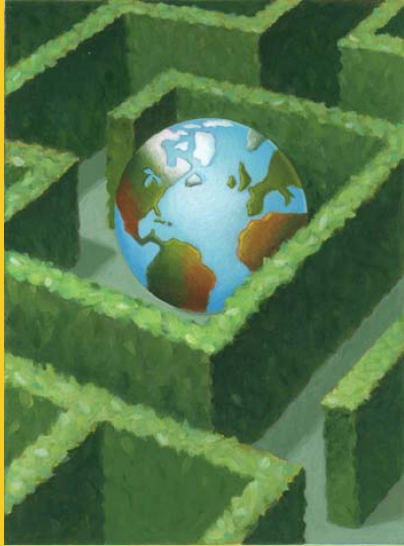


*core*  
**WEB**  
*programming*

# **Servlets**

# Agenda

- **Overview of servlet technology**
- **First servlets**
- **Handling the client request**
  - Form data
  - HTTP request headers
- **Generating the server response**
  - HTTP status codes
  - HTTP response headers
- **Handling cookies**
- **Session tracking**

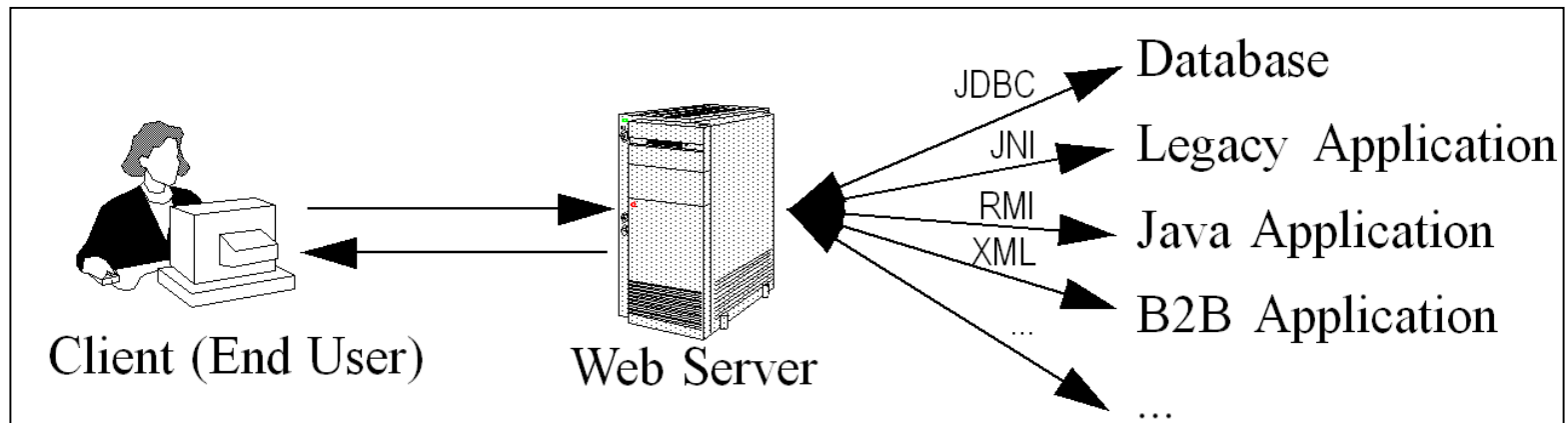


*core*  
**WEB**  
*programming*

# Overview

# A Servlet's Job

- Read explicit data sent by client (form data)
- Read implicit data sent by client (request headers)
- Generate the results
- Send the explicit data back to client (HTML)
- Send the implicit data to client (status codes and response headers)



# The Advantages of Servlets Over “Traditional” CGI

- **Efficient**
  - Threads instead of OS processes, one servlet copy, persistence
- **Convenient**
  - Lots of high-level utilities
- **Powerful**
  - Sharing data, pooling, persistence
- **Portable**
  - Run on virtually all operating systems and servers
- **Secure**
  - No shell escapes, no buffer overflows
- **Inexpensive**

# Why Build Pages Dynamically?

- **The Web page is based on data submitted by the user**
  - E.g., results page from search engines and order-confirmation pages at on-line stores
- **The Web page is derived from data that changes frequently**
  - E.g., a weather report or news headlines page
- **The Web page uses information from databases or other server-side sources**
  - E.g., an e-commerce site could use a servlet to build a Web page that lists the current price and availability of each item that is for sale

# Extending the Power of Servlets: JSP™

- **Idea:**

- Use regular HTML for most of page
- Mark dynamic content with special tags

```
<!DOCTYPE HTML PUBLIC
    "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Welcome to Our Store</TITLE></HEAD>
<BODY>
<H1>Welcome to Our Store</H1>
<SMALL>Welcome,
<!-- User name is "New User" for first-time visitors -->
<%= Utils.getUserNameFromCookie(request) %>
To access your account settings, click
<A HREF="Account-Settings.html">here.</A></SMALL><P>
Regular HTML for rest of on-line store's Web page
</BODY></HTML>
```

# Free Servlet and JSP Engines

- **Apache Tomcat**
  - <http://jakarta.apache.org/tomcat/>
  - See <http://archive.coreservlets.com/Using-Tomcat.html>
- **Allaire/Macromedia JRun**
  - <http://www.allaire.com/products/jrun/>
- **New Atlanta ServletExec**
  - <http://www.servletexec.com/>
- **Gefion Software LiteWebServer**
  - <http://www.gefionsoftware.com/LiteWebServer/>
- **Caucho's Resin**
  - <http://www.caucho.com/>

# Server-Side Java is Driving the Web



**Get on board or get out of the way**

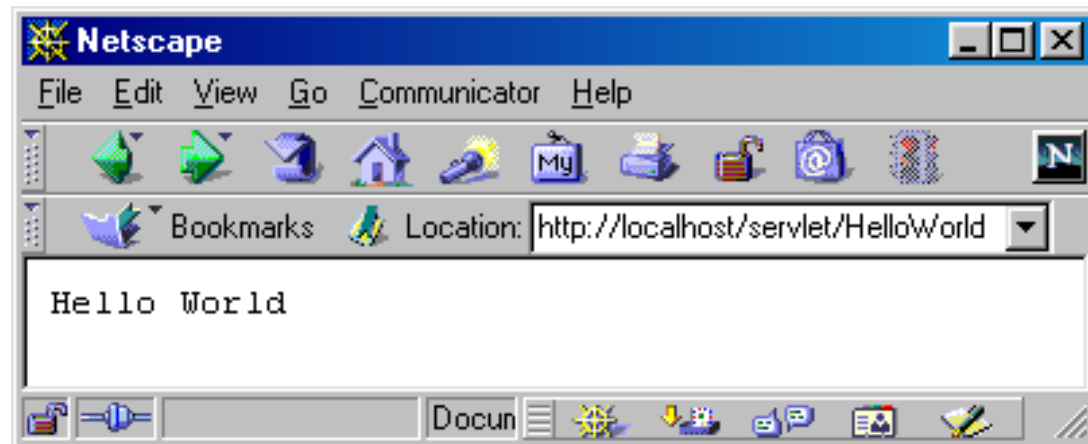
# Compiling and Invoking Servlets

- **Set your CLASSPATH**
  - Servlet JAR file (e.g., *install\_dir/lib/servlet.jar*).
  - Top of your package hierarchy
- **Put your servlet classes in proper location**
  - Locations vary from server to server. E.g.,
    - *tomcat\_install\_dir/webapps/ROOT/WEB-INF/classes*  
See <http://archive.coreservlets.com/Using-Tomcat.html>
    - *jrun install\_dir/servers/default/default-app/WEB-INF/classes*
- **Invoke your servlets**
  - `http://host/servlet/ServletName`
  - Custom URL-to-servlet mapping (via `web.xml`)

# A Simple Servlet That Generates Plain Text

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

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }
}
```



# Generating HTML

- **Set the Content-Type header**
  - Use `response.setContentType`
- **Output HTML**
  - Be sure to include the DOCTYPE
- **Use an HTML validation service**
  - <http://validator.w3.org/>
  - <http://www.htmlhelp.com/tools/validator/>

# A Servlet That Generates HTML

```
public class HelloWWW extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        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>Hello WWW</TITLE></HEAD>\n"+
                    "<BODY>\n" +
                    "<H1>Hello WWW</H1>\n" +
                    "</BODY></HTML>");
    }
}
```

# Some Simple HTML-Building Utilities

```
public class ServletUtilities {
    public static final String DOCTYPE =
        "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
        \"Transitional//EN\">";

    public static String headWithTitle(String title) {
        return(DOCTYPE + "\n" +
            "<HTML>\n" +
            "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
    }
    ...
}
```

- **Don't go overboard**

- Complete HTML generation packages usually not worth the bother (IMHO)
- The JSP framework is a better solution

# Packaging Servlets

- **Move the files to a subdirectory that matches the intended package name**
  - We'll use the **cwp** package. So, the class files need to go in a subdirectory called **cwp**.
- **Insert a package statement in the class file**
  - E.g., top of `SimplerHelloWWW.java`:  
`package cwp;`
- **Set CLASSPATH to include top-level development directory**
  - Same as with any Java programming, but everyone forgets this step!
- **Include package name in URL**
  - `http://localhost/servlet/cwp.SimplerHelloWWW`

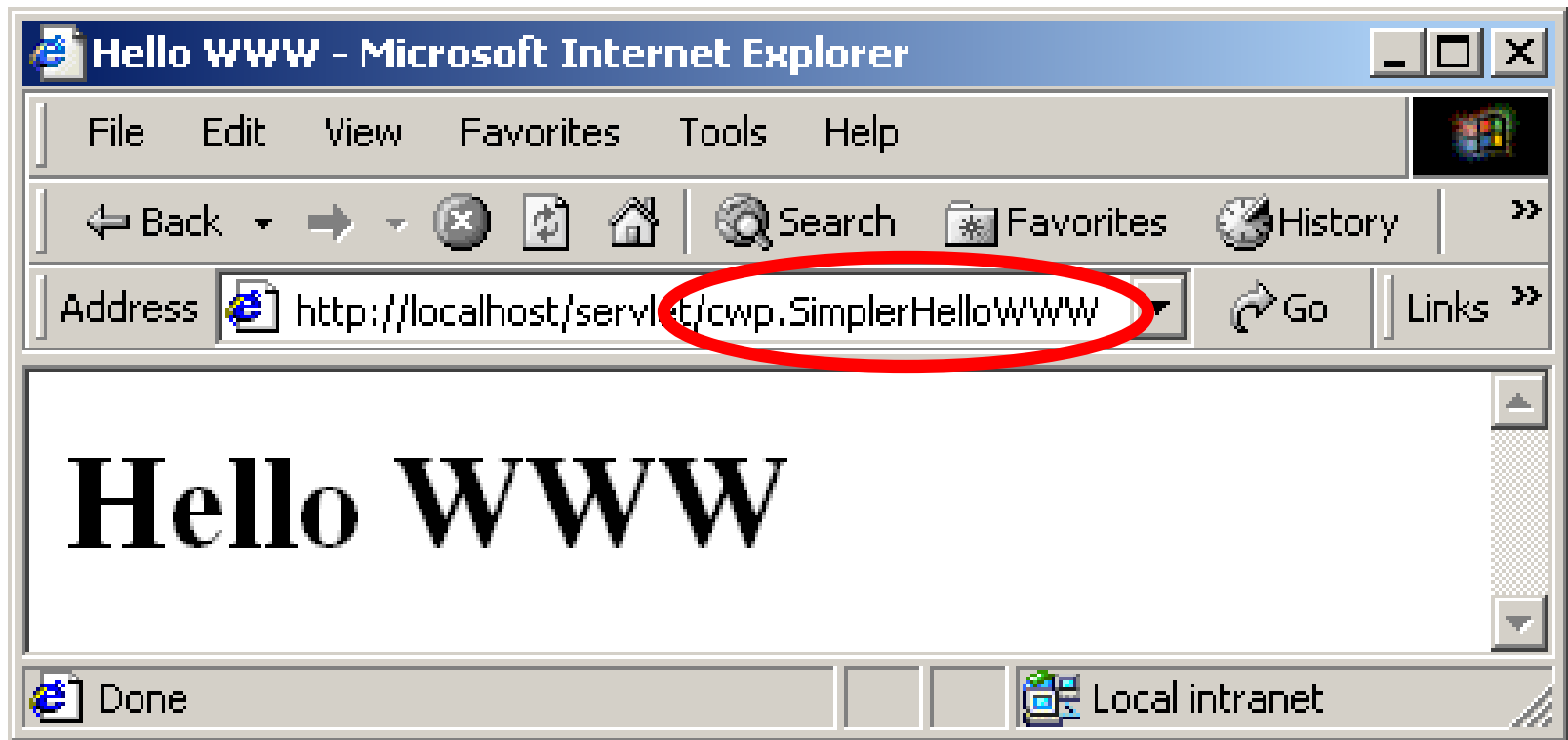
# HelloWWW with ServletUtilities and Packages

```
package cwp;

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

public class SimplerHelloWWW extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println(ServletUtilities.headWithTitle("Hello WWW") +
                    "<BODY>\n" +
                    "<H1>Hello WWW</H1>\n" +
                    "</BODY></HTML>");
    }
}
```

# SimplerHelloWWW Result



# The Servlet Life Cycle

- **init**
  - Executed once when the servlet is first loaded.  
Not called for each request
- **service**
  - Called in a new thread by server for each request.  
Dispatches to doGet, doPost, etc.  
*Don't override this method!*
- **doGet, doPost, doXxx**
  - Handles GET, POST, etc. requests
  - Override these methods to provide desired behavior
- **destroy**
  - Called when server deletes servlet instance.  
Not called after each request

# Why You Should Not Override service

- You can add support for other types of requests by adding doPut, doTrace, etc.
- You can add support for modification dates
  - Add a getLastModified method
- The service method gives you automatic support for:
  - HEAD, OPTIONS, and TRACE requests
- **Alternative: have doPost call doGet**

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response) ... {
    doGet(request, response);
}
```

# Initializing Servlets

- **Common in real-life servlets**
  - E.g., initializing database connection pools.
- **Use `ServletConfig.getInitParameter` to read initialization parameters**
  - Call `getServletConfig` to obtain the `ServletConfig` object
- **Set init parameters in `web.xml` (ver 2.2/2.3)**
  - `.../WEB-INF/web.xml`
  - Many servers have custom interfaces to create `web.xml`
- **It is common to use `init` even when you don't read init parameters**
  - E.g., to set up data structures that don't change during the life of the servlet, to load information from disk, etc.

