1. What is Session Tracking?

There are a number of problems that arise from the fact that HTTP is a "stateless" protocol. In particular, when you are doing on-line shopping, it is a real annoyance that the Web server can't easily remember previous transactions. This makes applications like shopping carts very problematic: when you add an entry to your cart, how does the server know what's already in your cart? Even if servers did retain contextual information, you'd still have problems with e-commerce. When you move from the page where you specify what you want to buy (hosted on the regular Web server) to the page that takes your credit card number and shipping address (hosted on the secure server that uses SSL), how does the server remember what you were buying?

There are three typical solutions to this problem.

1. **Cookies.** You can use HTTP cookies to store information about a shopping session, and each subsequent connection can look up the current session and then extract information about that session from some location on the server machine. This is an excellent alternative, and is the most widely used approach. However, even though servlets have a high-level and easy-to-use interface to cookies, there are still a number of relatively tedious details that need to be handled:
    - Extracting the cookie that stores the session identifier from the other cookies (there may be many, after all),
    - Setting an appropriate expiration time for the cookie (sessions interrupted by 24 hours probably should be reset), and
    - Associating information on the server with the session identifier (there may be far too much information to actually store it in the cookie, plus sensitive data like credit card numbers should *never* go in cookies).
2. **URL Rewriting.** You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session. This is also an excellent solution, and even has the advantage that it works with browsers that don't support cookies or where the user has disabled cookies. However, it has most of the same problems as cookies, namely that the server-side program has a lot of straightforward but tedious processing to do. In addition, you have to be very careful that every URL returned to the user (even via indirect means like `Location` fields in server redirects) has the extra information appended. And, if the user leaves the session and comes back via a bookmark or link, the session information can be lost.
3. **Hidden form fields.** 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. However, it has the major disadvantage that it only works if every page is dynamically generated, since the whole point is that each session has a unique identifier.

Servlets provide an outstanding technical solution: the `HttpSession` API. This is a high-level interface built on top of cookies or URL-rewriting. In fact, on many servers, they use cookies if

the browser supports them, but automatically revert to URL-rewriting when cookies are unsupported or explicitly disabled. But the servlet author doesn't need to bother with many of the details, doesn't have to explicitly manipulate cookies or information appended to the URL, and is automatically given a convenient place to store data that is associated with each session.

2. The Session Tracking API

Using sessions in servlets is quite straightforward, and involves looking up the session object associated with the current request, creating a new session object when necessary, looking up information associated with a session, storing information in a session, and discarding completed or abandoned sessions.

2.1 Looking up the HttpSession object associated with the current request.

This is done by calling the `getSession` method of `HttpServletRequest`. If this returns null, you can create a new session, but this is so commonly done that there is an option to automatically create a new session if there isn't one already. Just pass `true` to `getSession`. Thus, your first step usually looks like this:

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

2.2 Looking up Information Associated with a Session.

`HttpSession` objects live on the server; they're just automatically associated with the requester by a behind-the-scenes mechanism like cookies or URL-rewriting. These session objects have a builtin data structure that let you store any number of keys and associated values. In version 2.1 and earlier of the servlet API, you use `getValue("key")` to look up a previously stored value. The return type is `Object`, so you have to do a typecast to whatever more specific type of data was associated with that key in the session. The return value is `null` if there is no such attribute. In version 2.2, `getValue` is deprecated in favor of `getAttribute`, both because of the better naming match with `setAttribute` (the match for `getValue` is `putValue`, not `setValue`), and because `setAttribute` lets you use an attached HttpSessionBindingListener to monitor values, while `putValue` doesn't. Nevertheless, since few commercial servlet engines yet support version 2.2, I'll use `getValue` in my examples. Here's one representative example, assuming `ShoppingCart` is some class you've defined yourself that stores information on items being purchased.

```
HttpSession session = request.getSession(true);
ShoppingCart previousItems =
  (ShoppingCart)session.getValue("previousItems");
if (previousItems != null) {
  doSomethingWith(previousItems);
} else {
  previousItems = new ShoppingCart(...);
  doSomethingElseWith(previousItems);
}
```

