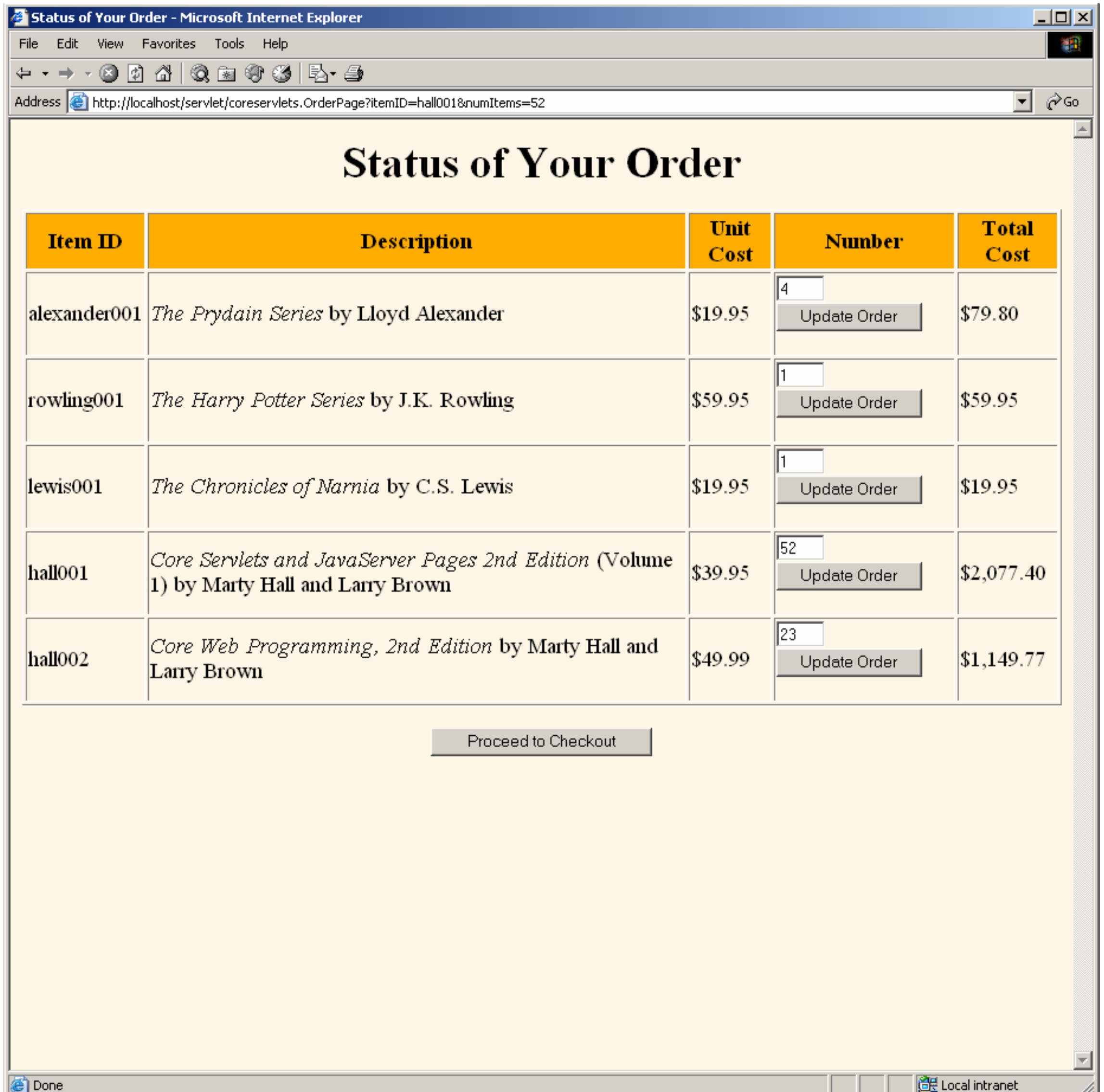
Handling the Orders

```
ItemOrder order;
NumberFormat formatter = NumberFormat.getCurrencyInstance();
    // For each entry in shopping cart, make table row showing ID,
    // description, per-item cost, number ordered, and total cost.
    // Put number ordered in textfield that user can change, with
    // "Update Order" button next to it, which resubmits to this same page
    // but specifying a different number of items.
    for(int i=0; i<itemsOrdered.size(); i++) {
      order = (ItemOrder)itemsOrdered.get(i);
      out.println ("<TR>\n" +
         "  <TD>" + order.getItemID() + "\n" +
         "  <TD>" + order.getShortDescription() + "\n" +
         "  <TD>" +
         formatter.format(order.getUnitCost()) + "\n" +
         "  <TD>" +
         "<FORM>\n" +  // Submit to current URL
         "<INPUT TYPE=\"HIDDEN\" NAME=\"itemID\"\n" +
         "       VALUE=\"" + order.getItemID() + "\">\n" +
         "<INPUT TYPE=\"TEXT\" NAME=\"numItems\"\n" +
         "       SIZE=3 VALUE=\"" + order.getNumItems() + "\">\n" +
         "<SMALL>\n" +
         "<INPUT TYPE=\"SUBMIT\"\n "+ " VALUE=\"Update Order\">\n" +
         "</SMALL>\n" + "</FORM>\n" + "  <TD>" +
         formatter.format(order.getTotalCost()));
    }
```

Handling the Orders

```
        String checkoutURL =
          response.encodeURL("../Checkout.html");
        // "Proceed to Checkout" button below table
        out.println
          ("</TABLE>\n" +
           "<FORM ACTION=\"" + checkoutURL + "\">\n" +
           "<BIG><CENTER>\n" +
           "<INPUT TYPE=\"SUBMIT\"\n" +
           "       VALUE=\"Proceed to Checkout\">\n" +
           "</CENTER></BIG></FORM>");
      }
      out.println("</BODY></HTML>");
    }
  }
}
```

An On-Line Bookstore


Status of Your Order - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/servlet/coreservlets.OrderPage?itemID=hall001&numItems=52
Go
Done
Local intranet

Status of Your Order

Item ID	Description	Unit Cost	Number	Total Cost
alexander001	*The Prydain Series* by Lloyd Alexander	$19.95	4 Update Order	$79.80
rowling001	*The Harry Potter Series* by J.K. Rowling	$59.95	1 Update Order	$59.95
lewis001	*The Chronicles of Narnia* by C.S. Lewis	$19.95	1 Update Order	$19.95
hall001	*Core Servlets and JavaServer Pages 2nd Edition* (Volume 1) by Marty Hall and Larry Brown	$39.95	52 Update Order	$2,077.40
hall002	*Core Web Programming, 2nd Edition* by Marty Hall and Larry Brown	$49.99	23 Update Order	$1,149.77

Proceed to Checkout

Shopping Cart Infrastructure

- **Shopping Cart**
  - ArrayList itemsOrdered of type ItemOrder
    - getItemsOrdered / addItem / setNumOrdered
- **CatalogItem**
  - String itemID
  - String shortDescription
  - String longDescription
  - double cost
    - get and set for each instance variable
- **ItemOrder**
  - CatalogItem item
  - int numItems
    - get and set for instance variables and i.v.s of CatalogItem
    - cancelOrder / getTotalCost / incrementNumItems
- **Catalog**
  - CatalogItem[] items

Distributed and Persistent Sessions

- **Some servers support distributed Web applications**
  - Load balancing used to send different requests to different machines
  - Session tracking still guaranteed to work
- **Some servers suport persistent sessions**
  - Session data written to disk and reloaded when server is restarted
- **To support both, session data should implement the java.io.Serializable interface**
  - There are no methods in this interface; it is just a flag.

What Changes if Server Uses URL Rewriting?

- **Session tracking code:**
  - No change
- **Code that generates hypertext links back to same site:**
  - Pass URL through response.encodeURL.
    - If server is using cookies, this returns URL unchanged
    - If server is using URL rewriting, this appends the session info to the URL
    - E.g.:
```
String url = "order-page.html";
url = response.encodeURL(url);
```
- **Code that does sendRedirect to own site:**
  - Pass URL through response.encodeRedirectURL