# A Servlet That Uses Initialization Parameters

```
public class ShowMessage extends HttpServlet {
    private String message;
    private String defaultMessage = "No message.";
    private int repeats = 1;

    public void init() throws ServletException {
        ServletConfig config = getServletConfig();
        message = config.getInitParameter("message");
        if (message == null) {
            message = defaultMessage;
        }
        try {
            String repeatString =
                config.getInitParameter("repeats");
            repeats = Integer.parseInt(repeatString);
        } catch (NumberFormatException nfe) {}
    }
}
```

# ShowMessage Servlet (Continued)

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "The ShowMessage Servlet";
    out.println(ServletUtilities.headWithTitle(title) +
               "<BODY BGCOLOR=\"#FDF5E6\">\n" +
               "<H1 ALIGN=CENTER>" + title + "</H1>");
    for(int i=0; i<repeats; i++) {
        out.println(message + "<BR>");
    }
    out.println("</BODY></HTML>");
}
}
```

# Setting Init Parameters (Ver. 2.2 and Later)

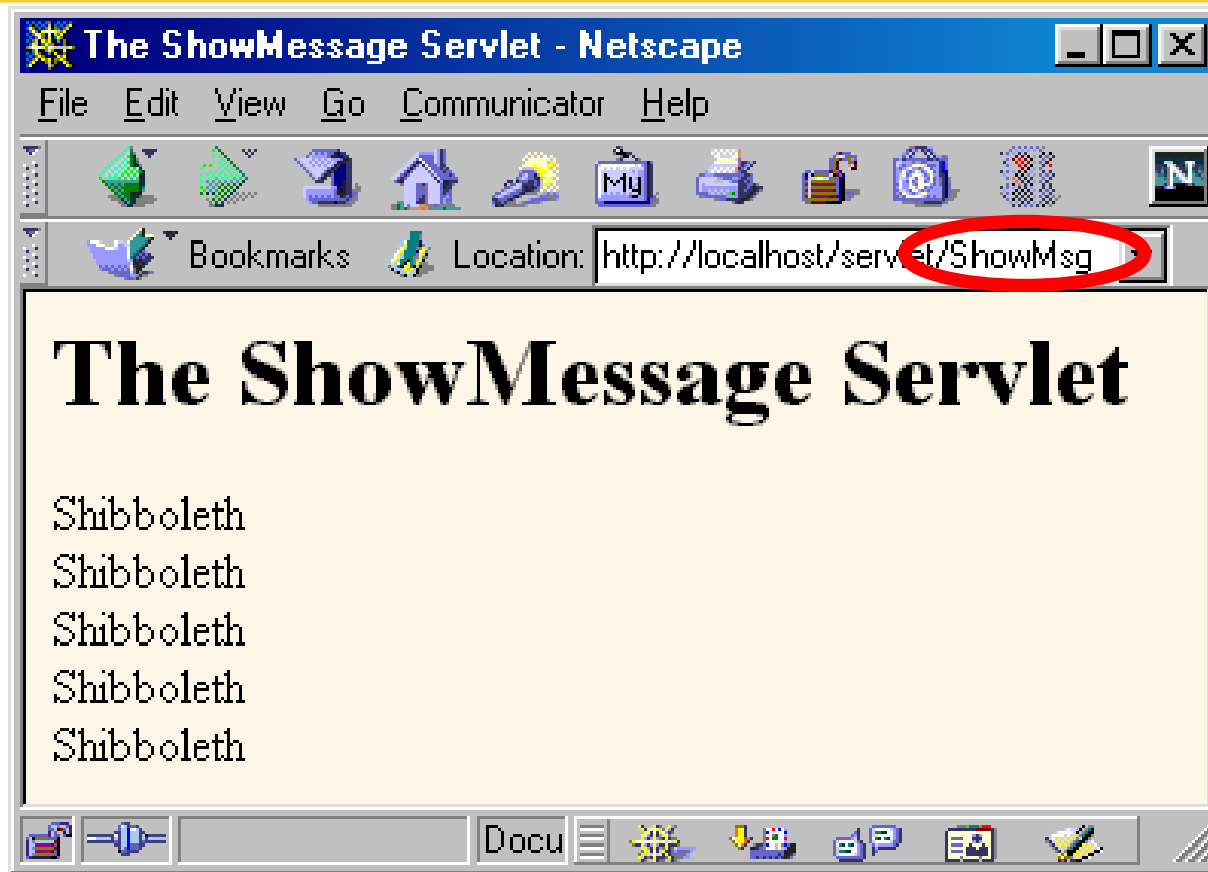
- **...\WEB-INF\web.xml**

- `tomcat_install_dir\webapps\ROOT\WEB-INF\web.xml`
- `jrun_install_dir\servers\default\default-app\WEB-INF\web.xml`

```
<web-app>
  <servlet>
    <servlet-name>ShowMsg</servlet-name>
    <servlet-class>cmp.ShowMessage</servlet-class>
    <init-param><param-name>message</param-name>
      <param-value>Shibboleth</param-value>
    </init-param>
    <init-param><param-name>repeats</param-name>
      <param-value>5</param-value>
    </init-param>
  </servlet>
</web-app>
```

- For lots of detail on web.xml, see  
*More Servlets and JavaServer Pages*

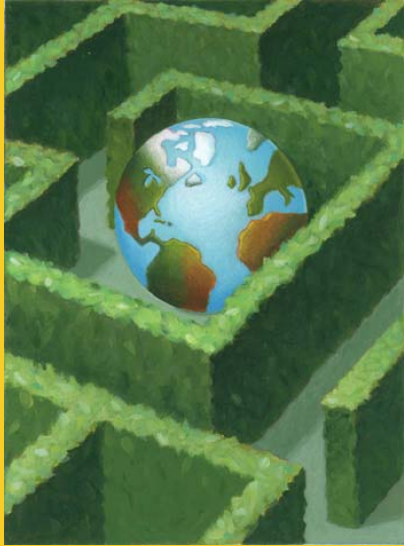
# ShowMessage Result



- **Note use of registered name**
  - `http://host/servlet/RegisteredName`, **not**  
`http://host/servlet/packageName.RealName`

# Debugging Servlets

- **Use print statements; run server on desktop**
- **Integrated debugger in IDE**
- **Look at the HTML source**
- **Return error pages to the client**
  - Plan ahead for missing/malformed data
- **Use the log file**
  - `log("message")` or `log("message", Throwable)`
- **Look at the request data separately**
  - See EchoServer at [archive.corewebprogramming.com](http://archive.corewebprogramming.com)
- **Look at the response data separately**
  - See WebClient at [archive.corewebprogramming.com](http://archive.corewebprogramming.com)
- **Stop and restart the server**



*core*  
**WEB**  
*programming*

# Handling the Client Request: Form Data

# Handling the Client Request: Form Data

- **Example URL at online travel agent**
  - `http://host/path?user=Marty+Hall&origin=iad&dest=nrt`
  - Names (user) come from HTML author; values (Marty+Hall) usually come from end user
- **Parsing form (query) data in traditional CGI**
  - Read the data one way for GET requests, another way for POST requests
  - Chop pairs at `&`, then separate parameter names (left of the `"="`) from parameter values (right of the `"="`)
  - URL decode values (e.g., `"%7E"` becomes `"~"`)
  - Need special cases for omitted values (`param1=val1&param2=&param3=val3`) and repeated params (`param1=val1&param2=val2&param1=val3`)

# Reading Form Data (Query Data)

- **getParameter("name")**
  - Returns value as user entered it. I.e., URL-decoded value of first occurrence of name in query string.
  - Works identically for GET and POST requests
  - Returns null if no such parameter is in query
- **getParameterValues("name")**
  - Returns an array of the URL-decoded values of all occurrences of name in query string
  - Returns a one-element array if param not repeated
  - Returns null if no such parameter is in query
- **getParameterNames()**
  - Returns Enumeration of request params

# An HTML Form With Three Parameters

```
<FORM ACTION="/servlet/cwp.ThreeParams">
```

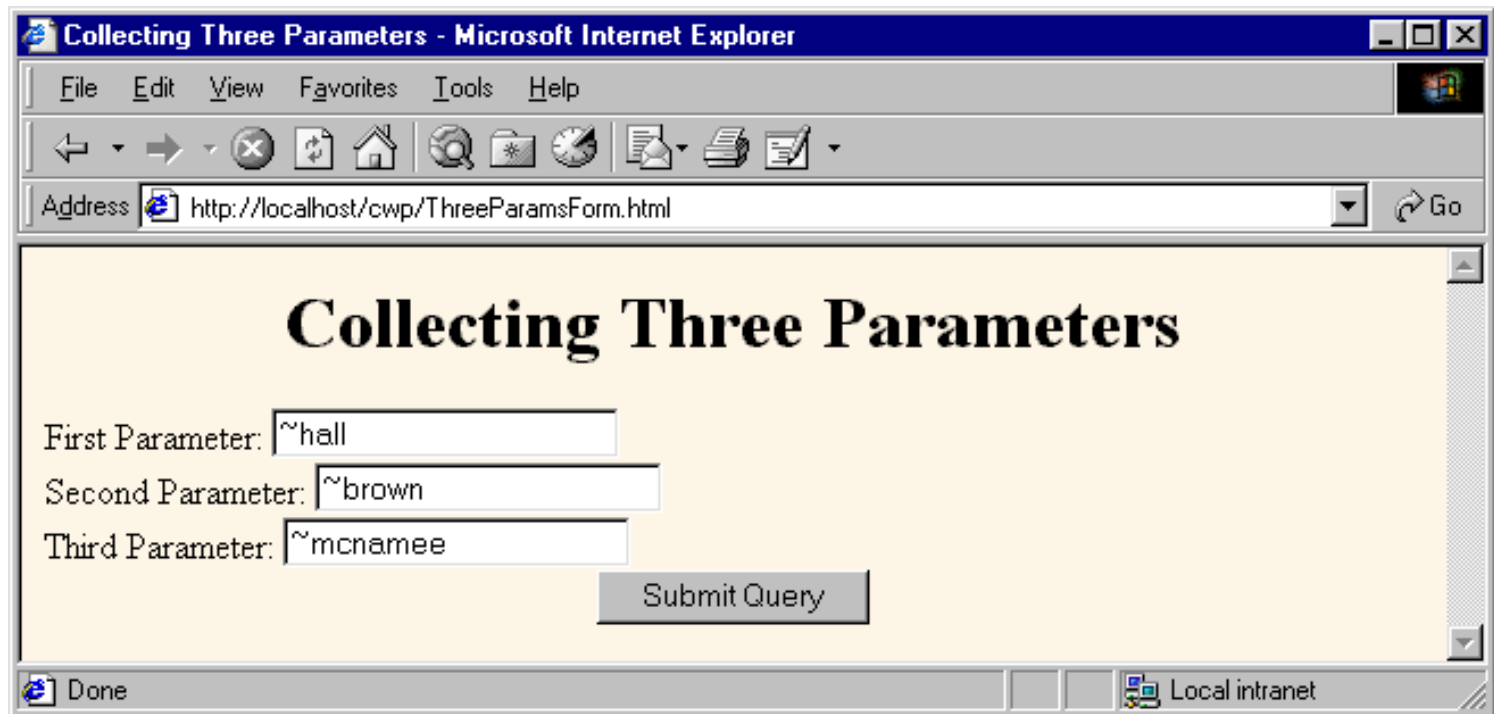
```
  First Parameter:  <INPUT TYPE="TEXT" NAME="param1"><BR>
```

