

9.7 Accumulating a List of User Data

The example of the previous section ([Section 9.6](#)) stores user-specific data in the user's `HttpSession` object. The object stored (an `Integer`) is an immutable data structure: one that cannot be modified. Consequently, a new `Integer` is allocated for each request, and that new object is stored in the session with `setAttribute`, overwriting the previous value.

Another common approach is to use a *mutable* data structure such as an array, `List`, `Map`, or application-specific data structure that has writable fields (instance variables). With this approach, you do not need to call `setAttribute` except when the object is first allocated. Here is the basic template:

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

Mutable data structures are most commonly used to maintain a set of data associated with the user. In this section we present a simplified example in which we maintain a basic list of items that each user has purchased. In the next section ([Section 9.8](#)), we present a full-fledged shopping cart example. Most of the code in that example is for automatically building the Web pages that display the items and for the shopping cart itself. Although these application-specific pieces can be somewhat complicated, the basic session tracking is quite simple. Even so, it is useful to see the fundamental approach without the distractions of the application-specific pieces. That's the purpose of the example here.

[Listing 9.2](#) shows an application that uses a simple `ArrayList` (the Java 2 platform's replacement for `Vector`) to keep track of the items each user has purchased. In addition to finding or creating the session and inserting the newly purchased item (the value of the `newItem` request parameter) into it, this example outputs a bulleted list of whatever items are in the "cart" (i.e., the `ArrayList`). Notice that the code that outputs this list is synchronized on the `ArrayList`. This precaution is worth taking, but you should be aware that the circumstances that make synchronization necessary are exceedingly rare. Since each user has a separate session, the only way a race condition could occur is if the same user submits two purchases in rapid succession. Although unlikely, this *is* possible, so synchronization is worthwhile.

Listing 9.2 ShowItems.java

```
package coreservlets;

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

/** Servlet that displays a list of items being ordered.
 *  * Accumulates them in an ArrayList with no attempt at
 *  * detecting repeated items. Used to demonstrate basic
 *  * session tracking.
 *  */

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);
    }
    String newItem = request.getParameter("newItem");
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Items Purchased";
    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>" + title + "</H1>");
    synchronized(previousItems) {
        if (newItem != null) {
            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>");
}

```

[Listing 9.3](#) shows an HTML form that collects values of the `newItem` parameter and submits them to the servlet. [Figure 9-3](#) shows the result of the form; [Figures 9-4](#) and [9-5](#) show the results of the servlet before the order form is visited and after it is visited several times, respectively.

Listing 9.3 OrderForm.html

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
    <TITLE>Order Form</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>Order Form</H1>
<FORM ACTION="servlet/coreservlets.ShowItems">
    New Item to Order:
    <INPUT TYPE="TEXT" NAME="newItem" VALUE="Yacht"><P>
    <INPUT TYPE="SUBMIT" VALUE="Order and Show All Purchases">
</FORM>
</CENTER></BODY></HTML>

```

Figure 9-3. Front end to the item display servlet.



Figure 9-4. The item display servlet before any purchases are made.

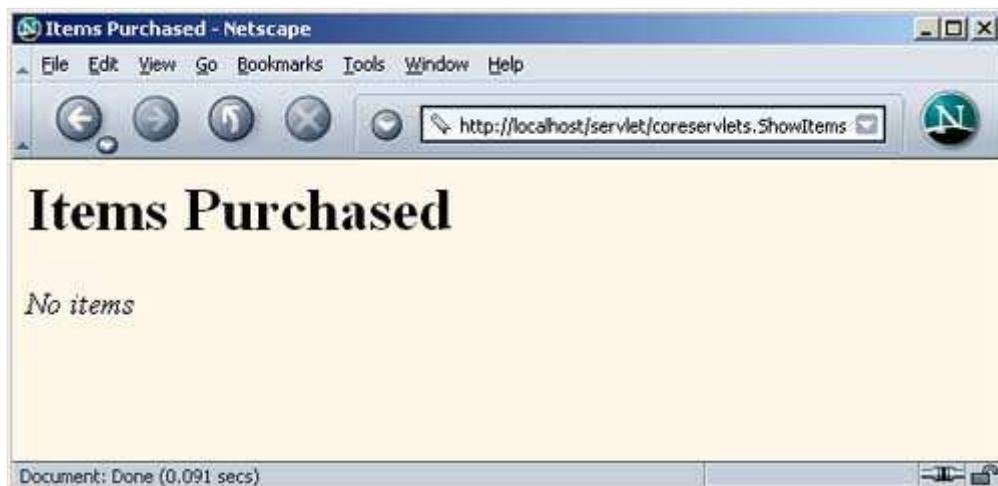


Figure 9-5. The item display servlet after a few valuable items are purchased.



[Team LiB]

◀ PREVIOUS NEXT ▶