In most cases, you have a specific attribute name in mind, and want to find the value (if any) already associated with it. However, you can also discover all the attribute names in a given session by calling `getValueNames`, which returns a `String` array. In version 2.2, use

`getAttributeNames`, which has a better name and which is more consistent in that it returns an `Enumeration`, just like the `getHeaders` and `getParameterNames` methods of `HttpServletRequest`.

Although the data that was explicitly associated with a session is the part you care most about, there are some other pieces of information that are sometimes useful as well.

- **getId.** This method returns the unique identifier generated for each session. It is sometimes used as the key name when there is only a single value associated with a session, or when logging information about previous sessions.
- **isNew.** This returns `true` if the client (browser) has never seen the session, usually because it was just created rather than being referenced by an incoming client request. It returns `false` for preexisting sessions.
- **getCreationTime.** This returns the time, in milliseconds since the epoch, at which the session was made. To get a value useful for printing out, pass the value to the `Date` constructor or the `setTimeInMillis` method of `GregorianCalendar`.
- **getLastAccessedTime.** This returns the time, in milliseconds since the epoch, at which the session was last sent from the client.
- **getMaxInactiveInterval.** This returns the amount of time, in seconds, that a session should go without access before being automatically invalidated. A negative value indicates that the session should never timeout.

2.3 Associating Information with a Session

As discussed in the previous section, you read information associated with a session by using `getValue` (or `getAttribute` in version 2.2 of the servlet spec). To specify information, you use `putValue` (or `setAttribute` in version 2.2), supplying a key and a value. Note that `putValue` replaces any previous values. Sometimes that's what you want (as with the `referringPage` entry in the example below), but other times you want to retrieve a previous value and augment it (as with the `previousItems` entry below). Here's an example:

```
HttpSession session = request.getSession(true);
session.putValue("referringPage", request.getHeader("Referer"));
ShoppingCart previousItems =
  (ShoppingCart)session.getValue("previousItems");
if (previousItems == null) {
  previousItems = new ShoppingCart(...);
}
String itemID = request.getParameter("itemID");
previousItems.addEntry(Catalog.getEntry(itemID));
// You still have to do putValue, not just modify the cart, since
// the cart may be new and thus not already stored in the session.
session.putValue("previousItems", previousItems);
```

3. Example: Showing Session Information

Here is a simple example that generates a Web page showing some information about the current session. You can also download the source or try it on-line.

```
package hall;
```

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

/** Simple example of session tracking. See the shopping
 *  cart example for a more detailed one.
 *  <P>
 *  Part of tutorial on servlets and JSP that appears at
 *  http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/
 *  1999 Marty Hall; may be freely used or adapted.
 */

public class ShowSession extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    HttpSession session = request.getSession(true);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Searching the Web";
    String heading;
    Integer accessCount = new Integer(0);;
    if (session.isNew()) {
      heading = "Welcome, Newcomer";
    } else {
      heading = "Welcome Back";
      Integer oldAccessCount =
        // Use getAttribute, not getValue, in version
        // 2.2 of servlet API.
        (Integer)session.getValue("accessCount");
      if (oldAccessCount != null) {
        accessCount =
          new Integer(oldAccessCount.intValue() + 1);
      }
    }
    // Use putAttribute in version 2.2 of servlet API.
    session.putValue("accessCount", accessCount);

    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + heading + "</H1>\n" +
                "<H2>Information on Your Session:</H2>\n" +
                "<TABLE BORDER=1 ALIGN=CENTER>\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" +
                "</BODY></HTML>");

  }

  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    doGet(request, response);
  }
}
```

Here's a typical result, shown after visiting the page several without quitting the browser in between: