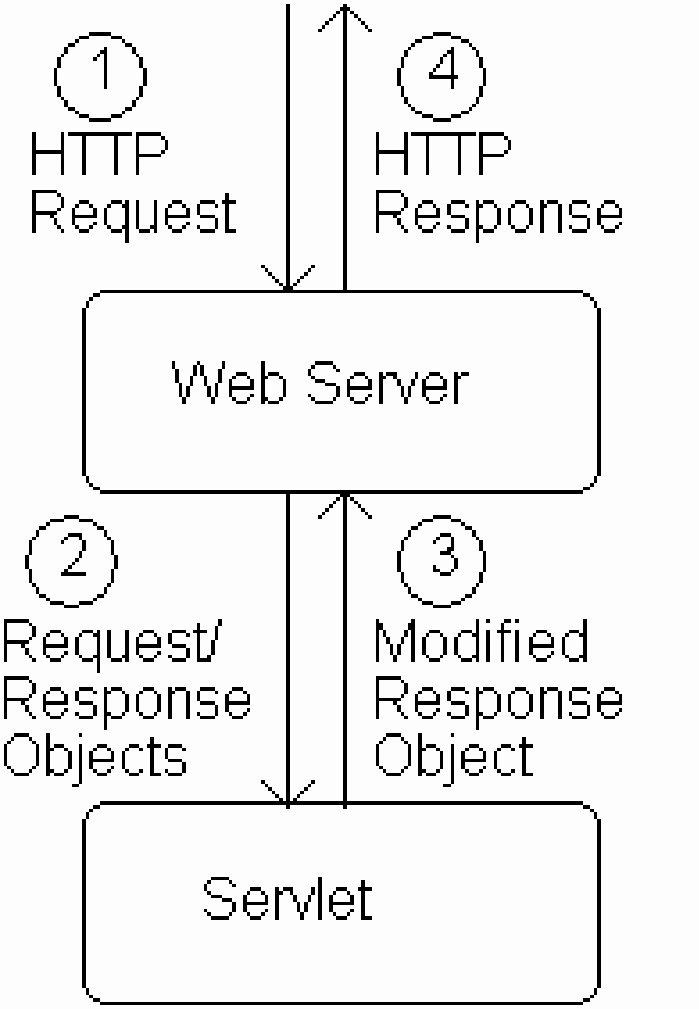
Web Engineering
Instructor: Wasim Ahmad Khan

Server-side Programming: Java Servlets

- Web server response can be static or dynamic
  - Static: When browser has requested an HTML document
  - HTML document is retrieved from the file system and returned to the client
  - Web server not responsible for generating content of response
  - Find and send the content
  - Dynamic: HTML document is generated by a program in response to an HTTP request
  - Eg: Visiting a search engine website

- Java servlets are one technology for producing dynamic server responses
- Servlet is a Java class instantiated by the server to produce a dynamic response
- A particular method is called on this instance when server receives HTTP request
- Code in the servlet method can obtain information about the request and produce information to be included in the response
- It is done by calling methods on parameter objects passed to the method
- When the servlet returns control to the server , it creates a response from the information dynamically generated by servlet

Web Server--Servlet Interaction



Web server Operation

1. When an HTTP request is received by a servlet capable server
   1. It determines based on URL whether the request handled by the servlet
   2. Any URL in which the path component begins with servlet
2. The servlet determines from URL which servlet handle the request and calls a method on that servlet
   - Two parameters are passed to the method
   - An object implementing the HttpservletRequest interface
   - An object implementing the HttpservletResponse interface
   - Both are defined as part of Java Servlet API which is implemented by server

- First method to access the information contained in the request message
- Second object to record the information that the servlet wishes to include in the HTTP response message

3. Servlet method executes by calling methods on HttpservletRequest and HttpservletResponse objects passed to it.
   - The information stored in the latter object by the servlet method includes an HTML document along with some HTTP header information
   - When servlet method finishes its processing it returns control to the server
4. The server formats the information stored in the HttpservletResponse object by the servlet into an HTTP response message then send to the client

A Hello World! Servlet

Java Servlet name: ServletHello
Responds with HTML document “Hello World!” in response to HTTP GET Request

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
        {
          // Set the HTTP content type in response header
          response.setContentType("text/html; charset=\"UTF-8\"");

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();
```

All servlets we will write are subclasses of `HttpServlet`

ServletHello.java - Notepad

File Edit Format View Help

ServletHello.java

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

/**
 * Hello World! servlet
 */
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
        {
          // Set the HTTP content type in response header
          response.setContentType("text/html; charset=\"UTF-8\"");

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();

          // Create the body of the response
          servletOut.println(
"<!DOCTYPE html \n" +
"    PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\" \n" +
"    \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\"> \n" +
"<html xmlns='http://www.w3.org/1999/xhtml'> \n" +
"  <head> \n" +
"    <title> \n" +
"      ServletHello.java \n" +
"    </title> \n" +
"  </head> \n" +
"  <body> \n" +
"    <p> \n" +
"       Hello World! \n" +
"    </p> \n" +
"  </body> \n" +
```

Ln 43, Col 10

C:\Documents an... H:\ Microsoft PowerP... ServletHello.java -... 10:53

Hello World! Servlet

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
        {
          // Set the HTTP content type in response header
          response.setContentType("text/html; charset=\"UTF-8\"");

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();
```

Server calls doGet() in response to GET request

Interfaces implemented by request/response objects

Hello World! Servlet

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
        {
          // Set the HTTP content type in response header
          response.setContentType("text/html; charset=\"UTF-8\"");

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();
```

Production servlet should catch these exceptions

A Production Servlet normally catch all exceptions internally rather than throwing to the server

(**Java Web Services Developer Pack**)

- JWSDP Tomcat server exception handling:
  - Writing a trace for the exception in logs/jwsdp_log.*.txt log file
  - Returning HTML page to client that may (or may not) contain partial exception trace
- If servlet prints a stack trace itself by calling printStackTrace(), or if it writes debugging output to System.out or System.err, this output will be appended to the file logs/launcher.server.log
- **printStackTrace(),** The trace prints exactly where the program was at the moment the exception was thrown

Body of a doGet() method action

- Set the HTTP content-type header of the response
  - MIME Type portion of this header is text/html
  - Also include the type of character encoding used
- Obtain PrinterWriter object from HttpservletResponse object by calling getwriter() method
  - It should not be called before the Content-Type is set
  - It is set by setContentType()
- Output a valid HTML document to the PrinterWriter object
- Close the PrinterWriter object

Hello World! Servlet

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
        {
          // Set the HTTP content type in response header
          response.setContentType("text/html; charset=\"UTF-8\"");

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();
```

First two things done by typical servlet; must be in this order

Hello World! Servlet

```
        // Create the body of the response
        servletOut.println(
"<!DOCTYPE html \n" +
"    PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\" \n" +
"    \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\"> \n" +
"<html xmlns='http://www.w3.org/1999/xhtml'> \n" +
"  <head> \n" +
"    <title> \n" +
"      ServletHello.java \n" +
"    </title> \n" +
"  </head> \n" +
"  <body> \n" +
"    <p> \n" +
"       Hello World! \n" +
"    </p> \n" +
"  </body> \n" +
"</html> ");
        servletOut.close();
      }
}
```

HTML generated by calling `print()` or `println()` on the servlet's `PrintWriter` object

Good practice to explicitly close the `PrintWriter` when done

Servlets vs. Java Applications

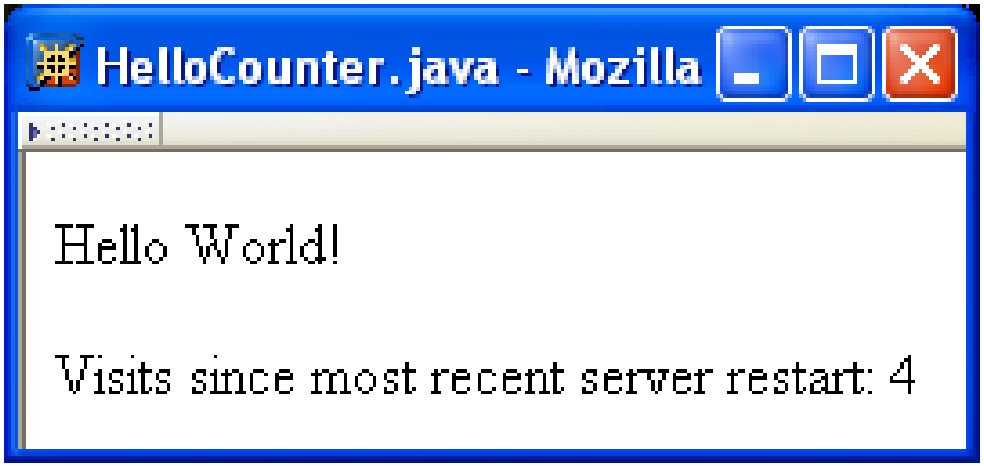
- Servlets do not have a `main()`
  - The `main()` is in the server (doGet())
  - Entry point to servlet code is via call to a method (`doGet()` in the example)
- Servlet interaction with end user is indirect via request/response object APIs
  - Actual HTTP request/response processing is handled by the server
- Primary servlet output is typically HTML

Running Servlets

- Simple way to run a servlet
  1. Compile servlet
  2. Copy `.class` file to `shared/classes` directory MALE
  3. (Re)start the Tomcat web server
  4. If the class is named `ServletHello`, browse to http://localhost:8080/servlet/ServletHello

Servlet Generating Dynamic Content

- The previous one produces static HTML File
- This prints Hello World and no of times the servlet has been visited since the servlet was started
- Page is no longer static



- A counter variable visits will be incremented
- Its value output as part of the HTML document produced by the servlet

```
public class HelloCounter extends HttpServlet
{
    // Number of times the servlet has been executed since
    // the program (web server) started
    private int visits=0;

[...] // removed doGet() declaration and initialization

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();

          // Compute the number of visits to the URL for this servlet
          visits++;
```

```
"    <p> \n" +
"       Hello World! \n" +
"    </p> \n" +
"    <p> \n" +
"       This page has been viewed \n" +
          visits +
"       times since the most recent server restart. \n" +
"    </p> \n" +
```

- Potential problems:
  - Assuming one instance of servlet on one server, but
    - Many Web sites are distributed over multiple servers
    - Even a single server can (not default) create multiple instances of a single servlet
    - If multiple users access this servlet at nearly the same time,the multiple execution of doGet() causes different users to see the same visit count.
  - Even if the assumption is correct, this servlet does not handle concurrent accesses properly

Servlet Life Cycle

- The java servlet API provide initialization tasks(opening files, establishing database connections)
- Servlet API life cycle methods
  - init(): called when servlet is instantiated; must return before any other methods will be called
    - If intialization processing causes error it throw throw an exception called UnavailableException
  - service(): method called directly by server when an HTTP request is received; this in turn calls doGet()
  - destroy(): called when server shuts down(taking a servlet out of service
    - Servlet to terminate cleanly or closing any database connection and opened files

Servlet Life Cycle

```
public void init()
    throws ServletException
{
    try {
        BufferedReader br =
            new BufferedReader(new FileReader("aFile"));
        visits = (new Integer(br.readLine())).intValue();
    }
    catch (FileNotFoundException fnfe) {
        throw new UnavailableException("File not found: " +
                                       fnfe.toString());
    }
    catch (Exception e) {
        throw new UnavailableException("Data problem: " +
                                       e.toString());
    }
}
```

Example life cycle method: attempt to initialize `visits` variable from file

Servlet Life Cycle

```
public void init()
    throws ServletException
{
    try {
        BufferedReader br =
            new BufferedReader(new FileReader("aFile"));
        visits = (new Integer(br.readLine())).intValue();
    }
    catch (FileNotFoundException fnfe) {
        throw new UnavailableException("File not found: " +
                                       fnfe.toString());
    }
    catch (Exception e) {
        throw new UnavailableException("Data problem: " +
                                       e.toString());
    }
}
```

Exception to be thrown if initialization fails and servlet should not be instantiated

Parameter Data

•The request object (which implements HttpServletRequest) provides information from the HTTP request to the servlet

•The most frequently used potion of the HTTP request called Parameter Data of the request

Parameter Data and Query Strings

• When we navigate to a URL the server calls doGet() method of servlet for us

•It returns a long string the return value of the method

•One type of information is parameter data, which is information from the query string portion of the HTTP request

- Example

```
http://www.example.com/servlet/PrintThis?arg=aString
```

Query string with one parameter

- ? End of the path portion of URL

  and the beginning of query portion of URI
- Query portion of URL consist of a query string
- Query string contains one parameter called arg assigned a string value astring
- All query string parameter treated as string and they should not be quoted
- Parameter data is the Web analog of calling a method in java

```
System.out.println("aString");
```

```
http://www.example.com/servlet/PrintThis?arg=aString
```

- Query string syntax and semantics
  - Multiple parameters separated by &

```
http://www.example.com/servlet/PrintThis?arg=aString&color=red
```

  - Order of parameters does not matter

```
http://www.example.com/servlet/PrintThis?color=red&arg=aString
```

  - All parameter values are strings

```
http://www.example.com/servlet/PrintThis?arg=&color=red
```

Value of `arg` is empty string

  - The empty string value assigned to a parameter by either following the equals sign after the parameter name with an ampersand

- A parameter name or value can be any sequence of 8-bit characters
- If a name or value contains any non alphanumeric characters then a transformation called URL encoding included in the query string
- URL encoding is used to represent non-alphanumeric characters:

```
http://www.example.com/servlet/PrintThis?arg=%27a+String%27
```

Value of `arg` is ‘a String’

- URL decoding applied by server to retrieve intended name or value
- The above example to send the string ‘a string’ as the parameter value

- URL encoding algorithm

```
initialize the result to the empty string
for each 8-bit character in the original string
  if the character is an alphanumeric
    concatenate the character to the result
  else if the character is a space
    concatenate a plus sign (+) to the result
  else
    concatenate a percent sign (%) followed by
      the two-digit hexadecimal value of the character
      to the result
  endif
endfor
return result
```

Servlet and Parameter data

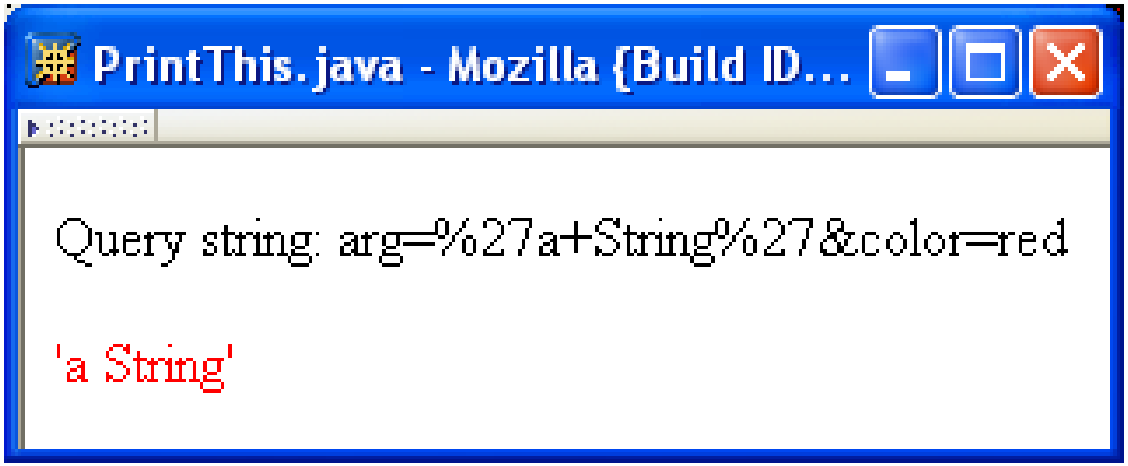
- Query strings can be included in any URL even to a static web page
- For static web page it will be ignored by web server
- The servlet obtain the query string as well as the parameter data by using HttpServletRequest methods

TABLE 6.1: Some `HttpServletRequest` methods for accessing parameter data.

Method	Purpose
`String getQueryString()`	Returns the entire query string in its original (URL encoded) form.
`Enumeration getParameterNames()`	Returns Enumeration of String values representing all parameter names (URL decoded) in the query string.
`String getParameter (String name)`	Returns String representing value (URL decoded) of parameter named `name`, or `null` if parameter is not present in the query string.
`String[] getParameterValues (String name)`	Returns array of String's representing all values (URL decoded) of parameter named `name`, or `null` if parameter is not present in the query string.

Ex for accessing Parameter Data from a servlet

- body of doGet() method of ServletPrintThis creates a web page displaying two paragraphs
- First containing the query string of the URL used to access the servlet
- Second containing either the URL-decoded value of the arg parameter, or the default text “Hello World!”



•Recall that the ampersand (&) and less-than (<) symbols are not allowed to appear in the character data or attribute values of an XHTML document

• The servlet performs this replacement by calling a static method escapeXML(String) that belongs to a class WebTechUtil.

•quote characters should be re-placed by references in a string that will appear as part of the value of an attribute in an XHTML document.

•The escapeQuotes() method of WebTechUtil performs this task

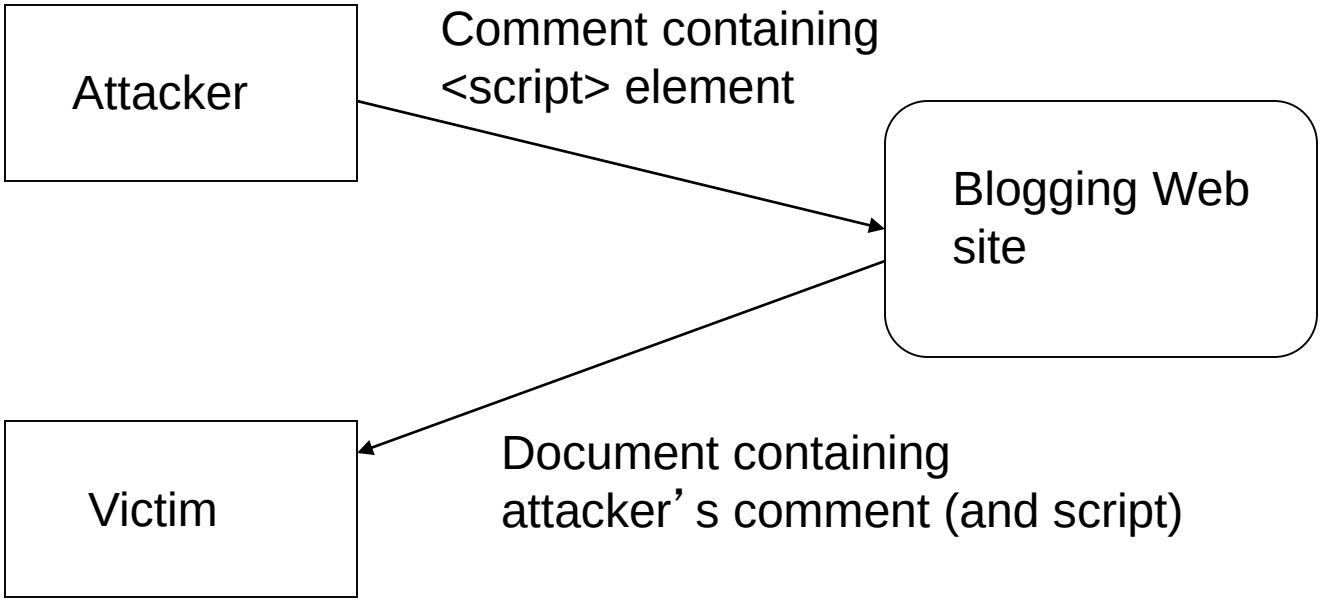
Parameter Data

Must escape XML special characters in all user-supplied data before adding to HTML to avoid *cross-site scripting* attacks

```
"   <body> \n" +
"     <p>Query string: " +
      WebTechUtil.escapeXML(request.getQueryString()) + "</p>" );

          // Decide whether or not to set color
          String color = request.getParameter("color");
          if (color == null) {
              servletOut.println(
"     <p> " );
          } else {
              servletOut.println(
"     <p style='color:" +
              WebTechUtil.escapeQuotes(WebTechUtil.escapeXML(color)) +
"'> " );
          }
```

Also need to escape quotes within attribute values.

- Cross-site scripting



Cross Site Scripting also known as XSS is a popular type of Client Site Attack, It is a type of attack which occurs in Web-Applications and allows an attacker to inject desired client-side scripts into Web-Pages viewed by others.

- **(CROSS-Site Scripting)** Causing a user's Web browser to execute a malicious(wicked) script. There are several ways this is done. One approach is to hide code in a "click here" hyperlink attached to a URL that points to a non-existent Web page. When the page is not found, the script is returned with the bogus(false) URL, and the user's browser executes it.
- clicking these links the Victims Unknowingly executes the injected code , Which in turn can result in Cookie stealing , Privacy Disclosure etc.

Forms and Parameter Data

- A form automatically generates a query string when submitted
  - Parameter name specified by value of `name` attributes of form controls

```
<input type="text" name="username" size="40" />
```

  - Parameter value depends on control type