```
  Second Parameter: <INPUT TYPE="TEXT" NAME="param2"><BR>
```

```
  Third Parameter:  <INPUT TYPE="TEXT" NAME="param3"><BR>
```

```
  <CENTER><INPUT TYPE="SUBMIT"></CENTER>
```

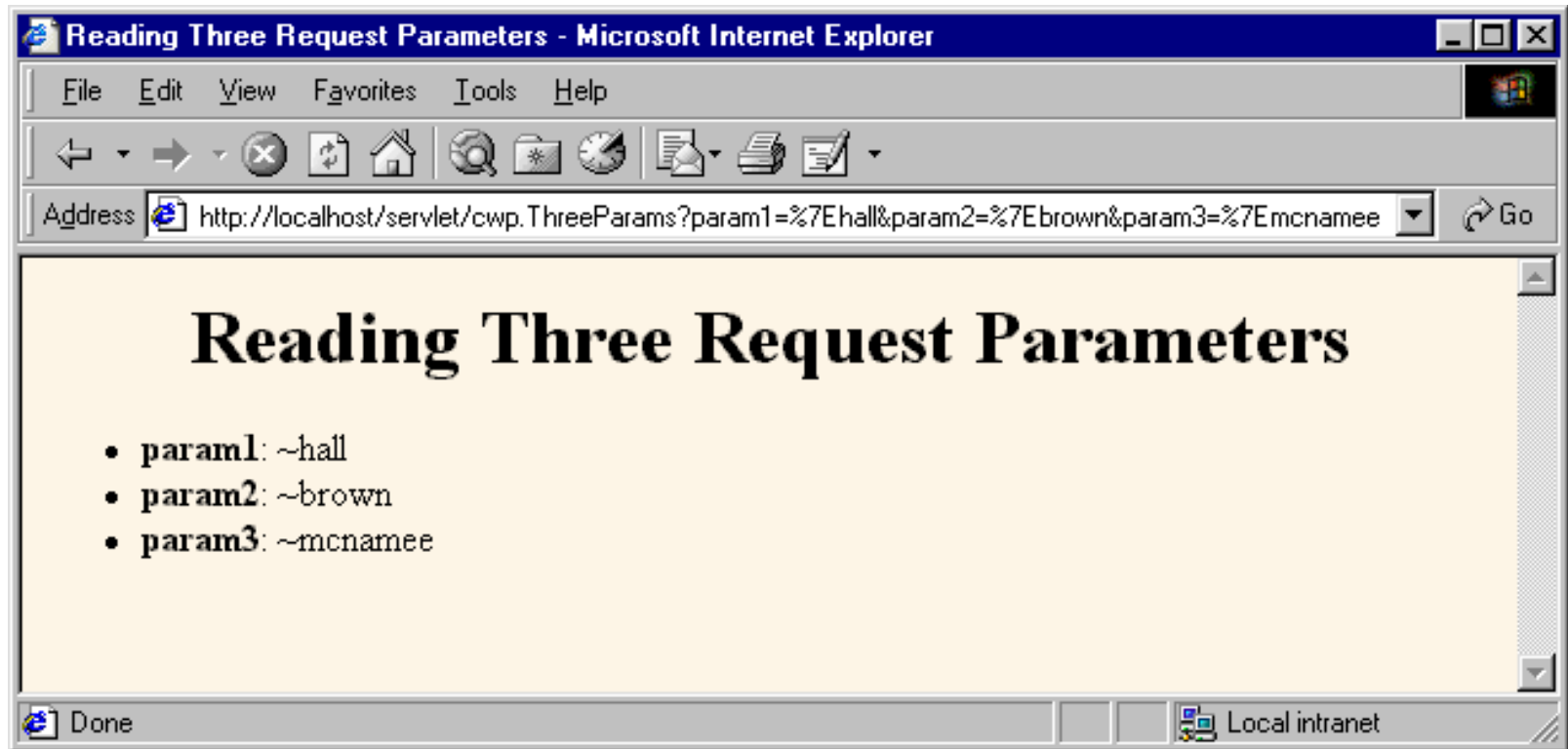
```
</FORM>
```



# Reading the Three Parameters

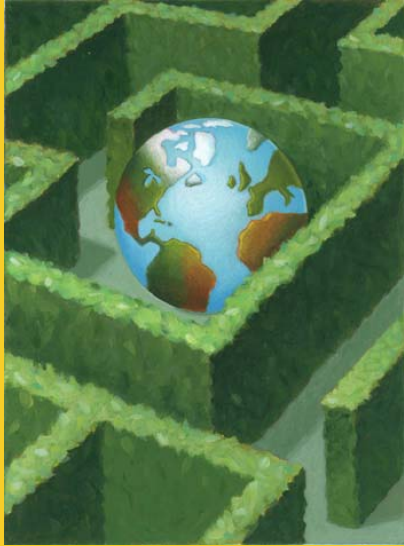
```
public class ThreeParams extends HttpServlet {
    public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Reading Three Request Parameters";
        out.println(ServletUtilities.headWithTitle(title) +
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
            "<UL>\n" +
            "    <LI><B>param1</B>: "
            + request.getParameter("param1") + "\n" +
            "    <LI><B>param2</B>: "
            + request.getParameter("param2") + "\n" +
            "    <LI><B>param3</B>: "
            + request.getParameter("param3") + "\n" +
            "</UL>\n" +
            "</BODY></HTML>");
    }
}
```

# Reading Three Parameters: Result



# Filtering Strings for HTML-Specific Characters

- **You cannot safely insert arbitrary strings into servlet output**
  - < and > can cause problems anywhere
  - & and " cause problems inside of HTML attributes
- **You sometimes cannot manually translate**
  - String is derived from a program excerpt or another source where it is already in standard format
  - String is derived from HTML form data
- **Failing to filter special characters makes you vulnerable to cross-site scripting attack**
  - <http://www.cert.org/advisories/CA-2000-02.html>
  - <http://www.microsoft.com/technet/security/crssite.asp>
- **See filter method of ServletUtilities at <http://www.corewebprogramming.com>**



*core*  
**WEB**  
*programming*

# Handling the Client Request: HTTP Request Headers

# Handling the Client Request: HTTP Request Headers

- **Example HTTP 1.1 Request**
  - GET /search?keywords=servlets+jsp HTTP/1.1
  - Accept: image/gif, image/jpg, \*/\*
  - Accept-Encoding: gzip
  - Connection: Keep-Alive
  - Cookie: userID=id456578
  - Host: www.somebookstore.com
  - Referer: http://www.somebookstore.com/findbooks.html
  - User-Agent: Mozilla/4.7 [en] (Win98; U)
- **It shouldn't take a rocket scientist to realize you need to understand HTTP to be effective with servlets or JSP**

# Reading Request Headers

- **General**
  - `getHeader`
  - `getHeaders`
  - `getHeaderNames`
- **Specialized**
  - `getCookies`
  - `getAuthType` and `getRemoteUser`
  - `getContentLength`
  - `getContentType`
  - `getDateHeader`
  - `getIntHeader`
- **Related info**
  - `getMethod`, `getRequestURI`, `getProtocol`

# Printing All Headers

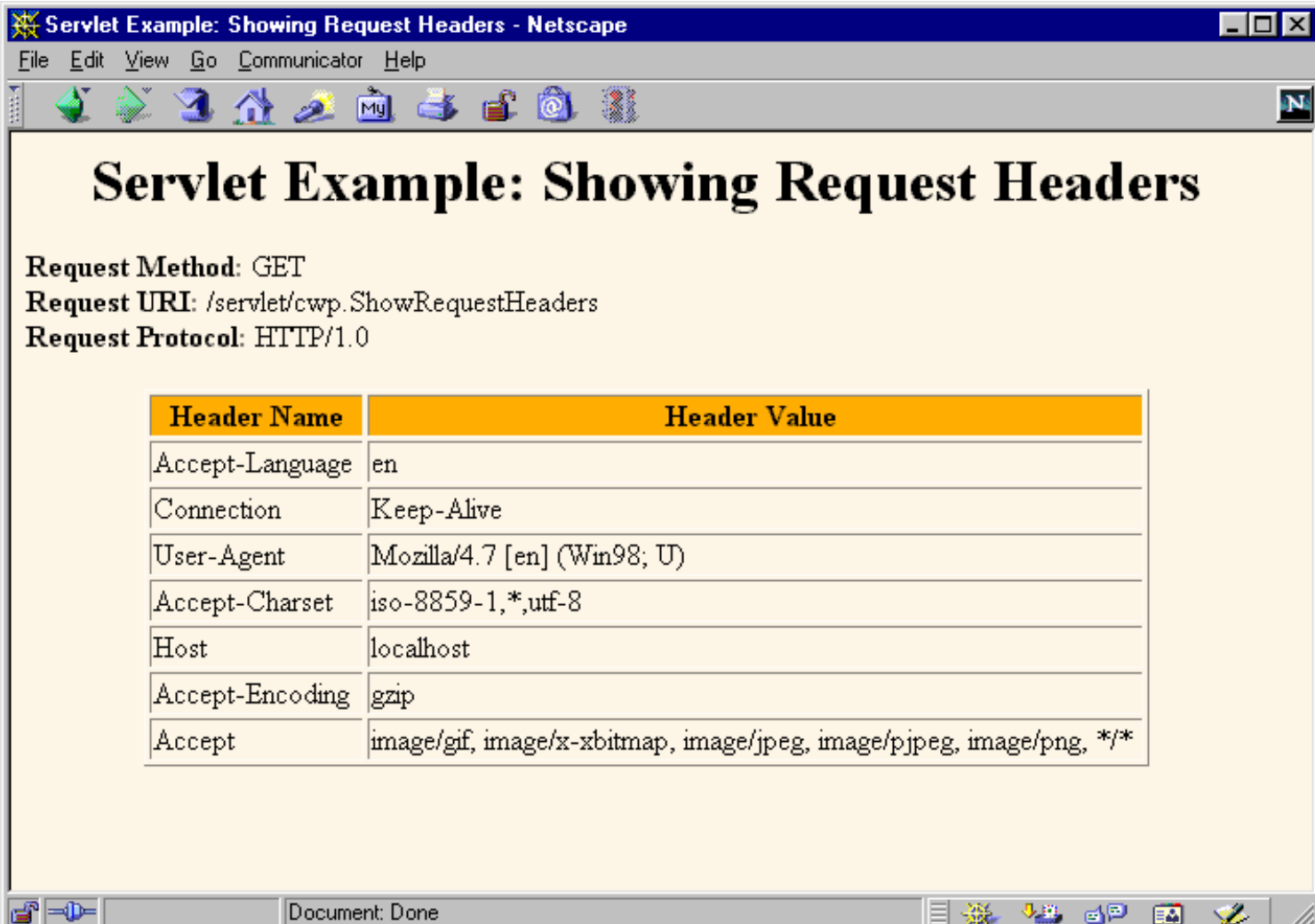
```
public class ShowRequestHeaders extends HttpServlet {
    public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Servlet Example: Showing Request Headers";
        out.println(ServletUtilities.headWithTitle(title) +
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
            "<B>Request Method: </B>" +
            request.getMethod() + "<BR>\n" +
            "<B>Request URI: </B>" +
            request.getRequestURI() + "<BR>\n" +
            "<B>Request Protocol: </B>" +
            request.getProtocol() + "<BR><BR>\n" +
```

# Printing All Headers (Continued)

```
        "<TABLE BORDER=1 ALIGN=CENTER>\n" +
        "<TR BGCOLOR=#FFAD00>\n" +
        "<TH>Header Name<TH>Header Value");
Enumeration headerNames = request.getHeaderNames();
while(headerNames.hasMoreElements()) {
    String headerName = (String)headerNames.nextElement();
    out.println("<TR><TD>" + headerName);
    out.println("    <TD>" + request.getHeader(headerName));
}
out.println("</TABLE>\n</BODY></HTML>");
}

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

# Printing All Headers: Netscape Result



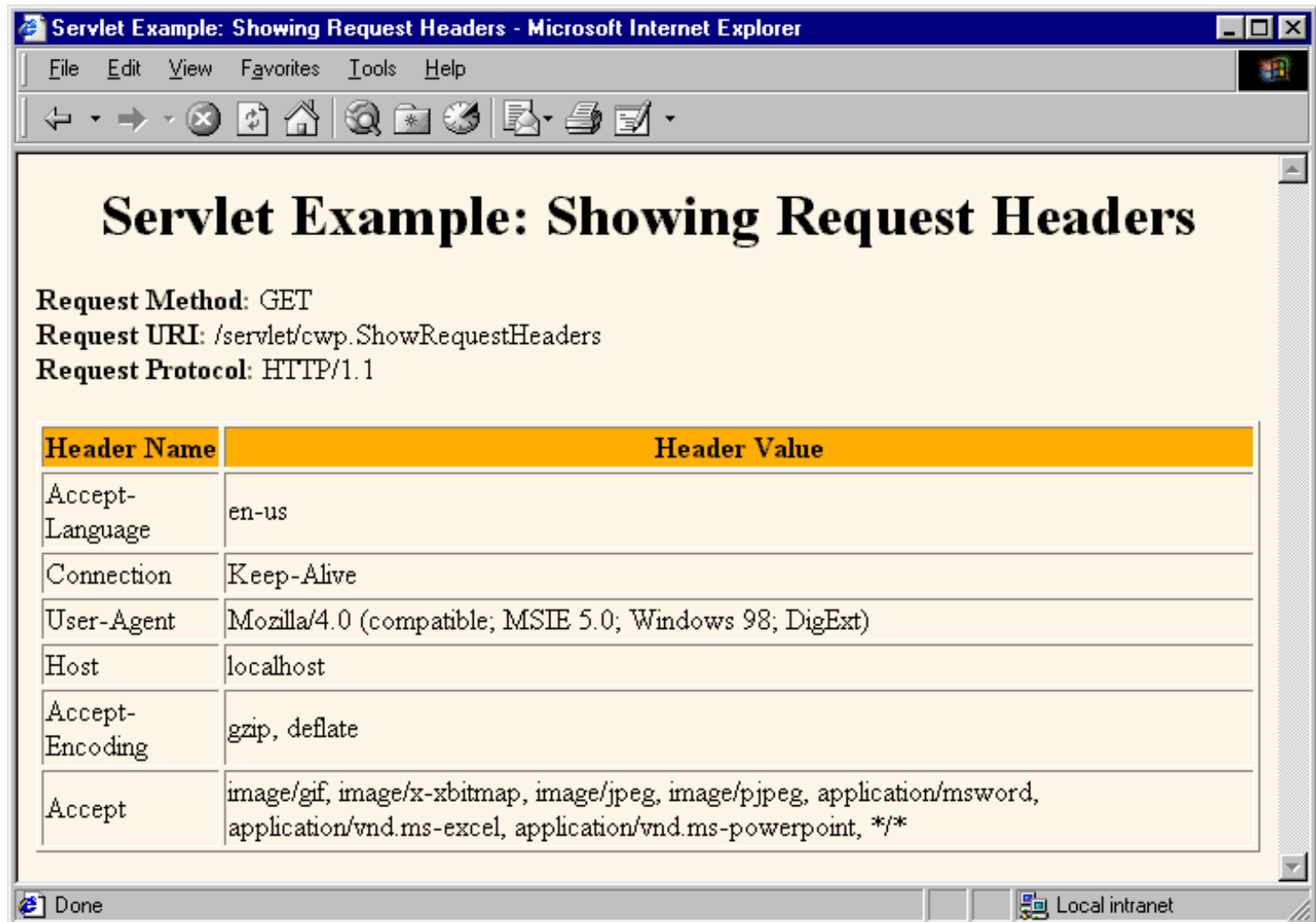
The screenshot shows a Netscape browser window titled "Servlet Example: Showing Request Headers - Netscape". The address bar contains the URL "/servlet/cwp.ShowRequestHeaders". The main content area displays the following information:

**Request Method:** GET  
**Request URI:** /servlet/cwp.ShowRequestHeaders  
**Request Protocol:** HTTP/1.0

Header Name	Header Value
Accept-Language	en
Connection	Keep-Alive
User-Agent	Mozilla/4.7 [en] (Win98; U)
Accept-Charset	iso-8859-1,*,utf-8
Host	localhost
Accept-Encoding	gzip
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*

The status bar at the bottom of the browser window shows "Document: Done".

# Printing All Headers: Internet Explorer Result



**Servlet Example: Showing Request Headers**

**Request Method:** GET  
**Request URI:** /servlet/cwp.ShowRequestHeaders  
**Request Protocol:** HTTP/1.1

Header Name	Header Value
Accept-Language	en-us
Connection	Keep-Alive
User-Agent	Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
Host	localhost
Accept-Encoding	gzip, deflate
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/msword, application/vnd.ms-excel, application/vnd.ms-powerpoint, */*

Done Local intranet

# Common HTTP 1.1 Request Headers

- **Accept**
  - Indicates MIME types browser can handle
  - Can send different content to different clients
- **Accept-Encoding**
  - Indicates encodings (e.g., gzip) browser can handle
  - See following example
- **Authorization**
  - User identification for password-protected pages.
  - Instead of HTTP authorization, use HTML forms to send username/password. Store in session object.
    - For details on programming security manually and using web.xml to tell the server to enforce security automatically, see *More Servlets and JavaServer Pages*.

# Common HTTP 1.1 Request Headers (Continued)

- **Connection**

- In HTTP 1.0, keep-alive means browser can handle persistent connection. In HTTP 1.1, persistent connection is default. Persistent connections mean that the server can reuse the same socket over again for requests very close together from the same client
- Servlets can't do this unilaterally; the best they can do is to give the server enough info to permit persistent connections. So, they should set Content-Length with `setContentLength` (using `ByteArrayOutputStream` to determine length of output). See example in *Core Servlets and JavaServer Pages*.

- **Cookie**

- Gives cookies previously sent to client. Use `getCookies`, not `getHeader`. (See later slides)

# Common HTTP 1.1 Request Headers (Continued)

- **Host**

- Indicates host given in original URL
- This is a required header in HTTP 1.1. This fact is important to know if you write a custom HTTP client (e.g., WebClient used in book) or telnet to a server and use the HTTP/1.1 version

- **If-Modified-Since**

- Indicates client wants page only if it has been changed after specified date
- Don't handle this situation directly; implement `getLastModified` instead. See example in *Core Servlets and JavaServer Pages* Chapter 2

# Common HTTP 1.1 Request Headers (Continued)

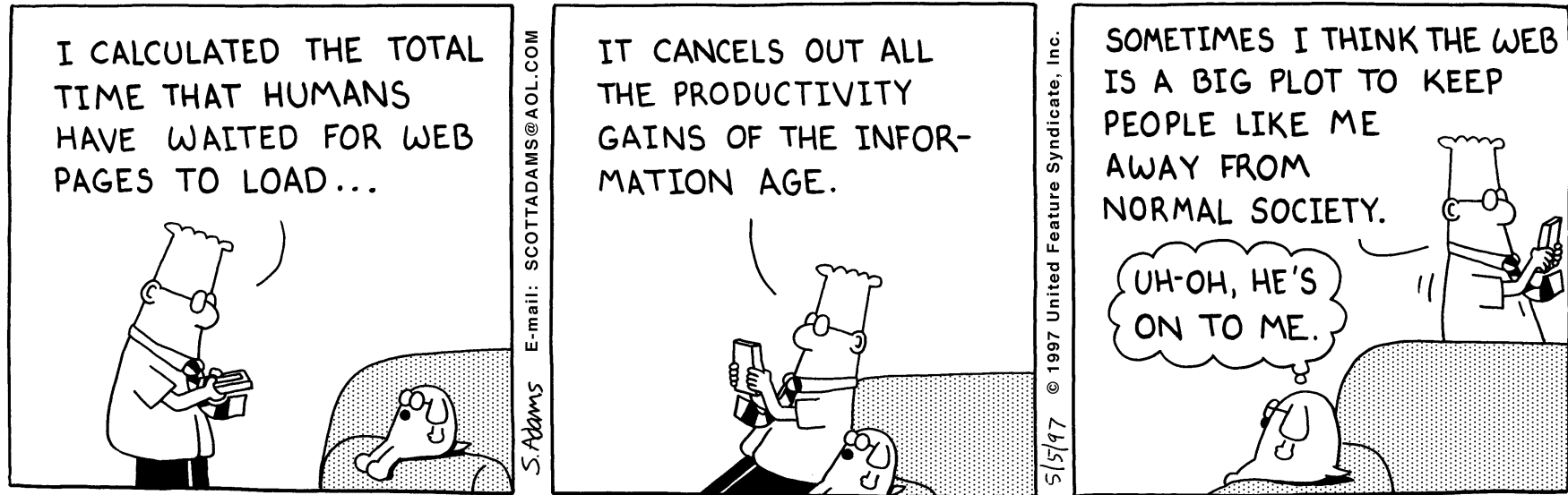
- **Referer**

- URL of referring Web page
- Useful for tracking traffic; logged by many servers
- Can be easily spoofed

- **User-Agent**

- String identifying the browser making the request
- Use sparingly
- Again, can be easily spoofed

# Sending Compressed Web Pages



Dilbert used with permission of United Syndicates Inc.

# Sending Compressed Pages: EncodedPage.java

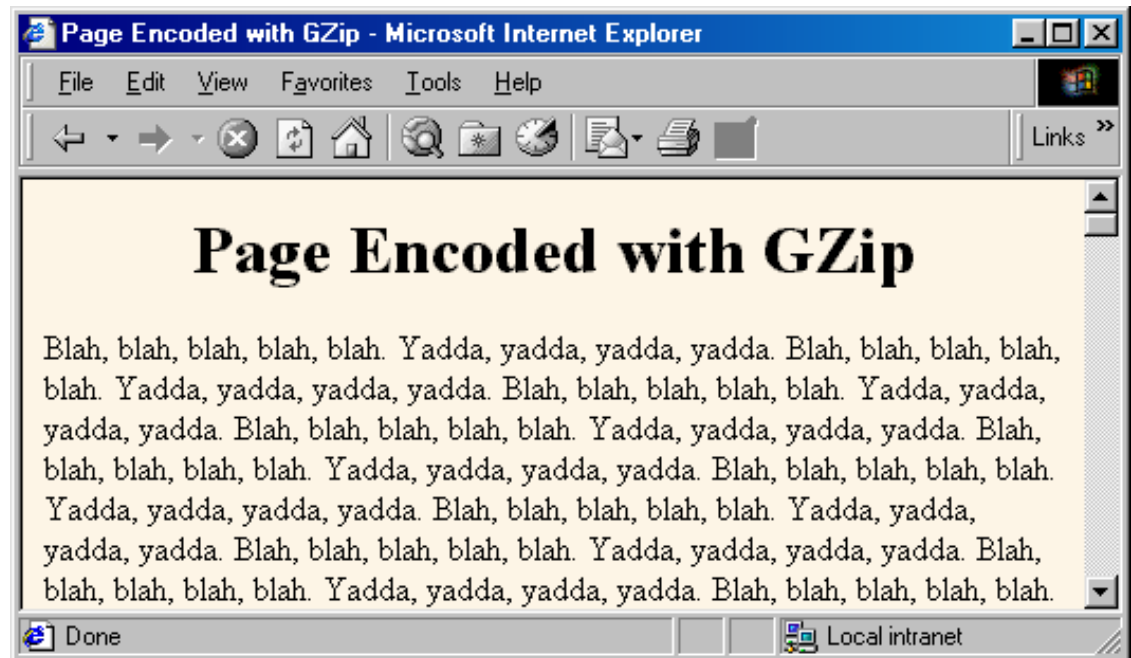
```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    String encodings = request.getHeader("Accept-Encoding");
    String encodeFlag = request.getParameter("encoding");
    PrintWriter out;
    String title;
    if ((encodings != null) &&
        (encodings.indexOf("gzip") != -1) &&
        !"none".equals(encodeFlag)) {
        title = "Page Encoded with GZip";
        OutputStream out1 = response.getOutputStream();
        out = new PrintWriter(new GZIPOutputStream(out1), false);
        response.setHeader("Content-Encoding", "gzip");
    } else {
        title = "Unencoded Page";
        out = response.getWriter();
    }
}
```

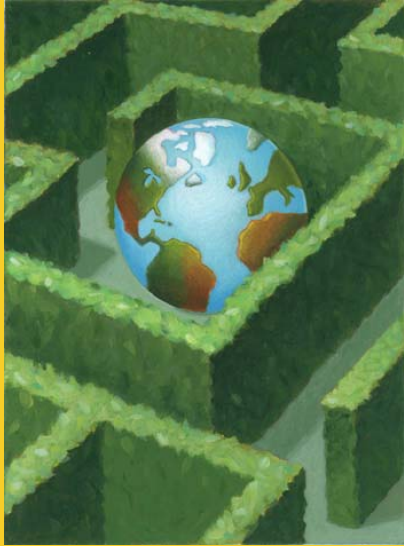
# EncodedPage.java (Continued)

```
out.println(ServletUtilities.headWithTitle(title) +
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1 ALIGN=CENTER>" + title + "</H1>\n");
String line = "Blah, blah, blah, blah, blah. " +
            "Yadda, yadda, yadda, yadda.";
for(int i=0; i<10000; i++) {
    out.println(line);
}
out.println("</BODY></HTML>");
out.close();
}
```

# Sending Compressed Pages: Results

- **Uncompressed (28.8K modem), Netscape 4.7 and Internet Explorer 5.0: > 50 seconds**
- **Compressed (28.8K modem), Netscape 4.7 and Internet Explorer 5.0: < 5 seconds**
- **Caution: be careful about generalizing benchmarks**





*core*  
**WEB**  
*programming*

# Generating the HTTP Response

# Generating the Server Response: HTTP Status Codes

- **Example HTTP 1.1 Response**

```
HTTP/1.1 200 OK
```

```
Content-Type: text/html
```

```
<!DOCTYPE ... >
```

```
<HTML>
```

```
...
```

```
</HTML>
```

- **Changing the status code lets you perform a number of tasks not otherwise possible**

- Forward client to another page
- Indicate a missing resource
- Instruct browser to use cached copy

- **Set status *before* sending document**

# Setting Status Codes

- **public void setStatus(int statusCode)**
  - Use a constant for the code, not an explicit int. Constants are in `HttpServletResponse`
  - Names derived from standard message. E.g., `SC_OK`, `SC_NOT_FOUND`, etc.
- **public void sendError(int code, String message)**
  - Wraps message inside small HTML document
- **public void sendRedirect(String url)**
  - Relative URLs permitted in 2.2/2.3
  - Also sets Location header

# Common HTTP 1.1 Status Codes

- **200 (OK)**
  - Everything is fine; document follows
  - Default for servlets
- **204 (No Content)**
  - Browser should keep displaying previous document
- **301 (Moved Permanently)**
  - Requested document permanently moved elsewhere (indicated in Location header)
  - Browsers go to new location automatically

# Common HTTP 1.1 Status Codes (Continued)

- **302 (Found)**
  - Requested document temporarily moved elsewhere (indicated in Location header)
  - Browsers go to new location automatically
  - Servlets should use `sendRedirect`, not `setStatus`, when setting this header. See example
- **401 (Unauthorized)**
  - Browser tried to access protected page without proper Authorization header. See example in book
- **404 (Not Found)**
  - No such page. Servlets should use `sendError` to set this header
  - Problem: Internet Explorer 5.0

# A Front End to Various Search Engines: Code

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    String searchString =
        request.getParameter("searchString");
    if ((searchString == null) ||
        (searchString.length() == 0)) {
        reportProblem(response, "Missing search string.");
        return;
    }
    searchString = URLEncoder.encode(searchString);
    String numResults =
        request.getParameter("numResults");
    ...
    String searchEngine =
        request.getParameter("searchEngine");
```

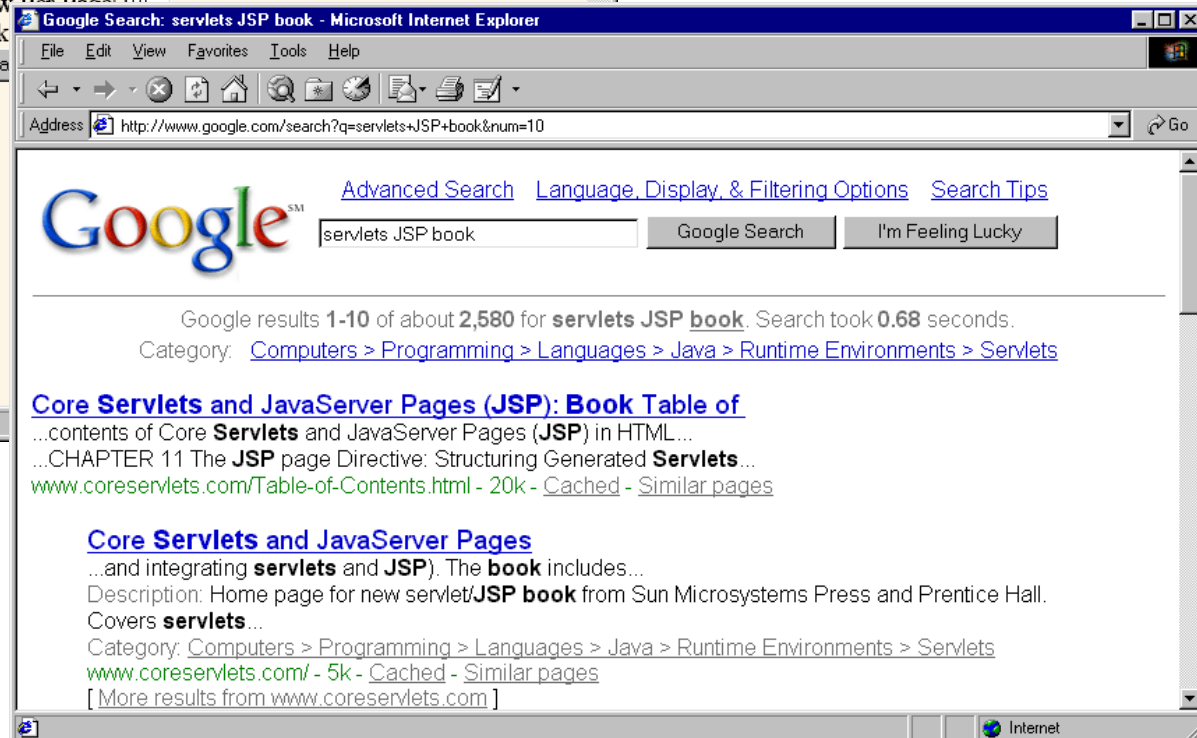
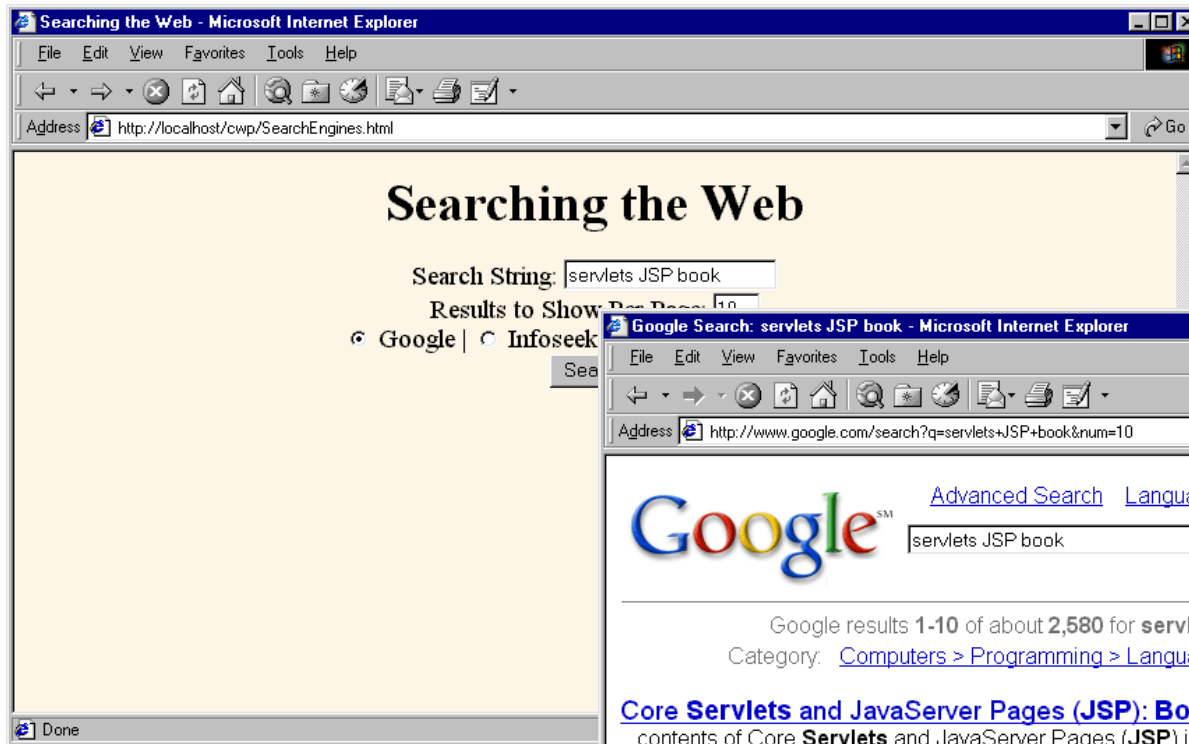
# A Front End to Various Search Engines: Code (Continued)

```
SearchSpec[] commonSpecs =
    SearchSpec.getCommonSpecs();
for(int i=0; i<commonSpecs.length; i++) {
    SearchSpec searchSpec = commonSpecs[i];
    if (searchSpec.getName().equals(searchEngine)) {
        String url =
            searchSpec.makeURL(searchString, numResults);
        response.sendRedirect(url);
        return;
    }
}
reportProblem(response,
    "Unrecognized search engine.");
```

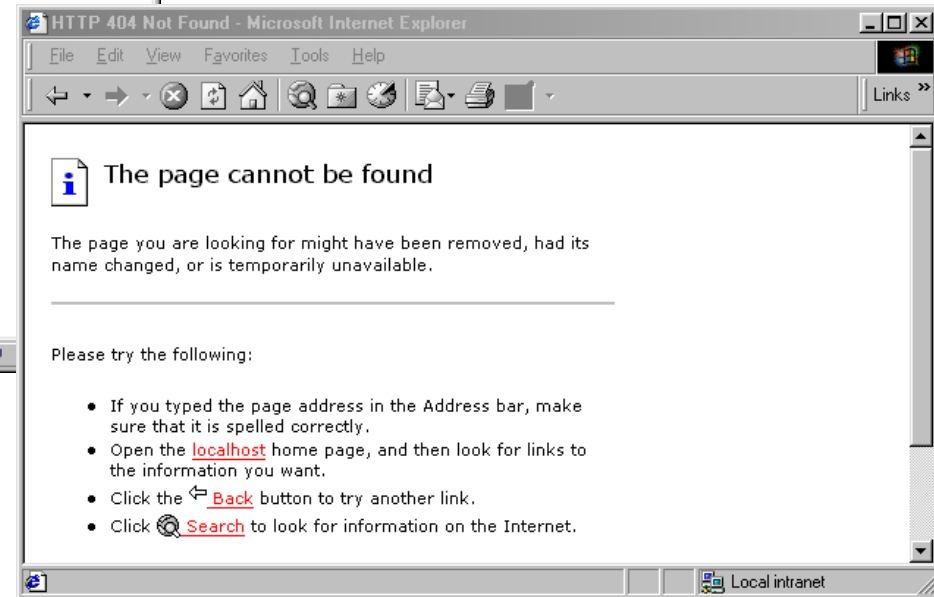
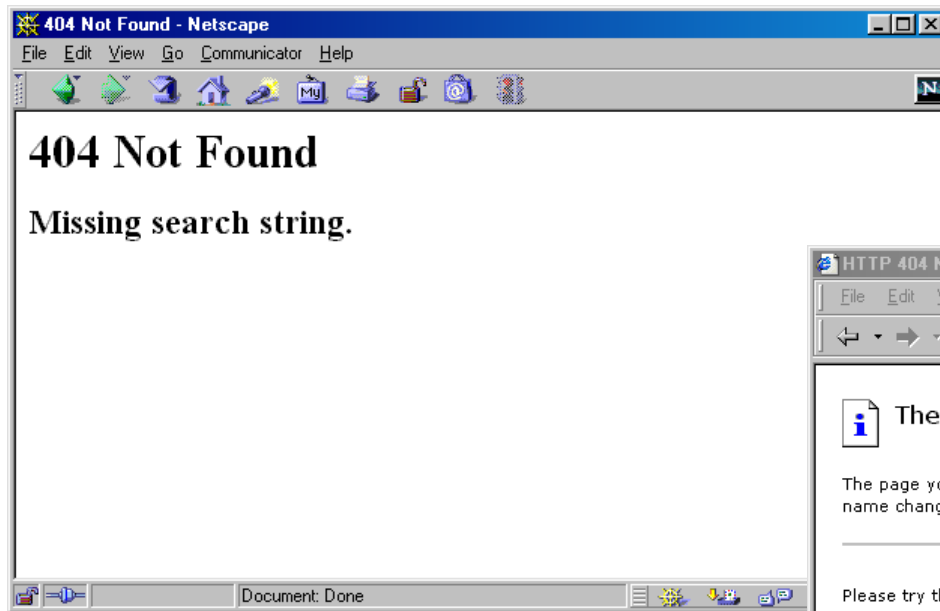
# A Front End to Various Search Engines: Code (Continued)

```
private void reportProblem(HttpServletRequestResponse response,  
                           String message)  
    throws IOException {  
    response.sendError(response.SC_NOT_FOUND,  
                       "<H2>" + message + "</H2>");  
}
```

# Front End to Search Engines: Result of Legal Request



# Front End to Search Engines: Result of Illegal Request



## — Fix:

- Tools, Internet Options, Advanced
- Deselect "Show 'friendly' HTTP error messages"
- Not a real fix -- doesn't help unsuspecting users of your pages

# Generating the Server Response: HTTP Response Headers

- **Purposes**

- Give forwarding location
- Specify cookies
- Supply the page modification date
- Instruct the browser to reload the page after a designated interval
- Give the document size so that persistent HTTP connections can be used
- Designate the type of document being generated
- Etc.

# Setting Arbitrary Response Headers

- **public void `setHeader`(String headerName, String headerValue)**
  - Sets an arbitrary header
- **public void `setDateHeader`(String name, long millisecs)**
  - Converts millis since 1970 to date in GMT format
- **public void `setIntHeader`(String name, int headerValue)**
  - Prevents need to convert int to String
- **`addHeader`, `addDateHeader`, `addIntHeader`**
  - Adds header instead of replacing

# Setting Common Response Headers

- **setContentType**

- Sets the Content-Type header.  
Servlets almost always use this header.  
See Table 19.1 (Common MIME Types).

- **setContentLength**

- Sets the Content-Length header.  
Used for persistent HTTP connections.  
See Connection request header.

- **addCookie**

- Adds a value to the Set-Cookie header.  
See separate section on cookies.

- **sendRedirect**

- Sets Location header (plus changes status code)

# Common HTTP 1.1 Response Headers

- **Cache-Control (1.1) and Pragma (1.0)**
  - A no-cache value prevents browsers from caching page. Send both headers or check HTTP version
- **Content-Encoding**
  - The way document is encoded. Browser reverses this encoding before handling document. See compression example earlier.
- **Content-Length**
  - The number of bytes in the response
  - See `setContentLength` on previous slide
  - Use `ByteArrayOutputStream` to buffer document so you can determine size.
    - See detailed example in *Core Servlets and JavaServer Pages*

# Common HTTP 1.1 Response Headers (Continued)

- **Content-Type**

- The MIME type of the document being returned.
- Use `setContentType` to set this header

- **Expires**

- The time at which document should be considered out-of-date and thus should no longer be cached
- Use `setDateHeader` to set this header

- **Last-Modified**

- The time document was last changed.
- Don't set this header explicitly; provide a `getLastModified` method instead.
  - See example in *Core Servlets and JavaServer Pages* Chapter 2

# Common HTTP 1.1 Response Headers (Continued)

- **Location**

- The URL to which browser should reconnect.
- Use `sendRedirect` instead of setting this directly.

- **Refresh**

- The number of seconds until browser should reload page. Can also include URL to connect to. See following example.

- **Set-Cookie**

- The cookies that browser should remember. Don't set this header directly; use `addCookie` instead.

- **WWW-Authenticate**

- The authorization type and realm needed in Authorization header. See details in *More Servlets & JavaServer Pages*.

# Persistent Servlet State and Auto-Reloading Pages

- **Idea: generate list of large (e.g., 150-digit) prime numbers**
  - Show partial results until completed
  - Let new clients make use of results from others
- **Demonstrates use of the Refresh header**
- **Shows how easy it is for servlets to maintain state between requests**
  - Very difficult in traditional CGI
- **Also illustrates that servlets can handle multiple simultaneous connections**
  - Each request is in a separate thread
  - Synchronization required for shared data

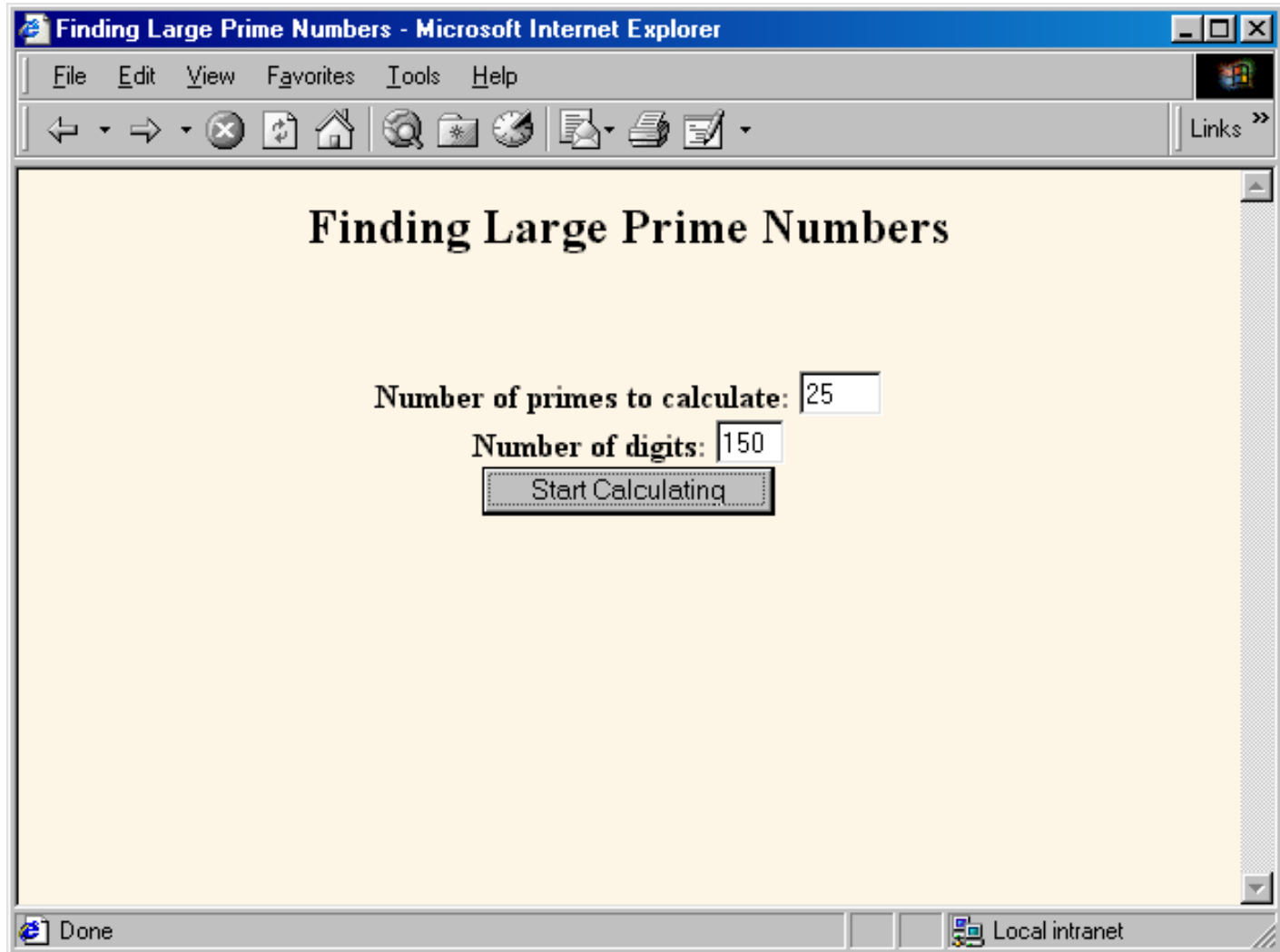
# Generating Prime Numbers: Source Code

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    int numPrimes =
        ServletUtilities.getIntParameter(request,
                                         "numPrimes", 50);
    int numDigits =
        ServletUtilities.getIntParameter(request,
                                         "numDigits", 120);
    // findPrimeList is synchronized
    PrimeList primeList =
        findPrimeList(primeListVector, numPrimes, numDigits);
    if (primeList == null) {
        primeList = new PrimeList(numPrimes, numDigits, true);
    }
}
```

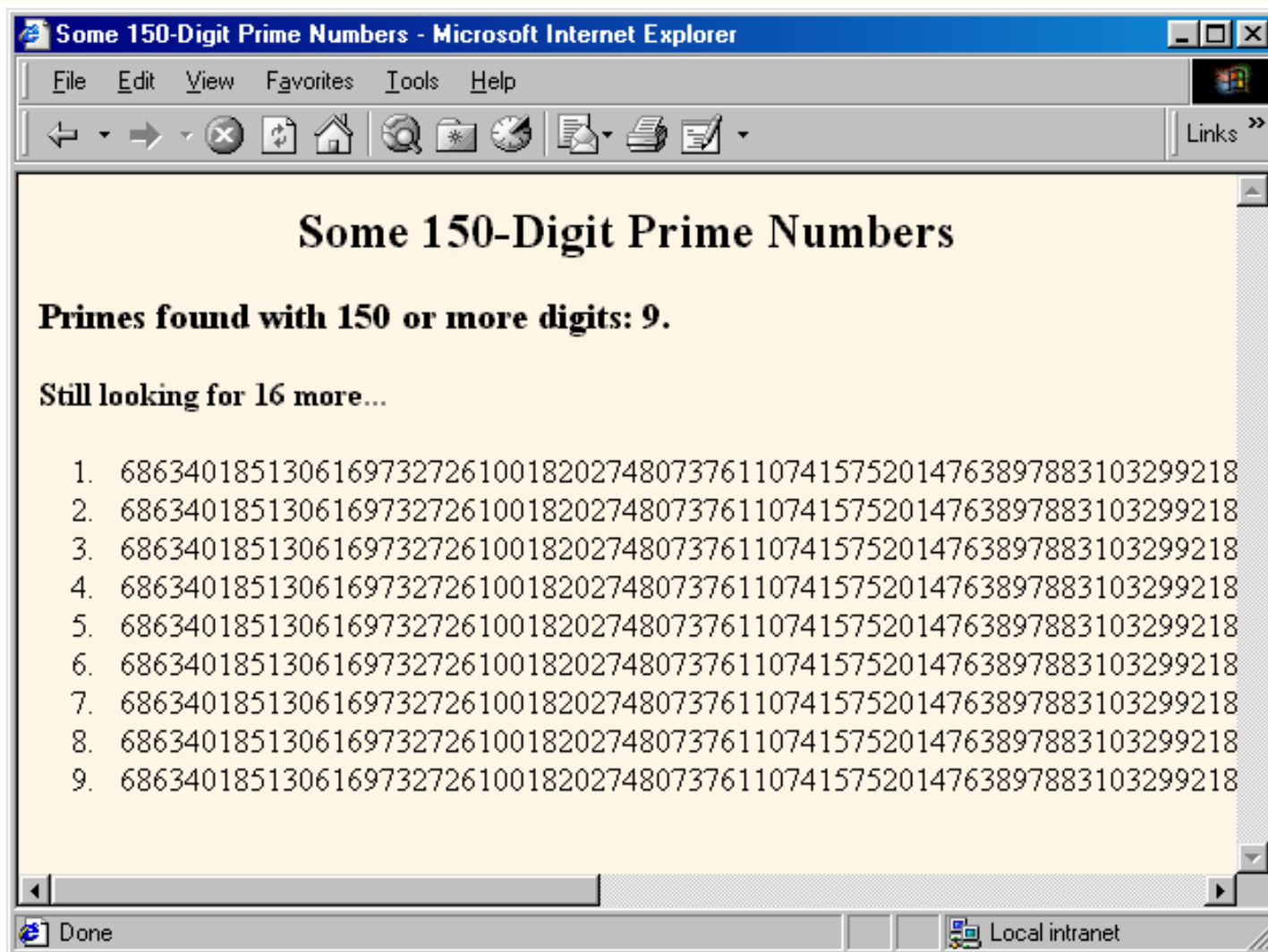
# Generating Prime Numbers: Source Code (Continued)

```
synchronized(primeListVector) {
    if (primeListVector.size() >= maxPrimeLists)
        primeListVector.removeElementAt(0);
    primeListVector.addElement(primeList);
}
}
Vector currentPrimes = primeList.getPrimes();
int numCurrentPrimes = currentPrimes.size();
int numPrimesRemaining = (numPrimes - numCurrentPrimes);
boolean isLastResult = (numPrimesRemaining == 0);
if (!isLastResult) {
    response.setHeader("Refresh", "5");
}
response.setContentType("text/html");
PrintWriter out = response.getWriter();
// Show List of Primes found ...
```

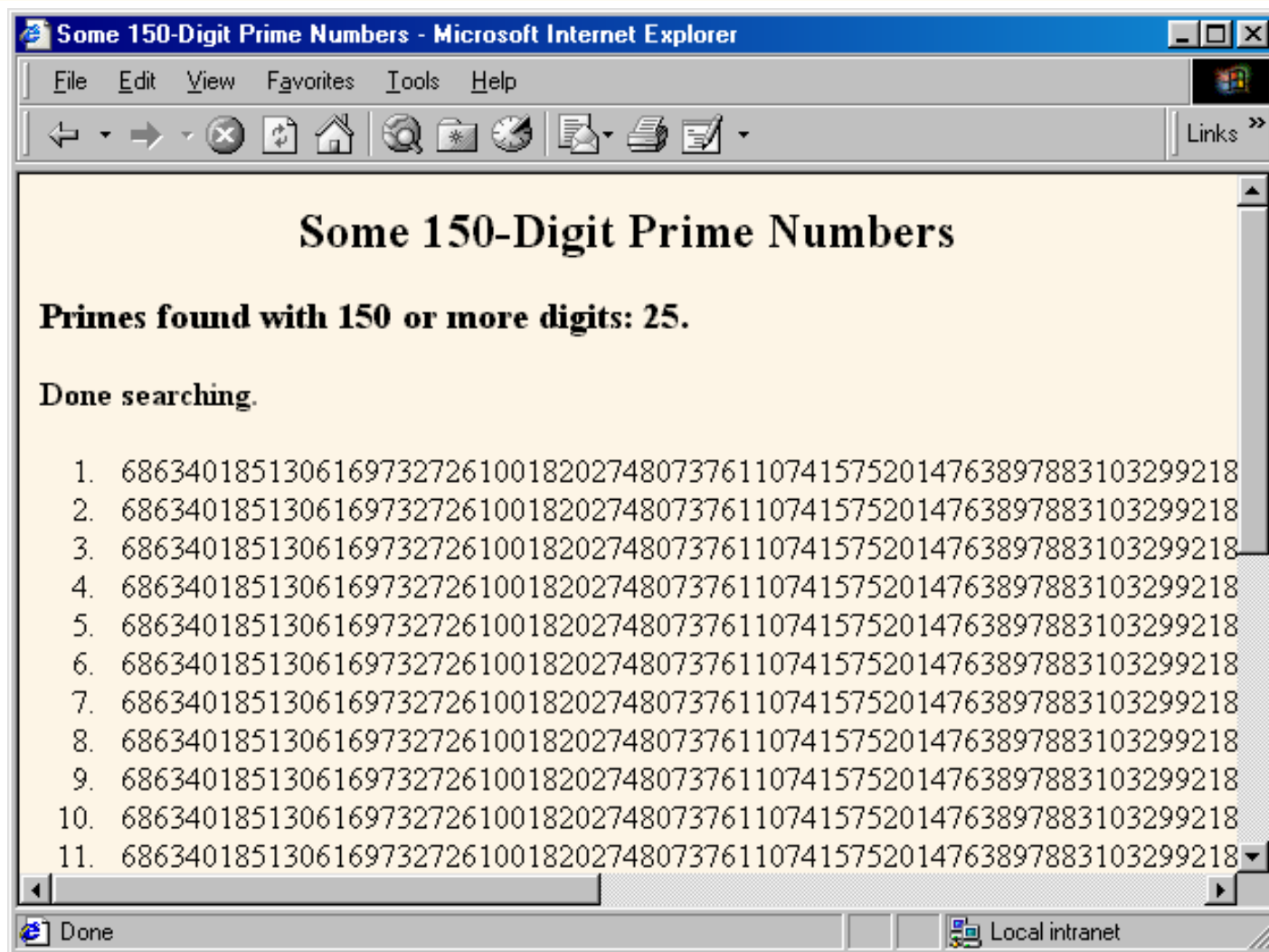
# Prime Number Servlet: Front End

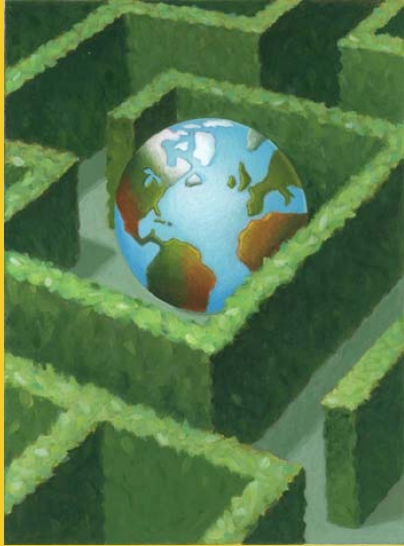


# Prime Number Servlet: Initial Result



# Prime Number Servlet: Final Result





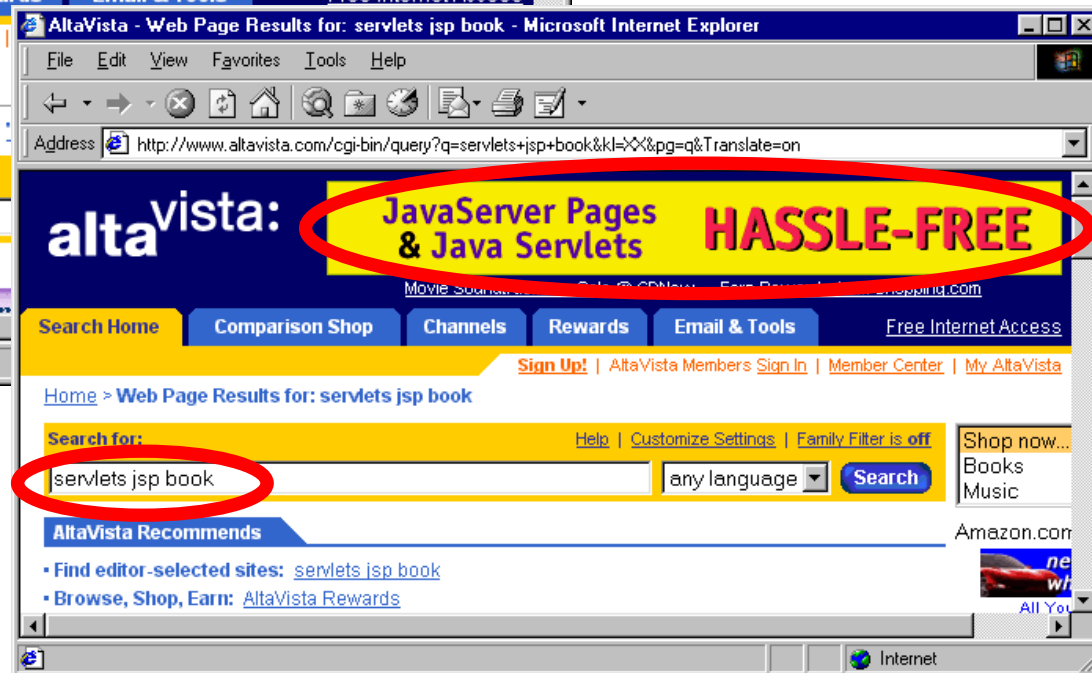
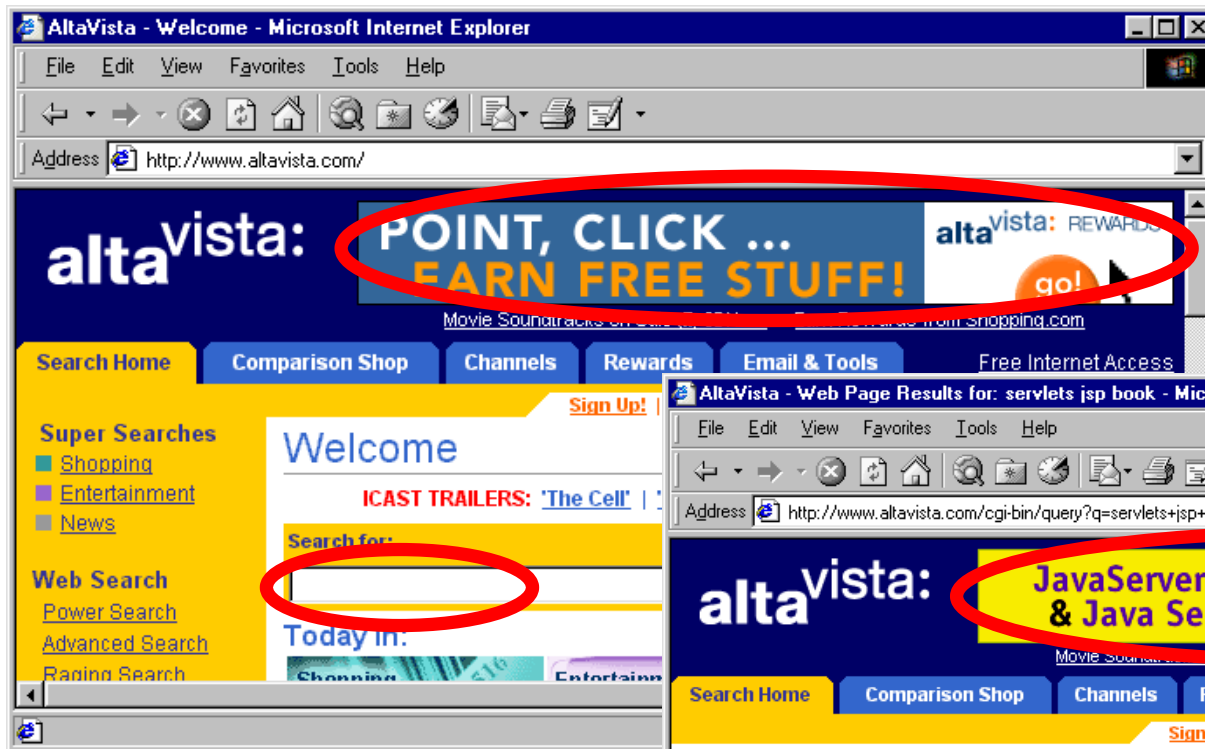
*core*  
**WEB**  
*programming*

# Handling Cookies

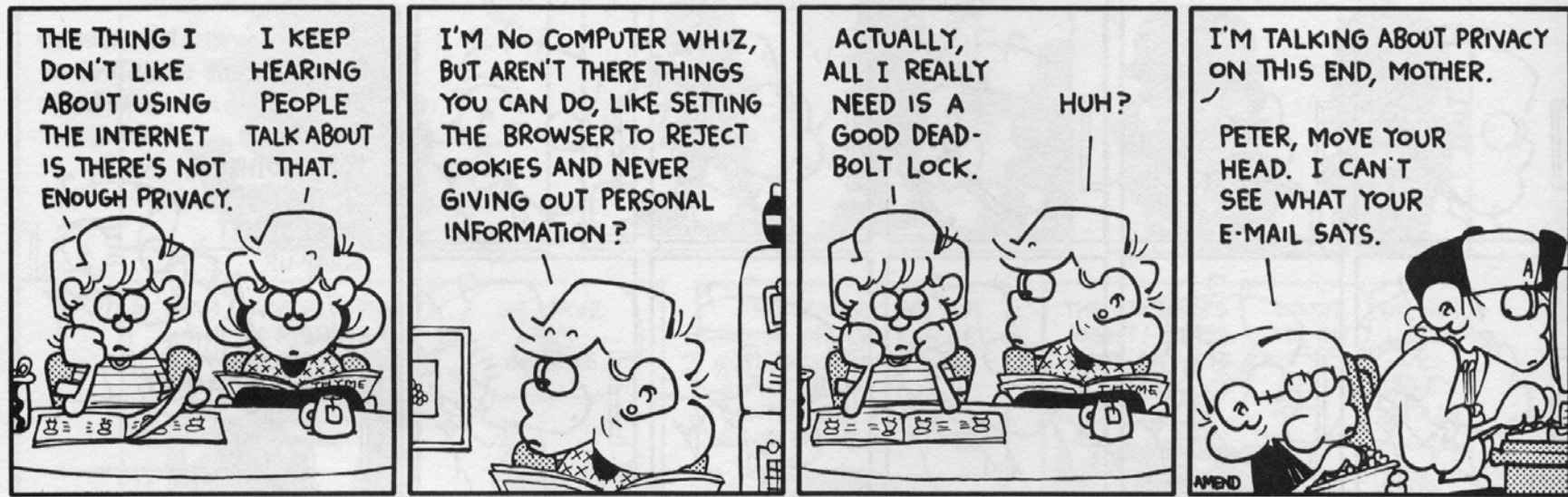
# The Potential of Cookies

- **Idea**
  - Servlet sends a simple name and value to client
  - Client returns same name and value when it connects to same site (or same domain, depending on cookie settings)
- **Typical Uses of Cookies**
  - Identifying a user during an e-commerce session
    - Servlets have a higher-level API for this task
  - Avoiding username and password
  - Customizing a site
  - Focusing advertising

# Cookies and Focused Advertising



# Cookies and Privacy



- FoxTrot © 1998 Bill Amend. Reprinted with permission of Universal Press Syndicate. All rights reserved.