```
<label>
  <input type="checkbox" name="boxgroup1" value="tall" />tall
</label>
```

Value for checkbox specified by `value` attribute

Parameter Data

TABLE 6.2: Values for HTML form controls (except input/file)

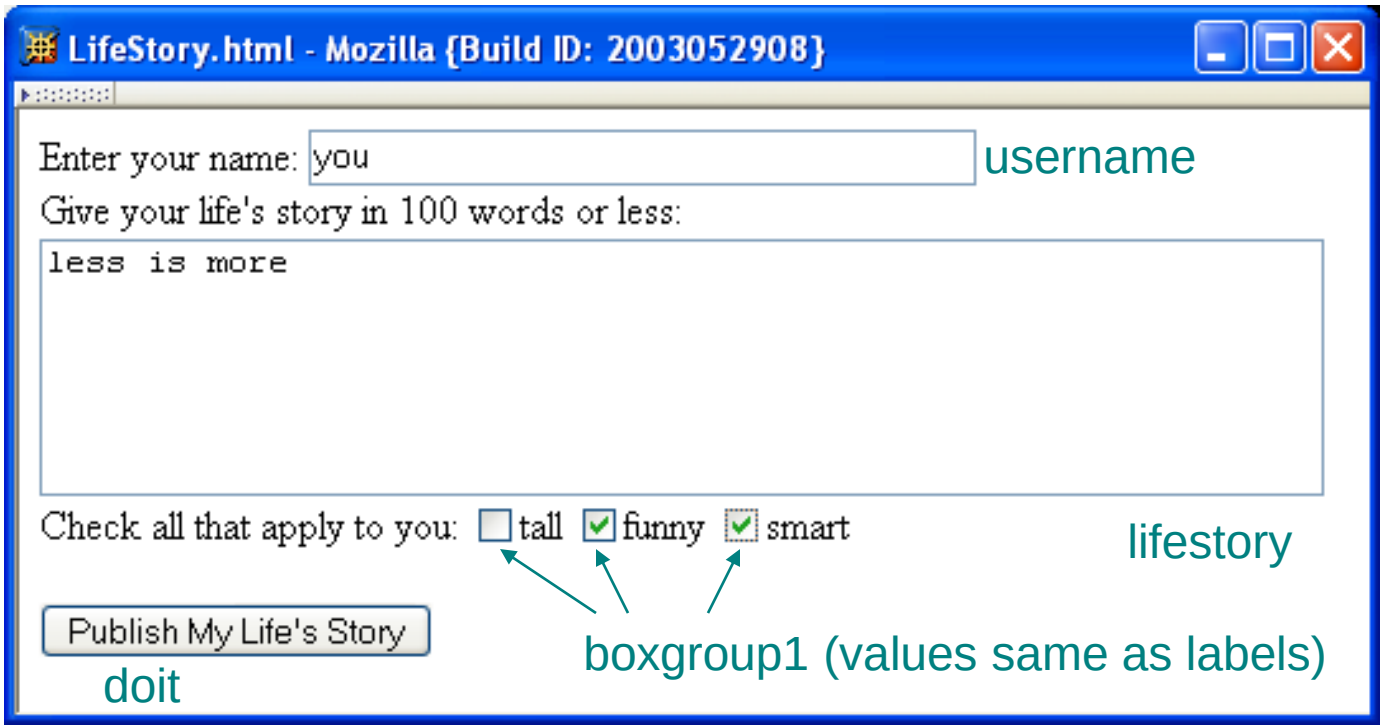
Control(s)	**Value**
`input/text`, `input/password`, `textarea`	text present in control when form is submitted
`input/checkbox`, `input/radio`, `input/submit`, `input/image`, `button/submit`	String assigned to corresponding `value` attribute. Control must be selected/clicked for parameter to be returned.
`input/hidden`	String assigned to corresponding `value` attribute.
`select`	String assigned to `value` attribute of selected `option`(s), or content of any selected `option` for which `value` is not defined.

•**Example:** LifeStory.html form

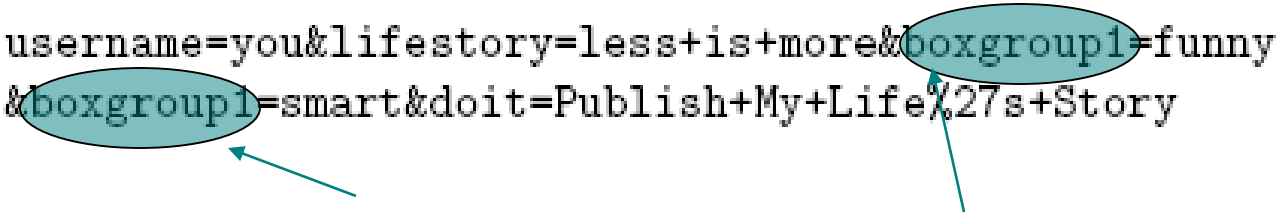
•a text field named username (that is, the value of its name attribute is username)

•a textarea named lifestory

•Three checkboxes all named boxgroup1 and a submit button named doit.



- Query string produced by browser (all one line):



Checkbox parameters have same name values; only checked boxes have corresponding parameters

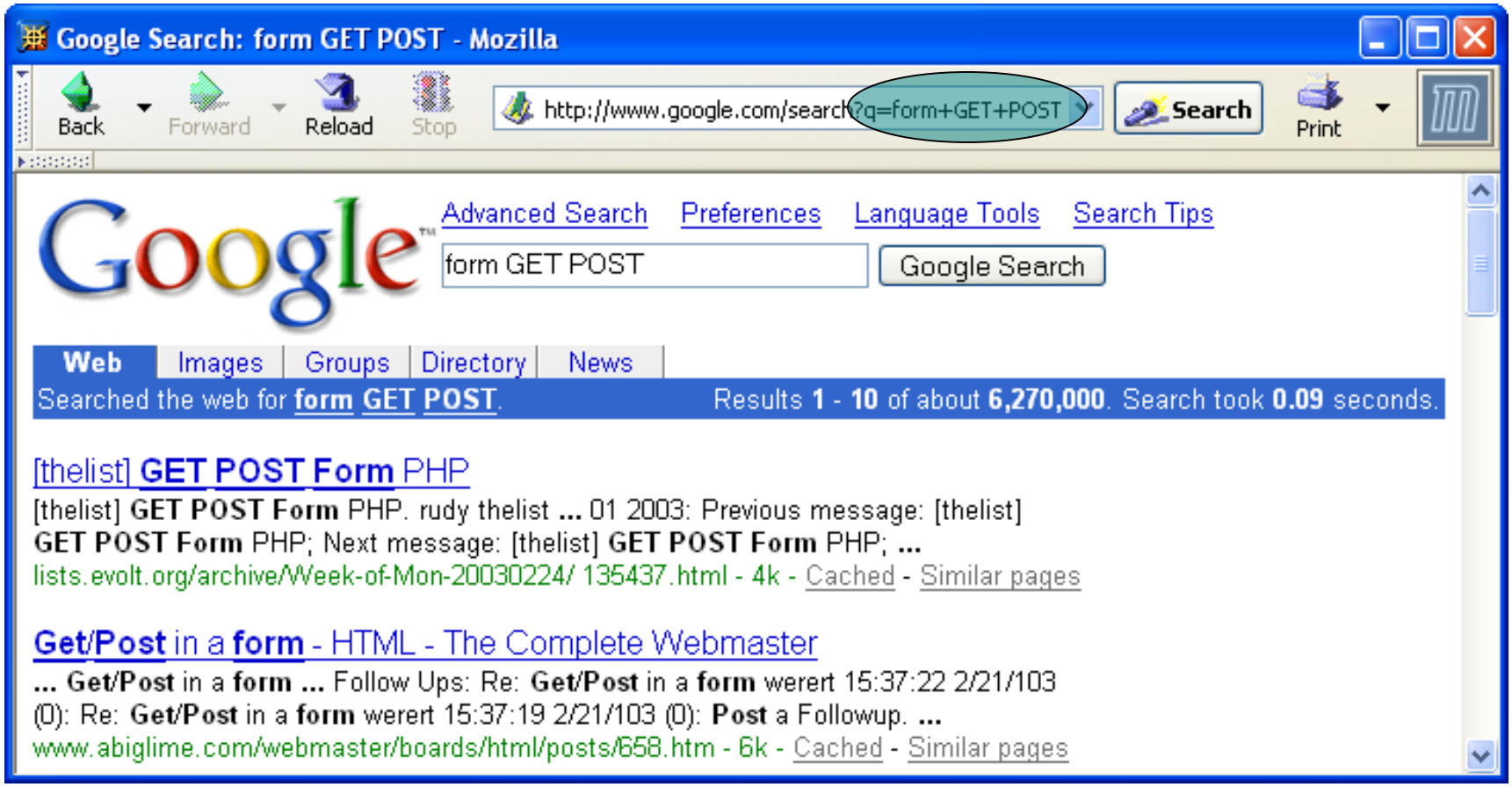
•Parameter boxgroup1 appears twice, because all of the checkboxes have this name.

•The one control that does not appear in the query string is the checkbox labeled “tall,” which was not checked.

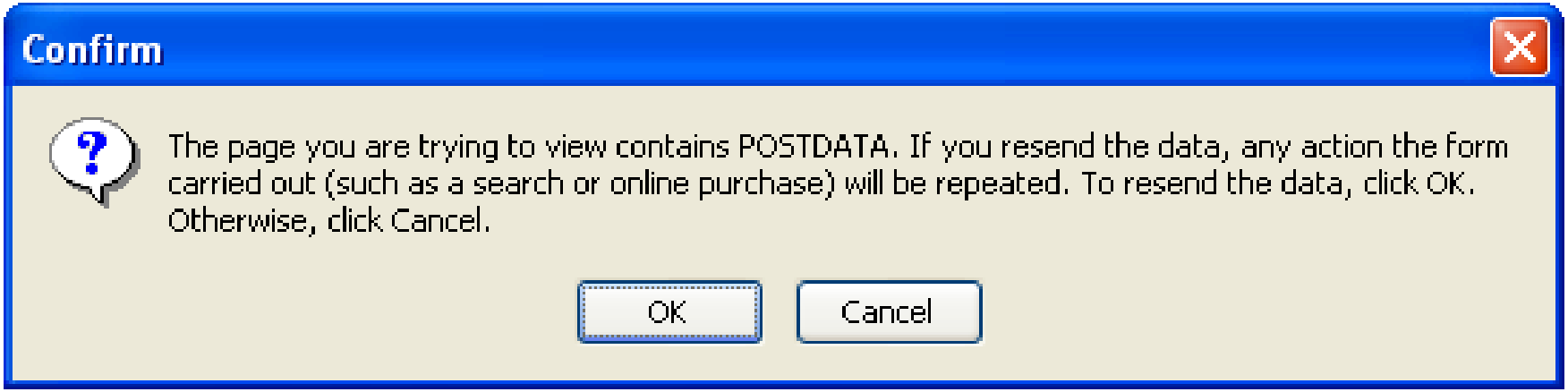
Method attribute of a form element

- GET vs. POST for the method attribute of forms:
  - **GET:**
    - Query string is part of URL
    - Length of query string may be limited
    - Recommended when parameter data is not stored or updated on the server, but used only to request information (*e.g.*, search engine query)
      - The URL can be bookmarked or emailed and the same data will be passed to the server when the URL is revisited

Ex GET Method



- GET vs. POST method for forms:
  - **POST:**
    - Query string is sent as body of HTTP request
    - Length of query string is unlimited
    - Recommended if parameter data is intended to cause the server to update stored data
    - Most browsers will warn you if they are about to resubmit POST data to avoid duplicate updates



Sessions

- Many web sites are designed to obtain information from site visitors over a series of pages rather than in one large page.
- Many interactive Web sites spread user data entry out over several pages:
  - Ex: add items to cart, enter shipping information, enter billing information
- **Problem:** how does the server know which users generated which HTTP requests?
  - Cannot rely on standard HTTP headers to identify a user

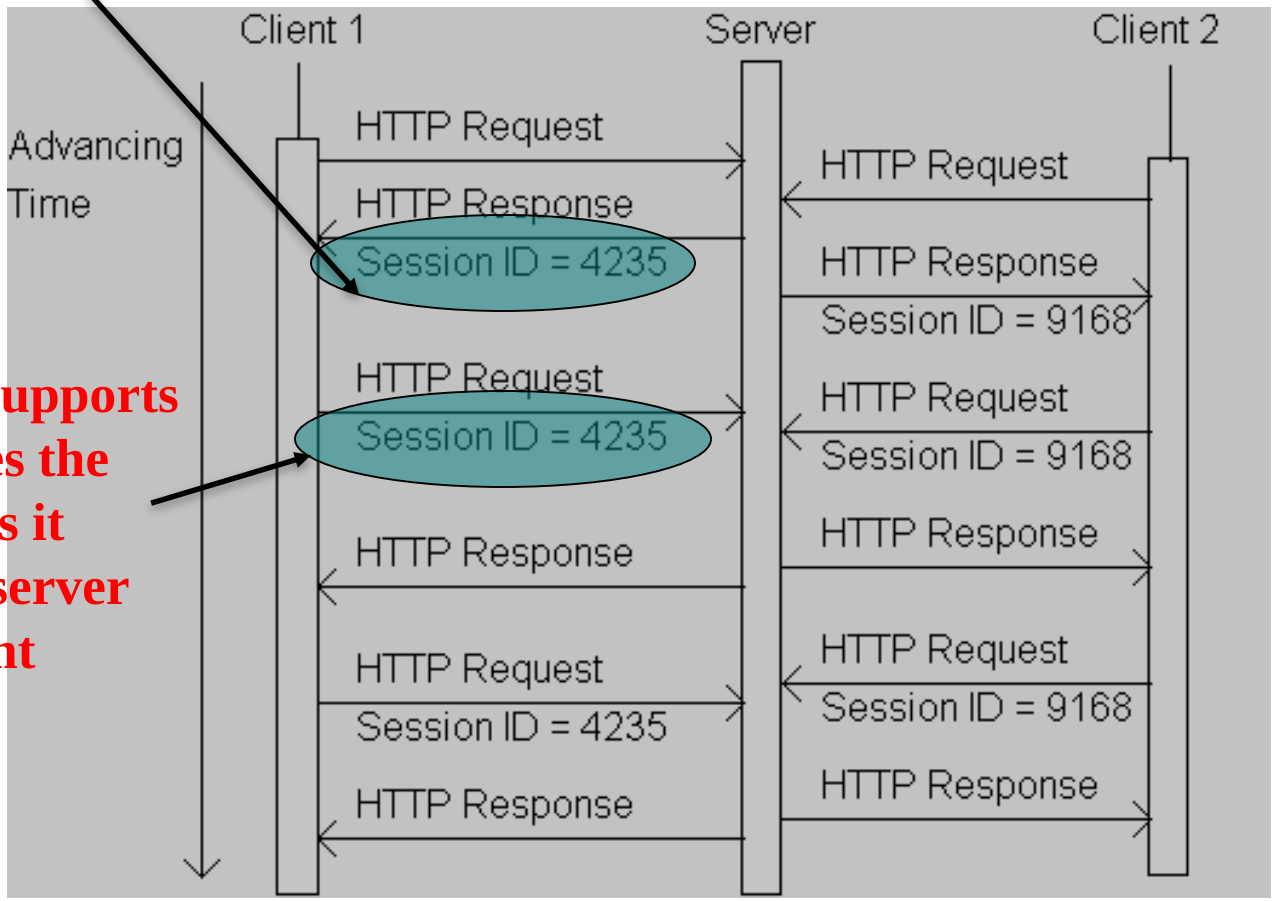
- A separate convention for passing **User-identifying information** between browsers and servers has been developed.
- Specifically, each HTTP request is examined by the server to see if it contains a special identifier known as a **session ID**
- If a request does not contain a session ID, then the request is assumed to be from a new user and the web server generates a new unique session ID that is associated with this user.
- When the HTTP response message is created by the web server, the session ID will be included as part of the response.

- If the browser receiving this response supports the session convention it will store the session ID contained in the response and send it back to the server as part of subsequent HTTP requests.
- when this convention successful, will allow a servlet to recognize all of the HTTP requests coming from a single user.
- Such **a collection of HTTP requests, all associated with a single session ID, is known as a session.**

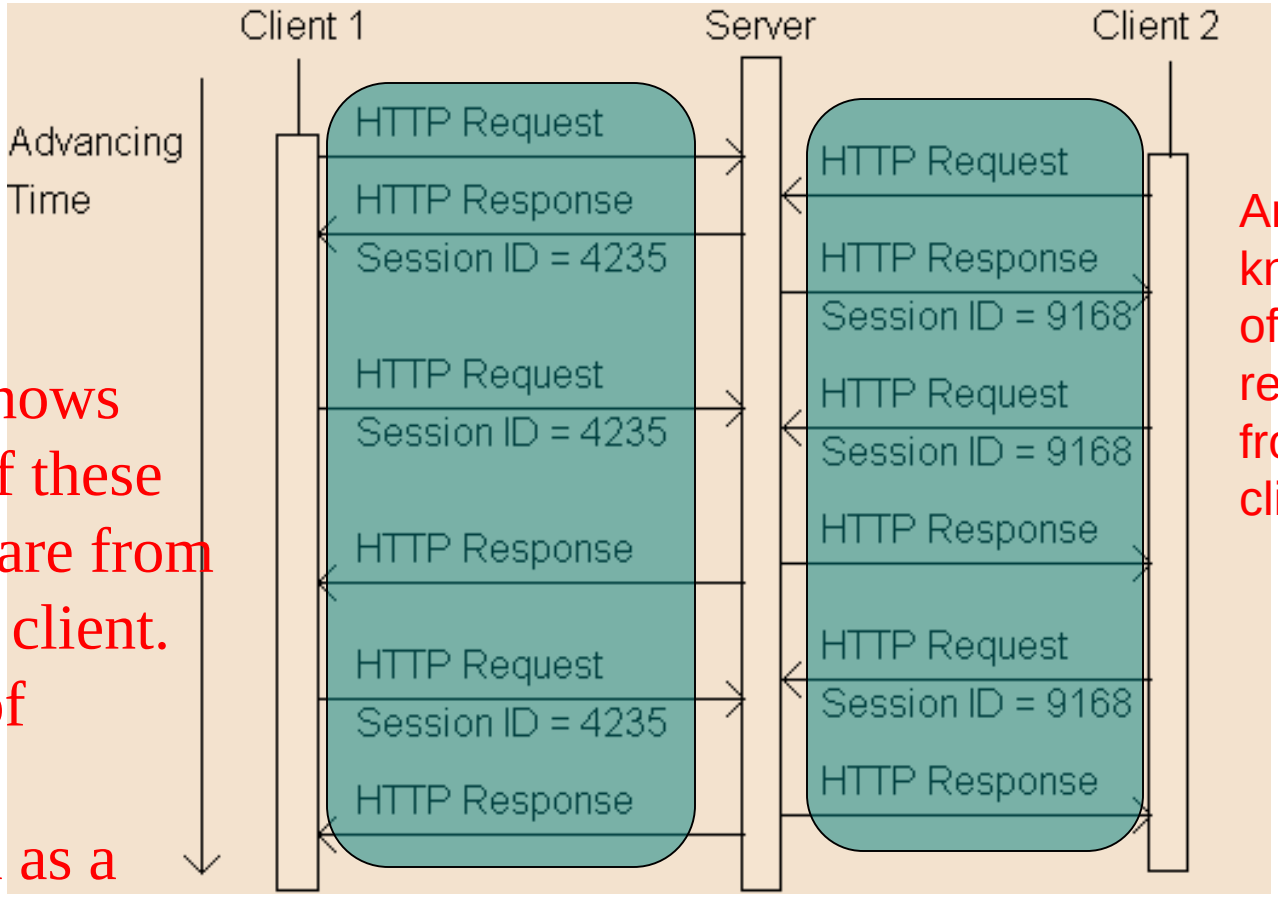
Sessions

Server sends back new unique session ID when the request has none

Client that supports session stores the ID and sends it back to the server in subsequent requests



Sessions



Server knows that all of these requests are from the same client. The set of requests is known as a ***session***.

And the server knows that all of these requests are from a different client.

Creating a Session

•A server complying with the Java servlet API supports the session concept by associating an HttpSession object with each session maintained by the server.

•Each object stores the session ID for its session as well as other session-related information

•An HttpSession object is created by the server when a servlet calls the getSession() method on its HttpServletRequest parameter and the associated HTTP request does not contain a valid session ID

•The getSession() method returns the newly created object in this case.

```
HttpSession session = request.getSession();
if (session.isNew()) {
    visits++;
}
```

Incremented once per session

Boolean indicating whether returned object was newly created or already existed.

HttpSession object is created by server when a servlet calls getSession() method on its HttpServletRequest parameter.

getSession() method returns HttpSession object associated with this HTTP request.

- Creates new HttpSession object if no valid session ID in HTTP request
- Otherwise, returns previously created HttpSession object containing the session ID

Modifies the earlier HelloCounter servlet

Display number of visitors to the page, rather than the number of visits

The difference is that in the original servlet a single user could visit the page multiple times, and each page visit would increment the visit count.

In the new version, this will not happen .

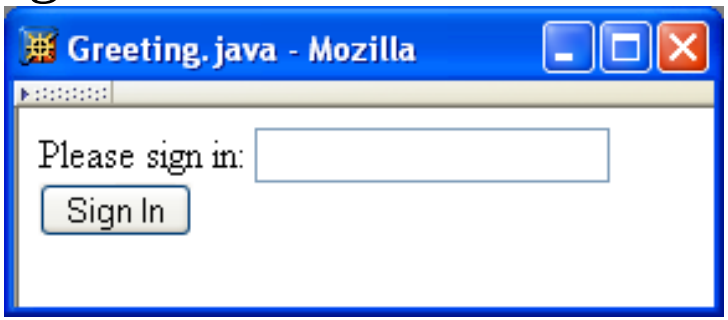
This is because the visit counter is now only incremented on the first visit by a user to the page, which can be detected by checking to see whether or not a new session has begun.

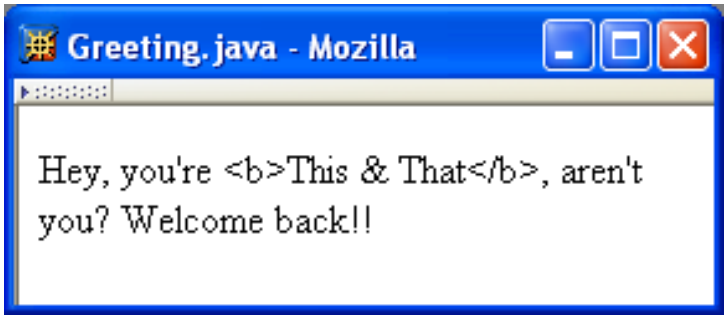
Specifically, if the session is not new, then the user has visited the page before, and the counter is not incremented.

Storing and Retrieving Attributes

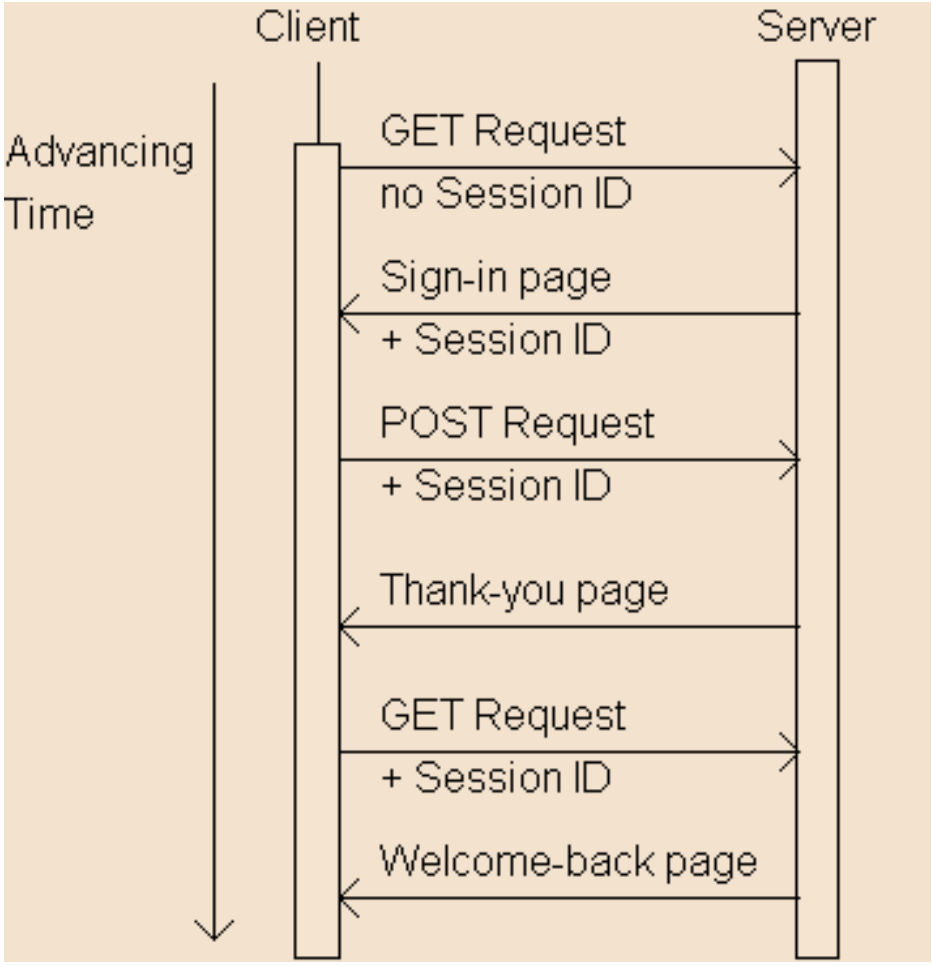
The Java servlet session facility can be used to associate multiple web pages with a single user

Three web pages produced by a single servlet





Client-server interaction producing this sequence of pages



Example Greeting Servlet

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)

  ...
  HttpSession session = request.getSession();
  String signIn = (String)session.getAttribute("signIn");
  if (session.isNew() || (signIn == null)) {
      printSignInForm(servletOut, "Greeting");
  } else {
      printWelcomeBack(servletOut, signIn);
  }
```

Session attribute is a name/value pair

•This servlet is implemented by storing and retrieving an attribute value in the HttpSession object for the user.

•A session attribute is simply a name-value pair that is stored in the HttpSession object.

•Two methods of HttpSession are used to store and retrieve attributes:

•setAttribute(String name, Object obj)

•getAttribute(String name)

Sessions

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)

  ...
  HttpSession session = request.getSession();
  String signIn = (String)session.getAttribute("signIn");
  if (session.isNew() || (signIn == null)) {
      printSignInForm(servletOut, "Greeting");
  } else {
      printWelcomeBack(servletOut, signIn);
  }
```

Session attribute will have null value until a value is assigned

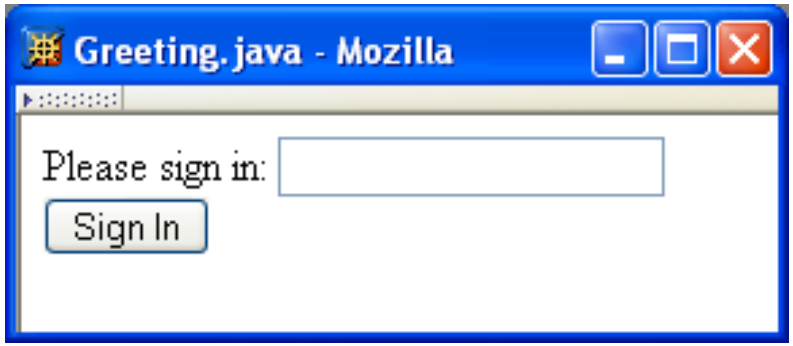
Sessions

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)

  ...
  HttpSession session = request.getSession();
  String signIn = (String)session.getAttribute("signIn");
  if (session.isNew() || (signIn == null)) {
      printSignInForm(servletOut, "Greeting");
  } else {
      printWelcomeBack(servletOut, signIn);
  }
```

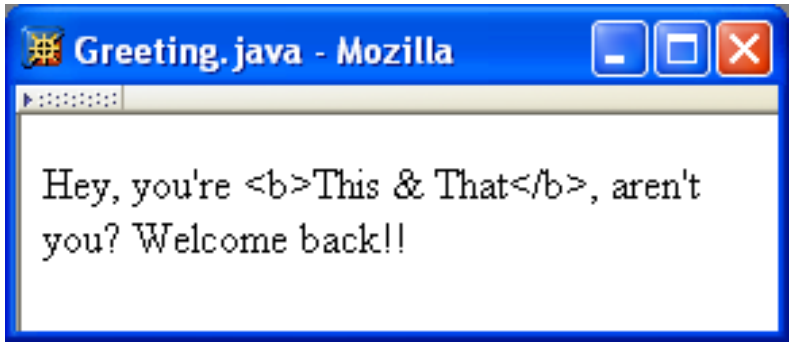
Generate sign-in form if session is new or signIn attribute has no value, generate weclome-back page otherwise.