# Some Problems with Cookies

- **The problem is *privacy*, not *security***
  - Servers can remember your previous actions
  - If you give out personal information, servers can link that information to your previous actions
  - Servers can share cookie information through use of a cooperating third party like doubleclick.net
  - Poorly designed sites store sensitive information like credit card numbers directly in cookie
- **Morals for servlet authors**
  - If cookies are not critical to your task, avoid servlets that totally fail when cookies are disabled
  - Don't put sensitive info in cookies

# Sending Cookies to Browser

- **Standard approach:**

```
Cookie c = new Cookie("name", "value");
c.setMaxAge(...); // Means cookie persists on disk
// Set other attributes.
response.addCookie(c);
```

- **Simplified approach:**

- Use LongLivedCookie class:

```
public class LongLivedCookie extends Cookie {
    public static final int SECONDS_PER_YEAR =
        60*60*24*365;

    public LongLivedCookie(String name, String value) {
        super(name, value);
        setMaxAge(SECONDS_PER_YEAR);
    }
}
```

# Reading Cookies from Browser

- **Standard approach:**

```
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for(int i=0; i<cookies.length; i++) {
        Cookie c = cookies[i];
        if (c.getName().equals("someName")) {
            doSomethingWith(c);
            break;
        }
    }
}
```

- **Simplified approach:**

- Extract cookie or cookie value from cookie array by using `ServletUtilities.getCookieValue` or `ServletUtilities.getCookie`

# ServletUtilities.getCookieValue

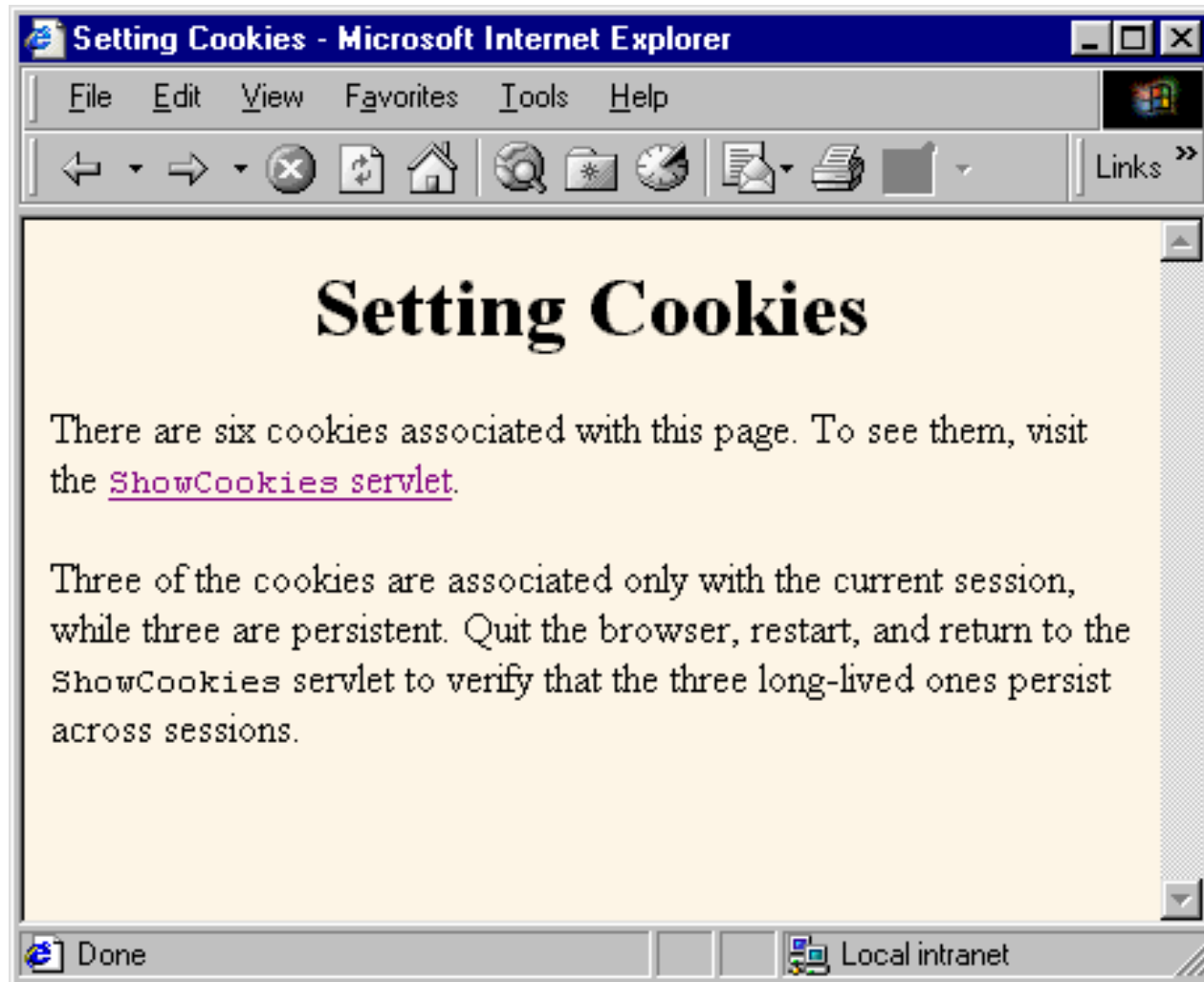
```
public static String getCookieValue(Cookie[] cookies,
                                    String cookieName,
                                    String defaultVal) {
    if (cookies != null) {
        for(int i=0; i<cookies.length; i++) {
            Cookie cookie = cookies[i];
            if (cookieName.equals(cookie.getName()))
                return(cookie.getValue());
        }
    }
    return(defaultVal);
}
```

- **The getCookie method is similar**
  - Returns the Cookie object instead of the value

# Simple Cookie-Setting Servlet

```
public class SetCookies extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        for(int i=0; i<3; i++) {
            Cookie cookie = new Cookie("Session-Cookie-" + i,
                                       "Cookie-Value-S" + i);
            response.addCookie(cookie);
            cookie = new Cookie("Persistent-Cookie-" + i,
                               "Cookie-Value-P" + i);
            cookie.setMaxAge(3600);
            response.addCookie(cookie);
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println(...);
    }
}
```

# Result of Cookie-Setting Servlet



# Simple Cookie-Viewing Servlet

```
public class ShowCookies extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Active Cookies";
        out.println(ServletUtilities.headWithTitle(title) +
                   "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                   "<H1 ALIGN=\"CENTER\">" + title +
                   "</H1>\n" +
                   "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
                   "<TR BGCOLOR=\"#FFAD00\">\n" +
                   "    <TH>Cookie Name\n" +
                   "    <TH>Cookie Value");
    }
}
```

# Simple Cookie-Viewing Servlet (Continued)

```
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    Cookie cookie;
    for(int i=0; i<cookies.length; i++) {
        cookie = cookies[i];
        out.println("<TR>\n" +
            "    <TD>" + cookie.getName() + "\n" +
            "    <TD>" + cookie.getValue());
    }
}
out.println("</TABLE></BODY></HTML>");
}
```


# Result of Cookie-Viewer (Before & After Restarting Browser)



The screenshot shows the 'Active Cookies' page in Microsoft Internet Explorer. The page title is 'Active Cookies'. Below the title is a table with two columns: 'Cookie Name' and 'Cookie Value'. The table contains six rows of cookies:

Cookie Name	Cookie Value
Session-Cookie-0	Cookie-Value-S0
Persistent-Cookie-0	Cookie-Value-P0
Session-Cookie-1	Cookie-Value-S1
Persistent-Cookie-1	Cookie-Value-P1
Session-Cookie-2	Cookie-Value-S2
Persistent-Cookie-2	Cookie-Value-P2

The status bar at the bottom shows 'Done' and 'Local intranet'.



The screenshot shows the 'Active Cookies' page in Microsoft Internet Explorer after restarting the browser. The page title is 'Active Cookies'. Below the title is a table with two columns: 'Cookie Name' and 'Cookie Value'. The table contains three rows of cookies:

Cookie Name	Cookie Value
Persistent-Cookie-0	Cookie-Value-P0
Persistent-Cookie-1	Cookie-Value-P1
Persistent-Cookie-2	Cookie-Value-P2

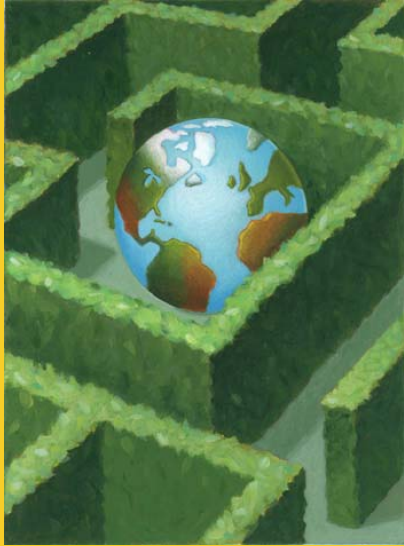
The status bar at the bottom shows 'Done' and 'Local intranet'.

# Methods in the Cookie API

- **getDomain/setDomain**
  - Lets you specify domain to which cookie applies. Current host must be part of domain specified
- **getMaxAge/setMaxAge**
  - Gets/sets the cookie expiration time (in seconds). If you fail to set this, cookie applies to current browsing session only. See LongLivedCookie helper class given earlier
- **getName/setName**
  - Gets/sets the cookie name. For new cookies, you supply name to constructor, not to setName. For incoming cookie array, you use getName to find the cookie of interest

# Methods in the Cookie API (Continued)

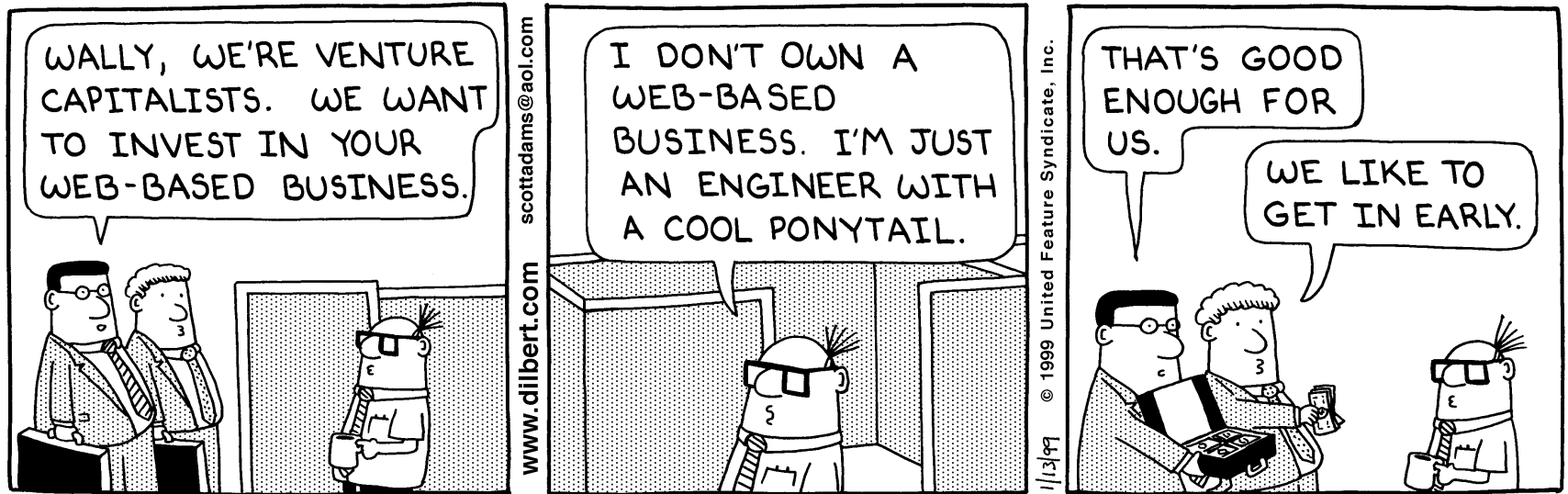
- **getPath/setPath**
  - Gets/sets the path to which cookie applies. If unspecified, cookie applies to URLs that are within or below directory containing current page
- **getSecure/setSecure**
  - Gets/sets flag indicating whether cookie should apply only to SSL connections or to all connections
- **getValue/setValue**
  - Gets/sets value associated with cookie. For new cookies, you supply value to constructor, not to setValue. For incoming cookie array, you use getName to find the cookie of interest, then call getValue on the result



*core*  
**WEB**  
*programming*

# Session Tracking

# Session Tracking and E-Commerce



Dilbert used with permission of United Syndicates Inc.

# Session Tracking

- **Why?**

- When clients at an on-line store add an item to their shopping cart, how does the server know what's already in the cart?
- When clients decide to proceed to checkout, how can the server determine which previously created shopping cart is theirs?

- **How?**

- Cookies
- URL-rewriting
- Hidden form fields

- **Higher-level API needed**

# The Session Tracking API

- **Session objects live on the server**
- **Automatically associated with client via cookies or URL-rewriting**
  - Use `request.getSession(true)` to get either existing or new 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

# Using Sessions

```
HttpSession session = request.getSession(true);
ShoppingCart cart =
    (ShoppingCart) session.getAttribute("shoppingCart");
if (cart == null) { // No cart already in session
    cart = new ShoppingCart();
    session.setAttribute("shoppingCart", cart);
}
doSomethingWith(cart);
```

# HttpSession Methods

- **getAttribute, getValue [2.1]**
  - Extracts a previously stored value from a session object. Returns null if no value is associated with given name
- **setAttribute, putValue [2.1]**
  - Associates a value with a name. Monitor changes: values implement HttpSessionBindingListener.
- **removeAttribute, removeValue [2.1]**
  - Removes values associated with name
- **getAttributeNames, getValueNames [2.1]**
  - Returns names of all attributes in the session
- **getId**
  - Returns the unique identifier