Sessions

Sign-in form



Welcome-back page



Sessions

```
    private void printSignInForm(PrintWriter servletOut,
                                 String action)
    {
        ...
        servletOut.println(
"     <form method='post' action='" + action + "'><div> \n" +
"       <label> \n" +
"         Please sign in: <input type='text' name='signIn' /> \n" +

    ...
```

Second argument ("Greeting") used as `action` attribute value (relative URL)

Sessions

```
private void printSignInForm(PrintWriter servletOut,
                             String action)
{
    ...
    servletOut.println(
"    <form method='post' action='" + action + "'><div> \n" +
"      <label> \n" +
"        Please sign in: <input type='text' name='signIn' /> \n" +

...
```

Form will be sent using POST HTTP Method, so `doPost()` method will be called

Sessions

```
    private void printSignInForm(PrintWriter servletOut,
                                 String action)
    {
        ...
        servletOut.println(
"     <form method='post' action='" + action + "'><div> \n" +
"       <label> \n" +
"         Please sign in: <input type='text' name='signIn' /> \n" +

    ...
```

Text field containing user name is named `signIn`

Sessions

```
public void doPost (HttpServletRequest request,
                    HttpServletResponse response)
  ...
  String signIn = request.getParameter("signIn");
  HttpSession session = request.getSession();
  if (signIn != null) {
      printThanks(servletOut, signIn, "Greeting");
      session.setAttribute("signIn", signIn);
  } else {
      printSignInForm(servletOut, "Greeting");
  }
```

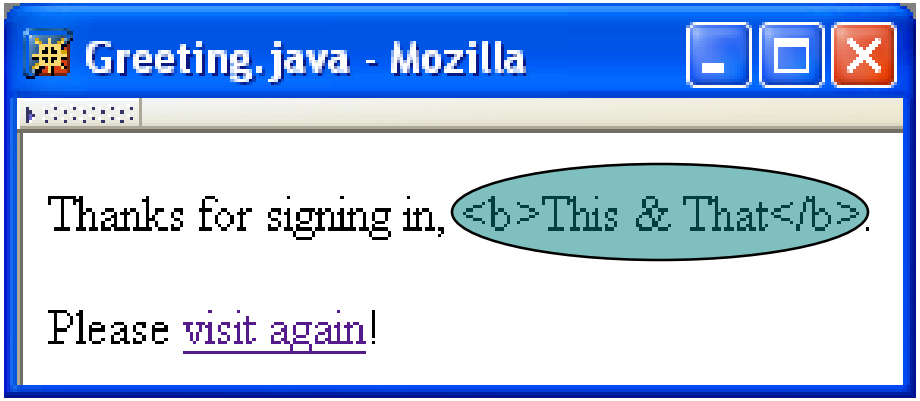
Retrieve `signIn` parameter value

Normal processing: `signIn` parameter is present in HTTP request

Generate HTML for response

Sessions

Thank-you page



Must escape XML special characters in user input

Sessions

```
public void doPost (HttpServletRequest request,
                    HttpServletResponse response)
  ...
  String signIn = request.getParameter("signIn");
  HttpSession session = request.getSession();
  if (signIn != null) {
      printThanks(servletOut, signIn, "Greeting");
      session.setAttribute("signIn", signIn);
  } else {
      printSignInForm(servletOut, "Greeting");
  }
```

Assign a value to the `signIn` session attribute

Session Termination

- By default, each session expires if a server-determined length of time elapses between a session's HTTP requests
  - Server destroys the corresponding session object
- Servlet code can:
  - Terminate a session by calling invalidate() method on session object(to terminate a runnong session)
  - Set the expiration time-out duration (secs) by calling setMaxInactiveInterval(int)

The primary mechanism used to implement the session concept—so-called **cookie processing**

Cookies

- A cookie is a name-value pair that a web server sends to a client machine as part of an HTTP response, specifically through the Set-Cookie header field.
- Browsers will typically store the cookie pairs found in the response in a file on the client machine.
- Then, before sending a request to a web server, the browser will check to see if it has stored any cookies received from this server.
- If so, the browser will include these cookies in the Cookie header field of its HTTP request.

- The cookie mechanism is a natural means of implementing the session concept automatically as part of the processing performed by the getSession() method.
- Specifically, if a server uses cookies to maintain a session, then a call to getSession() will cause the server to look for a cookie named JSESSIONID in the Cookie header field of the request.
- If a JSESSIONID cookie is found, its value is used to search the server's list of valid session objects for an object with the same session ID.

- If found, a reference to this object is returned as the value of the getSession() call.
- Otherwise, if no JSESSIONID cookie is found or if the cookie value does not match the session ID of any valid session object, a new session object is created.
- A JSESSIONID cookie having the session ID of this new object as its value is then created, and this cookie is added to the Set-Cookie header field of the HTTP response.

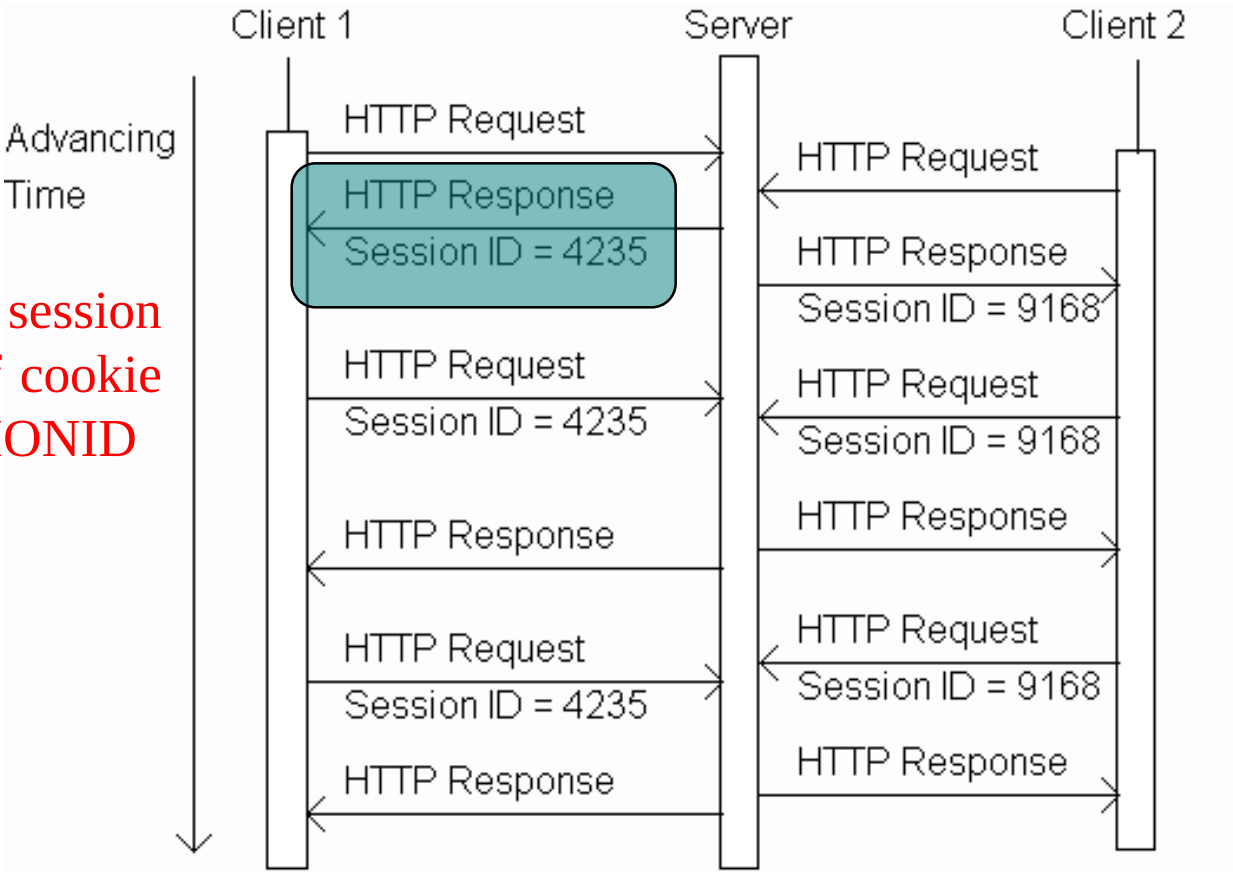
- The new session object is then returned to the servlet.
- Servlets can also explicitly create and use cookies.
- The Java servlet API provides a class called Cookie
- Each instance of this class corresponds to a single cookie.
- This class can be used to create internal representations of new cookies and to access the name-value data in existing Cookie objects

Method	Purpose
`Cookie(String name, String value)`	Constructor to create a cookie with given name and value.
`String getName()`	Return name of this cookie.
`String getValue()`	Return value of this cookie.
`void setMaxAge(int seconds)`	Set delay until cookie expires. Positive value is delay in seconds, negative value means that the cookie expires when the browser closes, and 0 means delete the cookie.

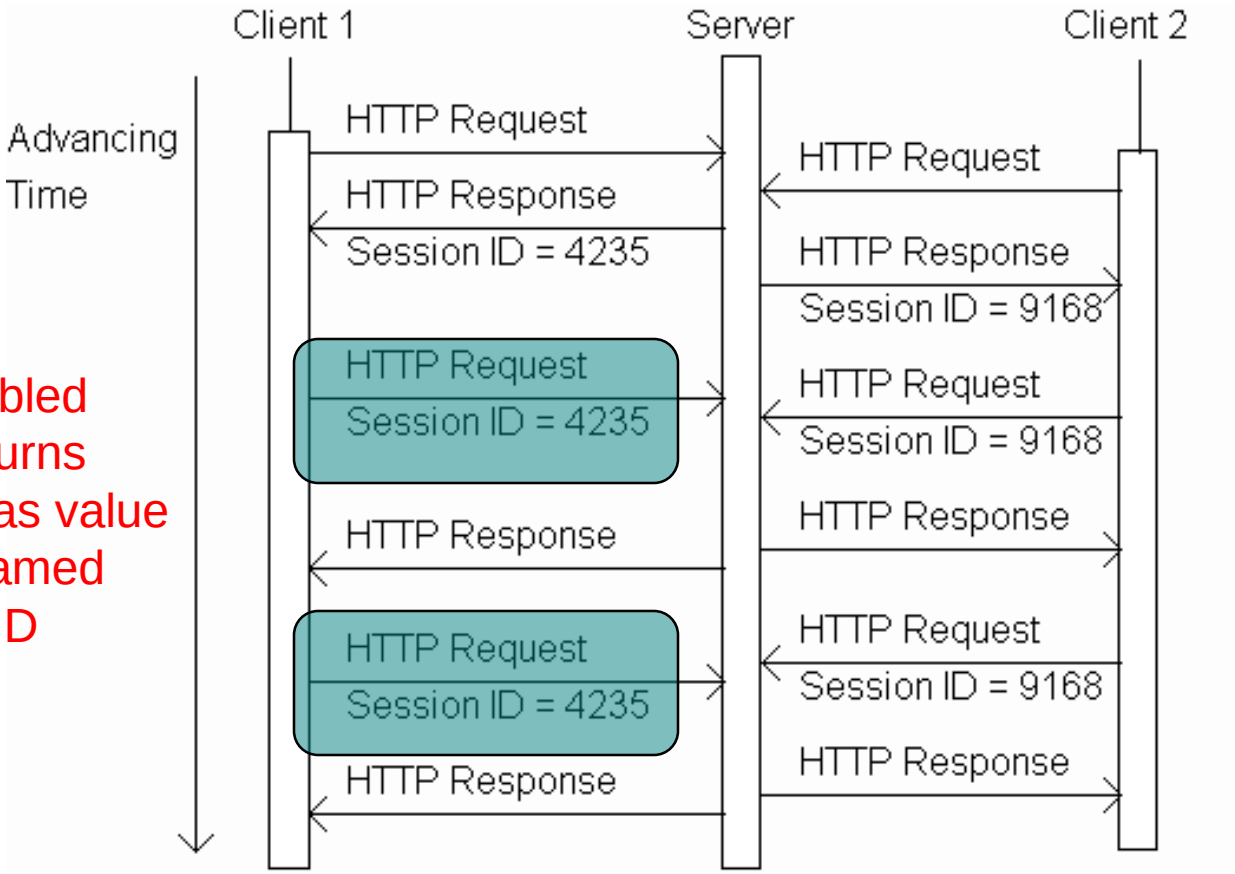
- Two other methods are used to transfer the information between this internal representation and the representation of a cookie in an HTTP header:
- getCookies() method on the HttpServletRequest parameter returns an array of Cookie objects corresponding to the cookies sent by a browser in the HTTP request.
- addCookie(Cookie cookie) method on the HttpServletResponse parameter tells the server to add the information in the given cookie to the Set-Cookie header field when the server later sends its HTTP response to the client.
- Cookies, like sessions, can expire, but the expiration is performed by the client, not the server. (server can request
- expiration date)
- A browser will not send an expired cookie in subsequent HTTP requests.

Cookies

Tomcat sends session ID as value of cookie named JSESSIONID



Cookies



Cookie-enabled browser returns session ID as value of cookie named JSESSIONID

Visit counter theme

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException
{
    // Get count from cookie if available, otherwise
    // use initial value.
    int count = 0;
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (int i=0; (i<cookies.length) && (count==0); i++) {
            if (cookies[i].getName().equals("COUNT")) {
                count = Integer.parseInt(cookies[i].getValue());
            }
        }
    }
```

Return array of cookies contained in HTTP request

Search for cookie named COUNT and extract value as an int

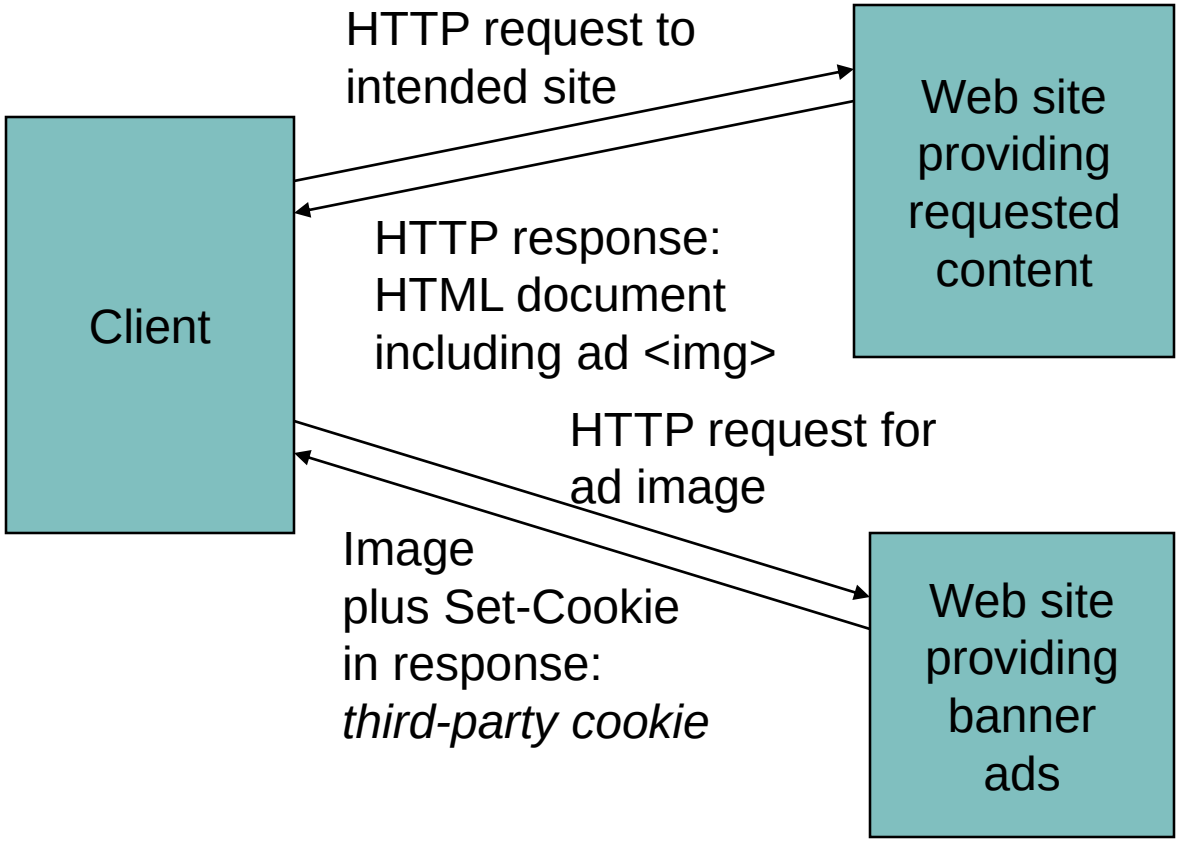
Cookies

Send replacement cookie value to client (overwrites existing cookie)

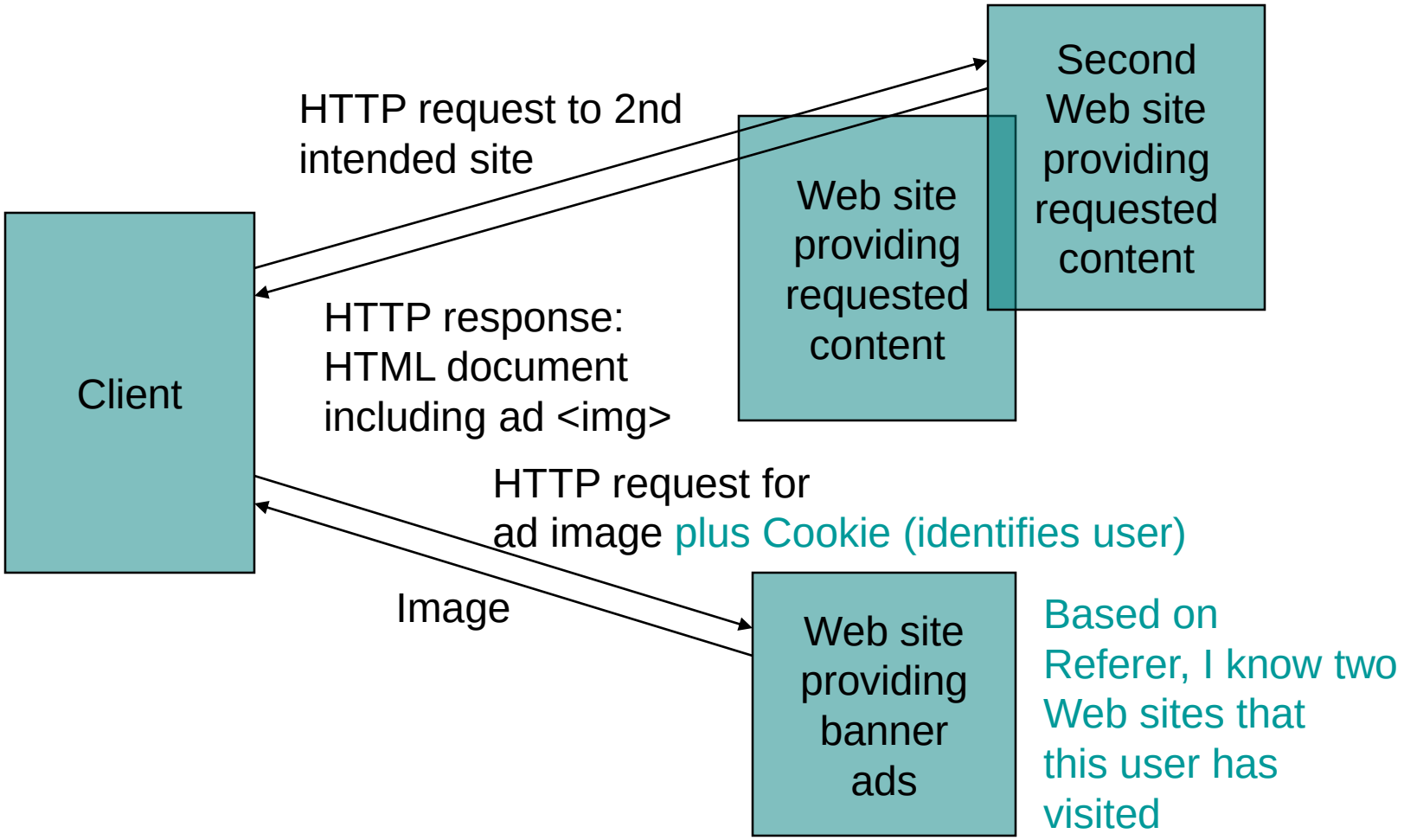
```
// Increment the count and add request to client to store it
// for one year.
count++;
Cookie cookie = new Cookie("COUNT",
                           new Integer(count).toString());
cookie.setMaxAge(oneYear);
response.addCookie(cookie);