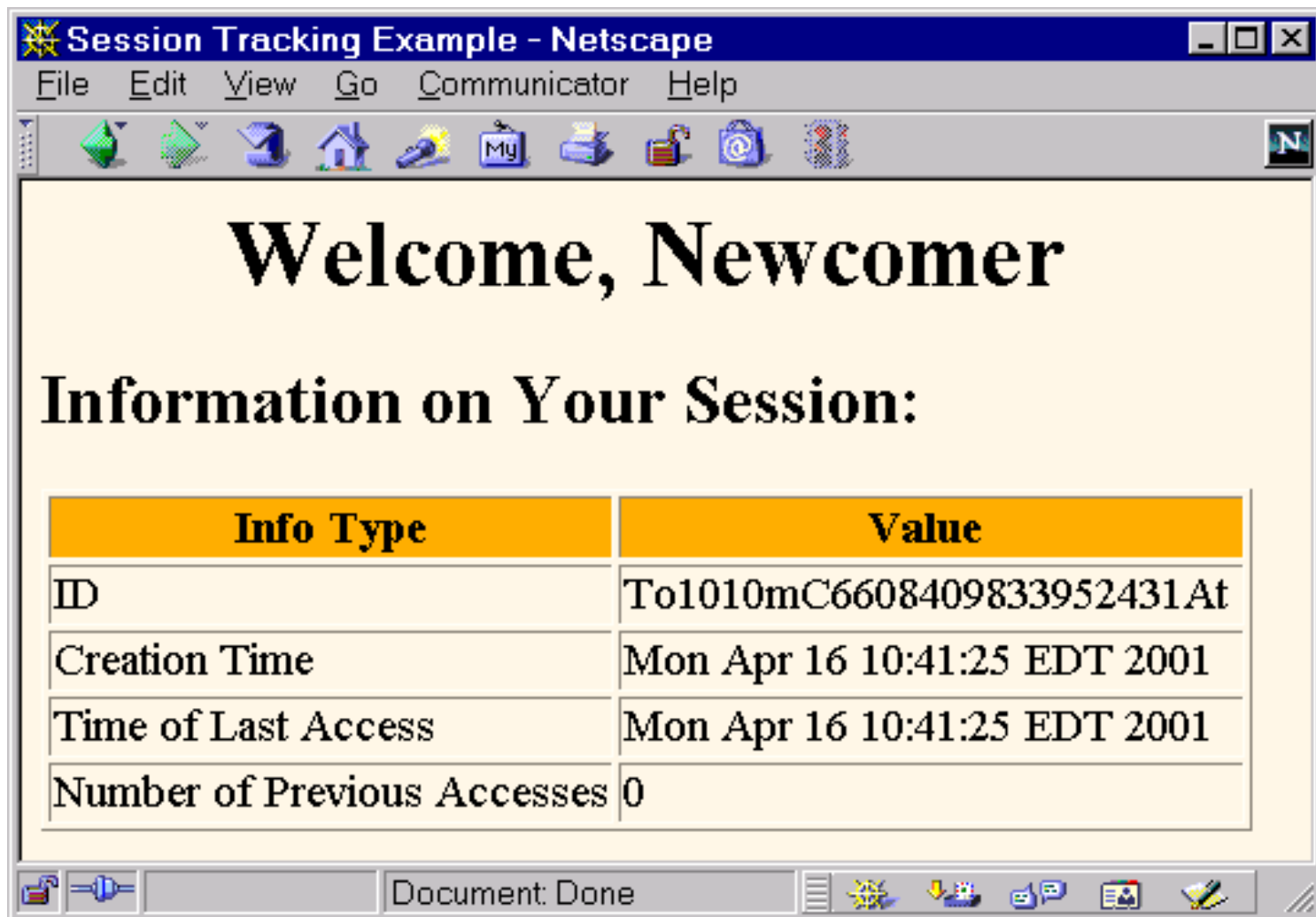
# HttpSession Methods (Continued)

- **isNew**
  - Determines if session is new to client (not to page)
- **getCreationTime**
  - Returns time at which session was first created
- **getLastAccessedTime**
  - Returns time session was last sent from client
- **getMaxInactiveInterval, setMaxInactiveInterval**
  - Gets or sets the amount of time session should go without access before being invalidated
- **invalidate**
  - Invalidates the session and unbinds all objects associated with it

# A Servlet Showing Per-Client Access Counts

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Session Tracking Example";
    HttpSession session = request.getSession(true);
    String heading;
    Integer accessCount =
        (Integer) session.getAttribute("accessCount");
    if (accessCount == null) {
        accessCount = new Integer(0);
        heading = "Welcome, Newcomer";
    } else {
        heading = "Welcome Back";
        accessCount = new Integer(accessCount.intValue() + 1);
    }
    session.setAttribute("accessCount", accessCount);
}
```

# First Visit to ShowSession Servlet



The screenshot shows a Netscape browser window titled "Session Tracking Example - Netscape". The address bar is empty. The main content area displays a welcome message and session information. The session information is presented in a table with two columns: "Info Type" and "Value".

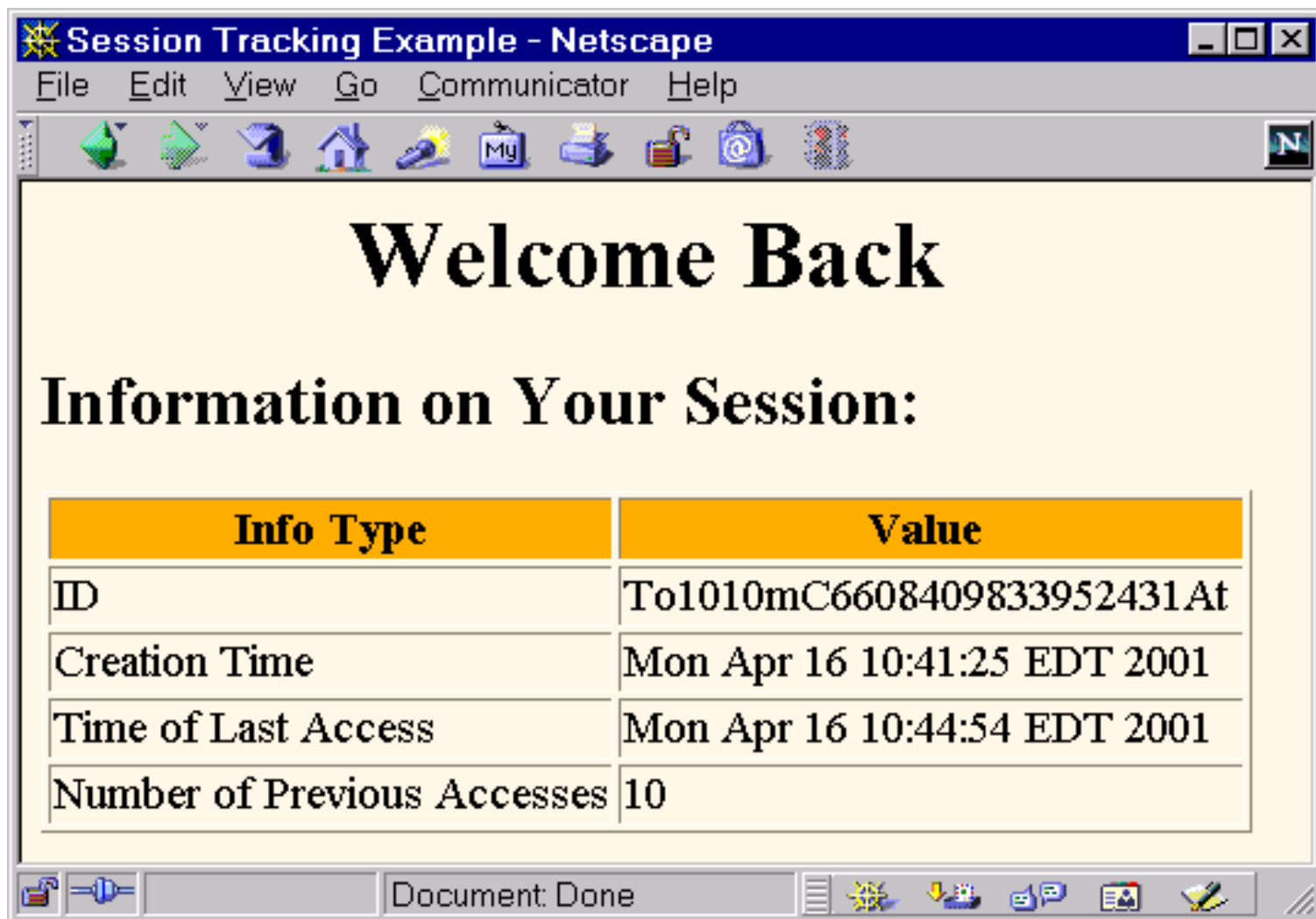
**Welcome, Newcomer**

**Information on Your Session:**

Info Type	Value
ID	To1010mC6608409833952431At
Creation Time	Mon Apr 16 10:41:25 EDT 2001
Time of Last Access	Mon Apr 16 10:41:25 EDT 2001
Number of Previous Accesses	0

The browser's status bar at the bottom shows "Document: Done" and various navigation icons.

# Eleventh Visit to ShowSession Servlet

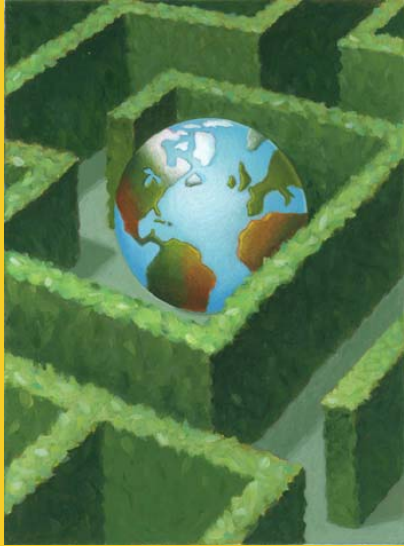


The screenshot shows a Netscape browser window titled "Session Tracking Example - Netscape". The browser's address bar is empty, and the status bar at the bottom indicates "Document: Done". The main content area displays a "Welcome Back" message and a table of session information.

**Welcome Back**

**Information on Your Session:**

Info Type	Value
ID	To1010mC6608409833952431At
Creation Time	Mon Apr 16 10:41:25 EDT 2001
Time of Last Access	Mon Apr 16 10:44:54 EDT 2001
Number of Previous Accesses	10



*core*  
**WEB**  
*programming*

# Review: Servlets

# Review: Getting Started

- **Servlets are efficient, portable, powerful, and widely accepted in industry**
- **Regardless of deployment server, run a free server on your desktop for development**
- **Getting started:**
  - Set your CLASSPATH
    - Servlet and JSP JAR files
    - Top of your package hierarchy
  - Put class files in proper location
    - .../WEB-INF/classes with servlets 2.2
  - Use proper URL; default is `http://host/servlet/ServletName`
- **Download existing servlet first time**
  - Start with HelloWWW from [www.corewebprogramming.com](http://www.corewebprogramming.com)

# Review: Getting Started (Continued)

- **Main servlet code goes in doGet or doPost:**
  - The `HttpServletRequest` contains the incoming information
  - The `HttpServletResponse` lets you set outgoing information
    - Call `setContentType` to specify MIME type
    - Call `getWriter` to obtain a `Writer` pointing to client
- **One-time setup code goes in init**
  - Servlet gets initialized and loaded once
  - Servlet gets invoked multiple times

# Review: Handling Form Data (Query Data)

- **Query data comes from HTML forms as URL-encoded name/value pairs**
- **Servlets read data by calling `request.getParameter("name")`**
  - Results in value as entered into form, not as sent over network. I.e. not URL-encoded.
- **Always check for missing or malformed data**
  - Special case: query data that contains special HTML characters
    - Need to be filtered if query data will be placed into resultant HTML page

# Review:

## Using HTTP Request Headers

- Many servlet tasks can only be accomplished by making use of HTTP headers coming from the browser
- Use `request.getHeader` for arbitrary header
- Cookies, authorization info, content length, and content type have shortcut methods
- Most important headers you read directly
  - Accept
  - Accept-Encoding
  - Connection
  - Referer
  - User-Agent

# Review:

## Generating the HTTP Response

- **Many servlet tasks can only be accomplished through use of HTTP status codes and headers sent to the browser**
- **Two parts of the response**
  - Status line
    - In general, set via `response.setStatus`
    - In special cases, set via `response.sendRedirect` and `response.sendError`
  - Response headers
    - In general, set via `response.setHeader`
    - In special cases, set via `response.setContentType`, `response.setContentLength`, `response.addCookie`, and `response.sendRedirect`

# Review: Generating the HTTP Response (Continued)

- **Most important status codes**
  - 200 (default)
  - 302 (forwarding; set via `sendRedirect`)
  - 401 (password needed)
  - 404 (not found; set via `sendError`)
- **Most important headers you set directly**
  - Cache-Control and Pragma
  - Content-Encoding
  - Content-Length
  - Expires
  - Refresh
  - WWW-Authenticate

# Review: Handling Cookies

- **Cookies involve name/value pairs sent from server to browser and returned when the same page, site, or domain is visited later**
- **Let you**
  - Track sessions (use higher-level API)
  - Permit users to avoid logging in at low-security sites
  - Customize sites for different users
  - Focus content or advertising
- **Setting cookies**
  - Cookie constructor, set age, `response.addCookie`
- **Reading cookies**
  - Call `request.getCookies`, check for null, look through array for matching name, use associated value

# Review: Session Tracking

- **Although it usually uses cookies behind the scenes, the session tracking API is higher-level and easier to use than the cookie API**
- **Session information lives on server**
  - Cookie or extra URL info associates it with a user
- **Obtaining session**
  - `request.getSession(true)`
- **Associating values with keys**
  - `session.setAttribute`
- **Finding values associated with keys**
  - `session.getAttribute`
    - Always check if this value is null before trying to use it

# Preview: The Need for JSP

- **With servlets, it is easy to**
  - Read form data
  - Read HTTP request headers
  - Set HTTP status codes and response headers
  - Use cookies and session tracking
  - Share data among servlets
  - Remember data between requests
  - Get fun, high-paying jobs
- **But, it sure is a pain to**
  - Use those println statements to generate HTML
  - Maintain that HTML

# Preview: Benefits of JSP

- **Although JSP technically can't do anything servlets can't do, JSP makes it easier to:**
  - Write HTML
  - Read and maintain the HTML
- **JSP makes it possible to:**
  - Use standard HTML tools such as HomeSite or UltraDev
  - Have different members of your team do the HTML layout and the programming
- **JSP encourages you to**
  - Separate the (Java™ technology) code that creates the content from the (HTML) code that presents it

# More Information

- **Core Servlets and JavaServer Pages**

- <http://www.coreservlets.com>
- More detail on all topics presented here

- **More Servlets and JavaServer Pages**

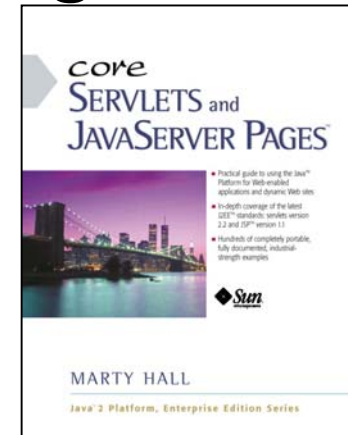
- <http://www.moreservlets.com>
- New features in servlets 2.3 and JSP 1.2 (filters and listeners), Web applications, security, standard JSP tag library

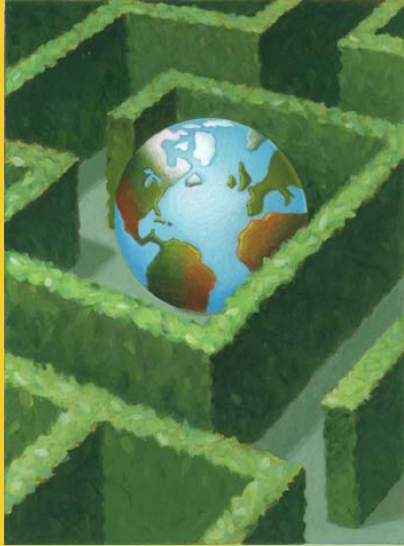
- **Servlet home page**

- <http://java.sun.com/products/servlet/>

- **JavaServer Pages home page**

- <http://java.sun.com/products/jsp/>





*core*  
**WEB**  
*programming*

**Questions?**