// Set the HTTP content type in response header
response.setContentType("text/html; charset=\"UTF-8\"");
. . .
"  <body> \n" +
"    <p>You have visited this page " + count + " time(s) \n" +
"       in the past year, or since clearing your cookies.</p> \n" +
"  </body> \n" +
```

Should call `addCookie()` before writing HTML

Cookies
Privacy issues



Cookies
Privacy issues



Cookies	Sessions
Cookies are client-side files that contain user information.	Sessions are server-side files that contain user information
cookies are stored in the user's browser	sessions are not
A cookie can keep information in the user's browser until deleted.	Sessions is that when you close your browser you also lose the session.
If you set the variable to "cookies", then your users will not have to log in each time they enter your community.	If you set the variable to "sessions", then user activity will be tracked using browser sessions, and your users will have to log in each time they re-open their browser.
Cookies can only store strings.	Store our objects in sessions.
save cookie for future reference	Session couldn’t. Users close their browser, they also lost the session.

Cookies
Privacy issues

- Users can remove their cookies.
- In Mozilla 1.4, for example, cookie removal can be performed by selecting **Tools| Cookie Manager| Manage Stored Cookies.**
- A user can also choose to block (refuse to accept) cookies from particular web sites, or to block cookies entirely.
- Alternative to cookies for maintaining session: URL rewriting

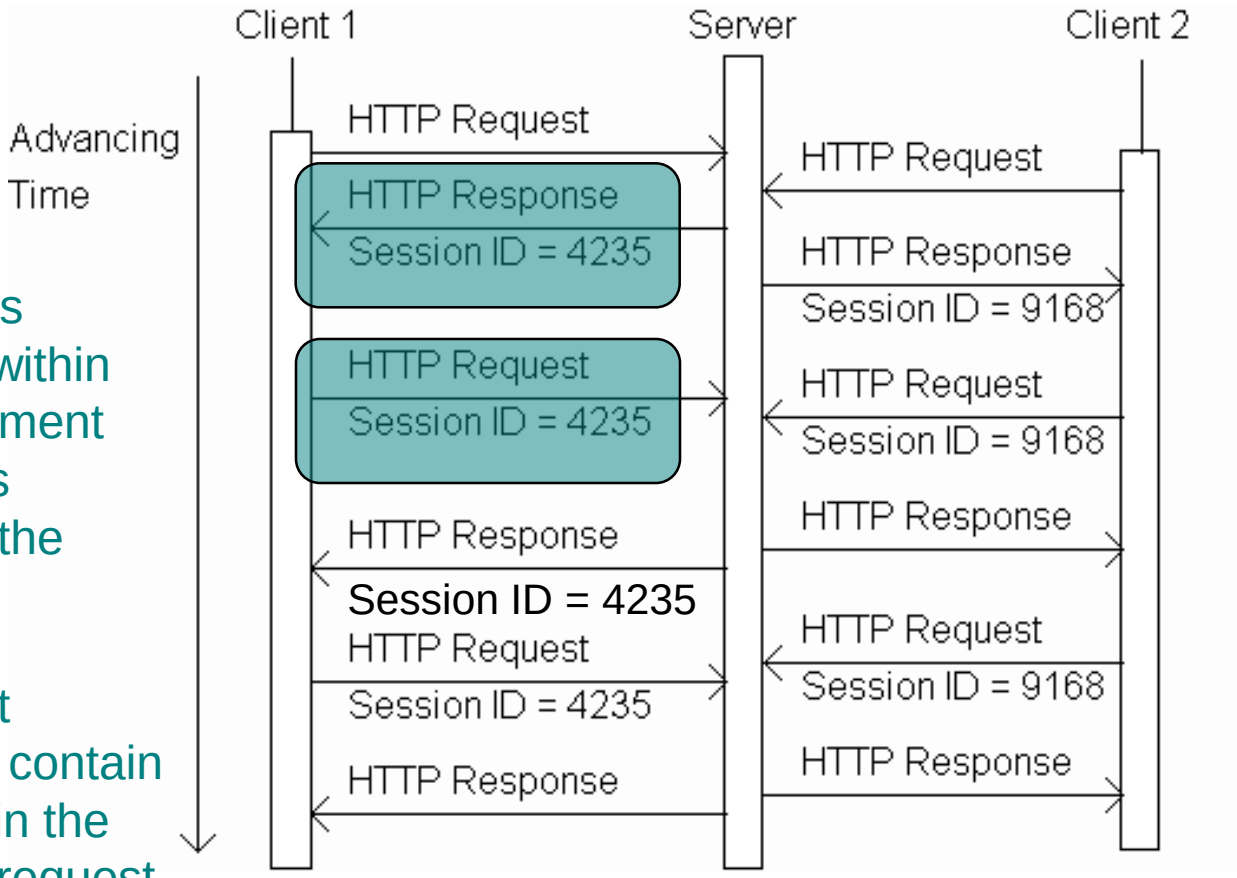
URL Rewriting

- Passing a session ID between server and client through HTTP headers is to pass it via the HTML documents themselves.
- the server to write the session ID within every HTML document it returns to the client
- Involves rewriting every URL referencing the servlet in the href attribute of any anchor and the action attribute of any form output by the servlet.
- Whenever the server receives an HTTP request, it must check the URL it receives for session ID information and, if found, use the session ID just as it would if it had been passed to the server via a cookie.

- HttpServletResponse interface supports this approach to maintaining session by defining an encodeURL(String url) method.
- Given a url argument, this method returns the same URL plus, if appropriate, a session ID
- The session ID is added via a little-used URL feature known as a *path parameter.*
- path parameter is added to a URL by appending a semicolon to the URL followed by a name-value pair
- The server checks for the presence of session information within the request URL when getSession() is called

- If a JSESSIONID cookie is not found, the server will check for a jsessionid path parameter in the request URL.
- If this is found, the server records that this session must be maintained using URL rewriting.
- It then continues with its session processing, using the session ID contained in the path parameter just as it would if the ID had come from a cookie.
- boolean HttpServletRequest methods isRequestedSessionIdFromCookie() and isRequestedSessionIdFromURL() can be called by the servlet code to determine how the session ID was transmitted in the current HTTP request.

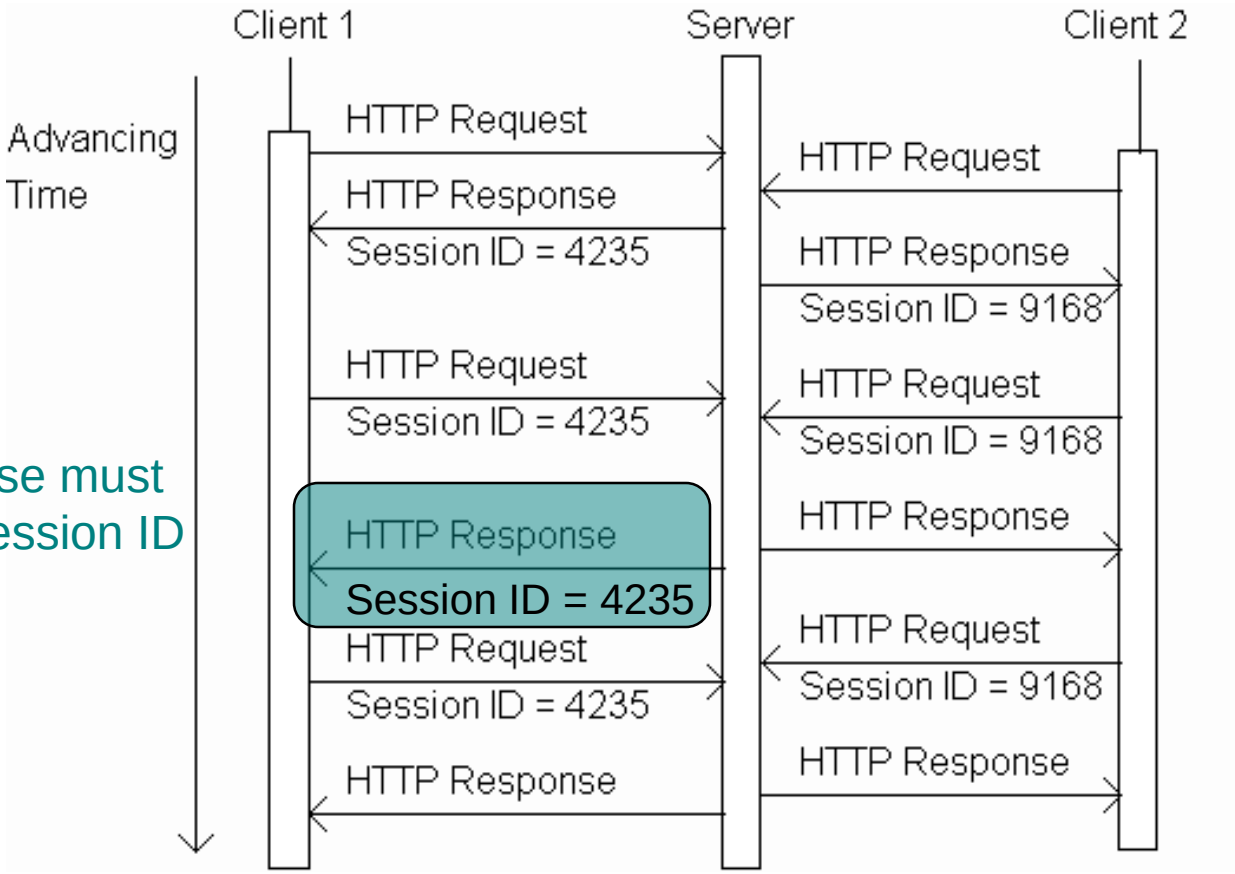
URL Rewriting



Tomcat adds session ID within HTML document to all URL's referring to the servlet

Subsequent request will contain session ID in the URL of the request

URL Rewriting



Next response must again add session ID to all URL's

URL Rewriting

- Original (relative) URL:

  `href="URLEncodedGreeting"`

- URL containing session ID:

  `href="URLEncodedGreeting;jsessionid=0157B9E85"`

  Path parameter

- Path parameter is treated differently than query string parameter
  - Ex: invisible to `getParameter()`

URL Rewriting

- `HttpServletResponse` method `encodeURL()` will add session id path parameter to argument URL

Original servlet

```
printSignInForm(servletOut, "Greeting");
```

Relative URL of servlet

Servlet using URL rewriting

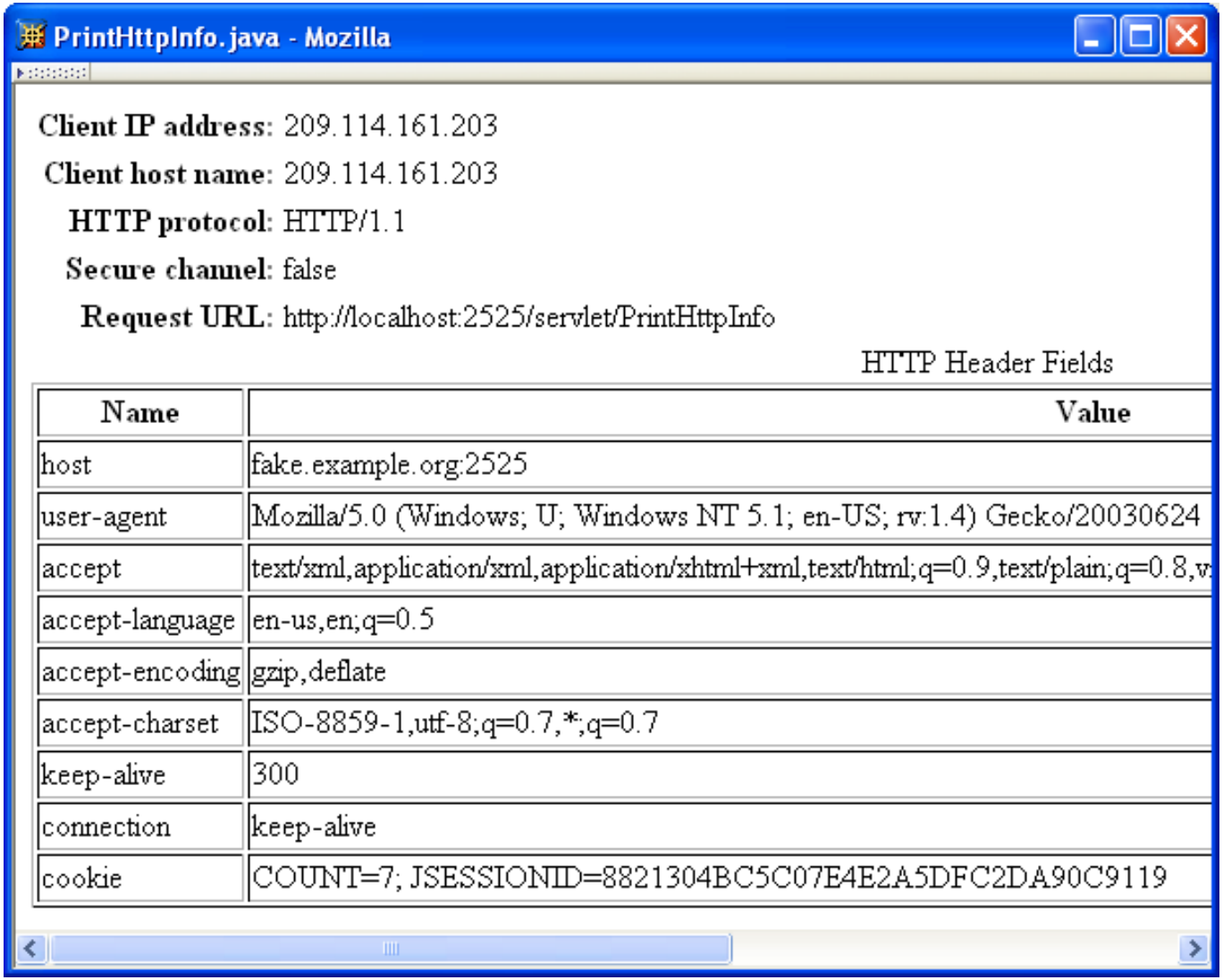
```
printSignInForm(servletOut,
                response.encodeURL("URLEncodedGreeting"));
```

Other Servlet Capabilities
HttpServletRequest methods

TABLE 6.4: Additional `HttpServletRequest` methods.

Method	Purpose
`String getRemoteAddr()`	Return IP address of the client machine making this request.
`String getRemoteHost()`	Return fully qualified name of the client making this request, or its IP address if name is not available.
`String getProtocol()`	Return the type and version of communication protocol used by the client to make this request (e.g., "HTTP/1.1").
`boolean isSecure()`	Return boolean indicating whether or not this request was made over a secure communication channel.
`StringBuffer getRequestURL()`	Return a StringBuffer containing the URL used to access this servlet, excluding any query string appended to the URL as well as any jsessionid path parameter.
`Enumeration getHeaderNames()`	Return an Enumeration of String objects representing names of all header fields in the request.
`String getHeader(String fieldName)`	Given a valid header field name, return a String representing the value of the field, or `null` if header field is not present in request. The match of `fieldName` against header field names is case-insensitive.

More Servlet Methods



HttpServletResponse methods

TABLE 6.5: Additional `HttpServletResponse` methods.

Method	Purpose
`void setHeader(String name, String value)`	Include a header field with the given `name` and `value` in the HTTP response.
`void setDateHeader(String name, long value)`	Include a date header field (such as Expires) with the given `name` in the HTTP response. The given `value` is converted from milliseconds since 00:00 01 January 1970 UTC to an equivalent time in HTTP date format.
`void setContentLength(int len)`	Set the Content-Length header field to the given value.
`void setBufferSize(int size)`	Set the desired size of the response buffer (see below). The server may override the specified `size` and use a larger value. This method must be called before any data is written into the response buffer
`int getBufferSize()`	Return an integer representing the actual size of the response buffer.

More Servlet Methods

- Response buffer
  - All data sent to the `PrintWriter` object is stored in a buffer
  - When the buffer is full, it is automatically flushed:
    - Contents are sent to the client (preceded by header fields, if this is the first flush)
    - Buffer becomes empty
  - Note that all header fields must be defined before the first buffer flush

More Servlet Methods

TABLE 6.5: Additional `HttpServletResponse` methods.

`void setStatus(int statusCode)`	Set the status code in the HTTP response (status code is 200 (OK) by default). Any information contained in the response buffer is cleared. Use only for non-error status codes.
`void sendError(int statusCode, String msg)`	Set the status code in the HTTP response to the given error `statusCode` (status code beginning with 4 or 5), and in the body of the response send a server-generated HTML error page containing the given `msg`.
`void sendRedirect(String url)`	Cause HTTP response with status code 307 (Temporary Redirect) to be sent to the client, causing the client to send a new HTTP request to the given `url`. Client will behave as if it had sent request to the specified `url`.
`void encodeRedirectURL(String url)`	Perform URL rewriting (for session management) on `url` that will be used for redirection.

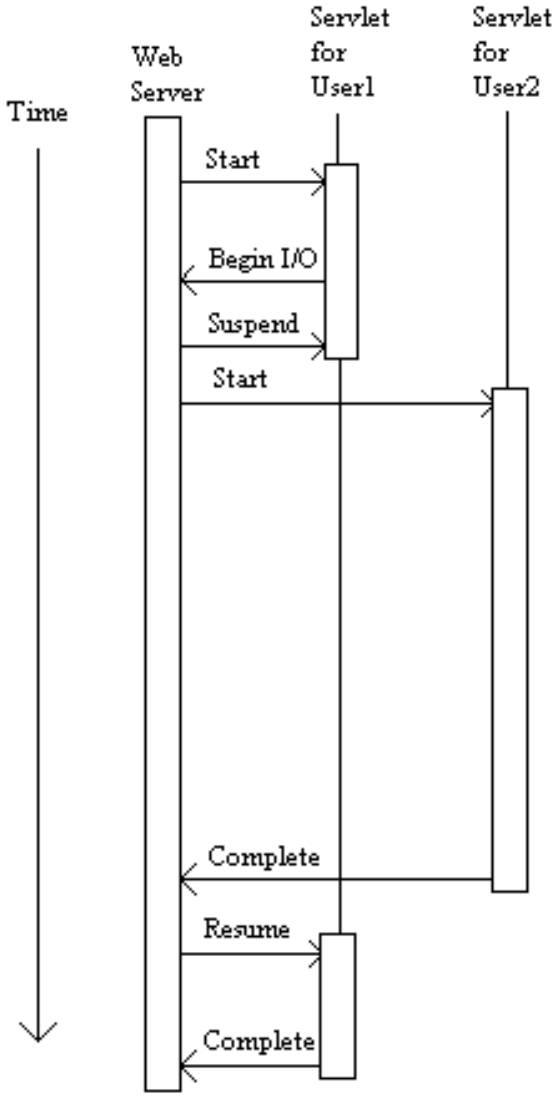
Other Http methods

- In addition to `doGet()` and `doPost()`, servlets have methods corresponding to other HTTP request methods
  - `doHead()`: automatically defined if `doGet()` is overridden
  - `doOptions()`, `doTrace()`: useful default methods provided
  - `doDelete()`, `doPut()`: override to support these methods

Data Storage

- Almost all web applications (servlets or related dynamic web server software) store and retrieve data
  - Typical web app uses a data base management system (DBMS)
  - Another option is to use the file system
  - Not web technologies, so beyond our scope
- Some Java data storage details provided in Appendices B (file system) and C (DBMS)
- One common problem: concurrency

Servlets and Concurrency



Concurrency

- Tomcat creates a separate thread for each HTTP request
- Java thread state saved:
  - Which statement to be executed next
  - The call stack: where the current method will return to, where that method will return to, *etc*. plus parameter values for each method
  - The values of local variables for all methods on the call stack

Concurrency

- Some examples of values that are *not* saved when a thread is suspended:
  - Values of instance variables (variables declared outside of methods)
  - Values of class variables (variables declared as static outside of methods)
  - Contents of files and other external resources

Concurrency

```
public class HelloCounter extends HttpServlet
{
    // Number of times the servlet has been executed since
    // the program (web server) started
    private int visits=0;

[...] // removed doGet() declaration and initialization

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();

          // Compute the number of visits to the URL for this servlet
          visits++;
```

// Output HTML document

Concurrency

User1 Thread	User2 Thread
<started>	
.	
.	
.	
visits++;	
<visits now 18>	
<suspended>	
	<started>
	.
	.
	.
	visits++;
	<visits now 19>
	servletOut.println(...
	visits + ...);
	<outputs 19>
	<completed>
<resumed>	
servletOut.println(...	
visits + ...);	
<outputs 19>	

Concurrency

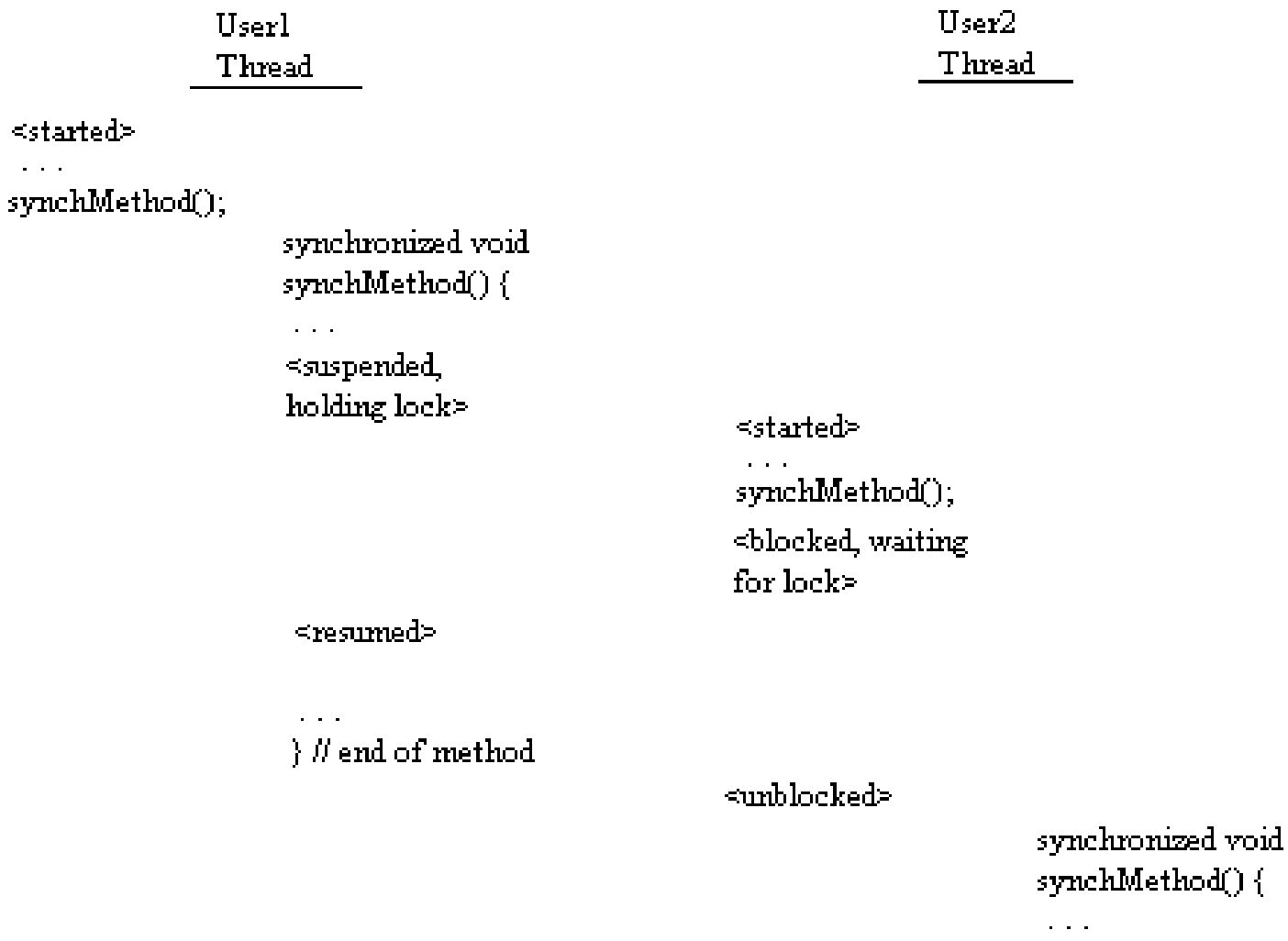
- Java support thread synchronization

```
synchronized public void doGet (HttpServletRequest request,
                                HttpServletResponse response)
```

Only one thread at
at time can call `doGet()`

  - Only one synchronized method within a class can be called at any one time

Concurrency



Concurrency

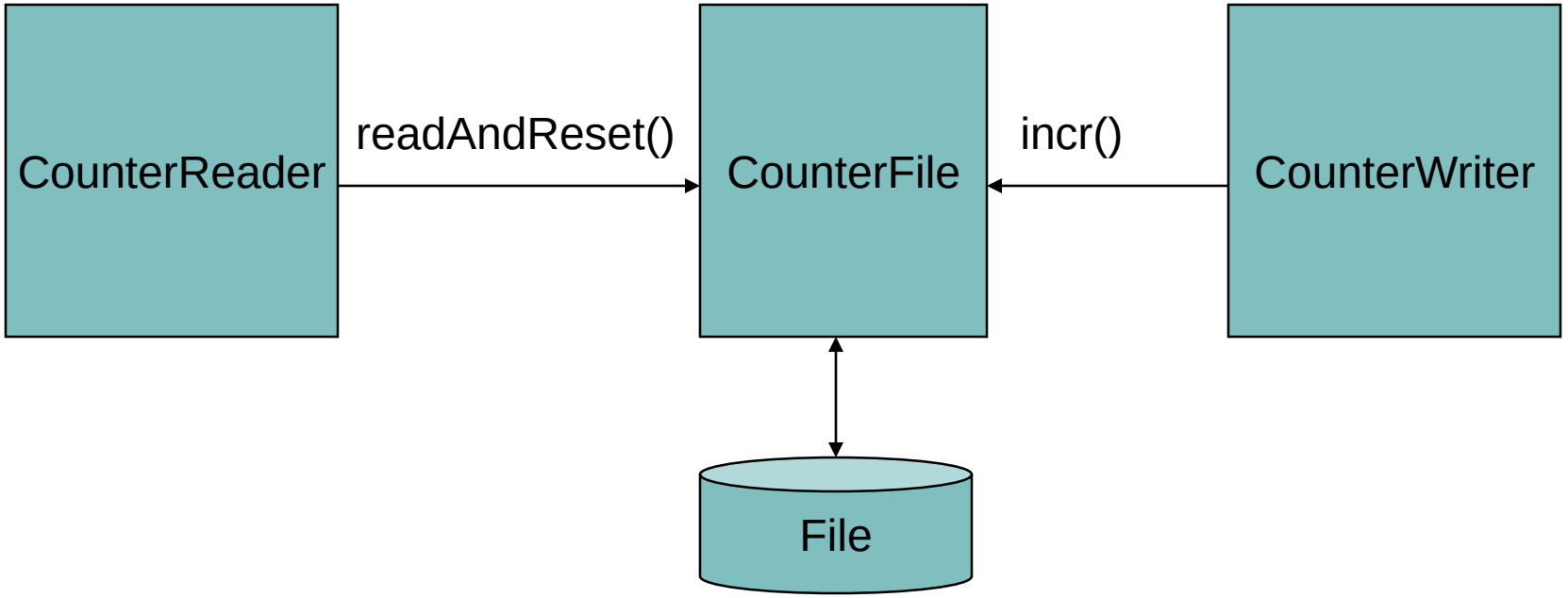
- Web application with multiple servlet classes and shared resource:

```
Servlet 1 (CounterReader)
-------------------------
Input count from file (24)
<suspended>



<resumed>
Overwrite file with 0
Display count (24)
```

```
Servlet 2 (CounterWriter)
-------------------------


Input count from file (24)
Increment count (25)
Overwrite file with count (25)
Display "Hello World!" page
```

Concurrency

- Solution: create a shared class with synchronized static methods called by both servlets



Common Gateway Interface

- CGI was the earliest standard technology used for dynamic server-side content
- CGI basics:
  - HTTP request information is stored in environment variables (*e.g.*, QUERY_STRING, REQUEST_METHOD, HTTP_USER_AGENT)
  - Program is executed, output is returned in HTTP response

Common Gateway Interface

- Advantage:
  - Program can be written in any programming language (Perl frequently used)
- Disadvantages:
  - No standard for concepts such as session
  - May be slower (programs normally run in separate processes, not server process)