

Recommended reading:

Core SERVLETS and JAVASERVER by Marty Hall, Sun Microsystems Press/Prentice Hall PTR Book, ISBN 0-13-089340-4

These handouts are adapted from Hall's notes.

COMPSCI334

1

A Servlet's Job

- A servlet can be regarded as a Java program residing on a web server.
- Servlets are used to generate web pages dynamically.
- 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)

COMPSCI334

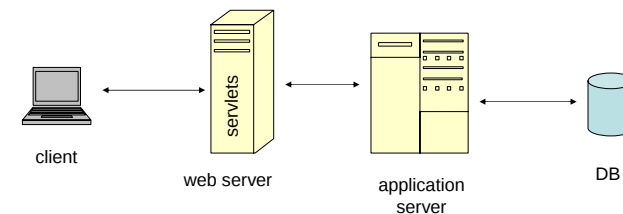
2

Response from the web server

- The response from a web server consists of
 - a status line
 - HTTP/1.1 200 OK
 - Some response headers
 - Content-Type: text/html
 - Date: Sun, 17 Feb 2008 23:11:38 GMT
 - The document.
 - <HTML><BODY>Hello World </BODY></HTML>

COMPSCI334

3



COMPSCI334

4

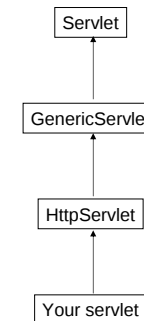
Why Build Web Pages Dynamically?

- The Web page is based on data submitted by the user
 - results page from search engines and DB queries
- The Web page is derived from data that changes frequently
 - a weather report or stock price

COMPSCI334

5

Servlet Classes



COMPSCI334

6

Commonly used servlet methods

- Class HttpServlet includes some methods for handling the corresponding HTTP request methods.
 - doGet(), doPost(), doHead() etc.
- In your servlets, you need to overwrite these methods to make the servlets behave according to your requirements.

COMPSCI334

7

First Servlet

```
package examples;  
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!");  
    }  
}
```

obtain output object

send string to client

COMPSCI334

8

Creating, Deploying and Executing Servlet/JSP in Tomcat 6

- Tomcat 6 is the official reference implementation of the Servlet 2.5 and JavaServer Pages 2.1 specifications.
- It can be used as a small stand-alone server for testing servlets and JSP pages.
- Tomcat is free from <http://tomcat.apache.org/>
- There are many versions of Tomcat. The version that we use in 334 is Tomcat 6.

COMPSCI334

9

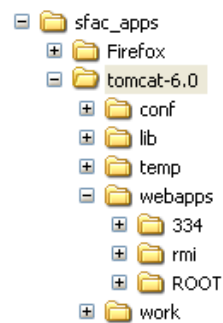
Setup Environment on Windows

- After installing Tomcat, you need to set up some environment variables.
- JAVA_HOME should be set to the directory where J2SE is installed.
 - e.g. set JAVA_HOME = d:\jdk
 - In Windows XP, use “control panel” → “system”. Then, click “Advanced” tab → “Environment Variables” button. Then, use “New” or “Edit” button to set the variable.
- CLASSPATH should also be extended to include the jar file which contains the servlet and jsp package.
 - e.g. set CLASSPATH = .;C:\apache-tomcat-6.0\common\lib\servlet-api.jar;C:\apache-tomcat-6.0\common\lib\jsp-api.jar;%CLASSPATH%
 - or, use “control panel” → “system” in Windows XP as above.

COMPSCI334

10

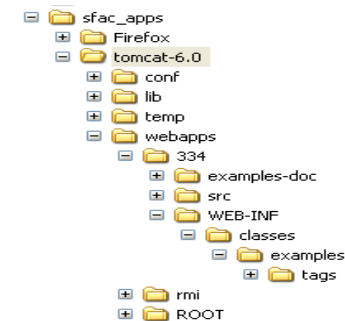
- download rmi-class.zip file from <http://www.cs.auckland.ac.nz/compsci334s1t/resources/rmi-classes.zip>
- unpack the file and store it at H:\sfac_apps\tomcat-6.0\webapps



COMPSCI334

11

- You can download the servlet and jsp examples from 334 web site and unpack them under webapps directory of the Tomcat directory.
- After unpacking, in the lab your directory under webapps should be as below



12

- Type <http://localhost:8080/334/>, the page below is shown.
- Clicking the program description link, e.g. [Hello World](#), will activate the corresponding program and show the outputs of the program.
- Clicking the [source](#) link next to a program description link will display the source code of the program.
- The sources code of the programs can be found in folder 'webapps/334/src/'
- The class files of the program are stored in folder 'webapps/334/WEB-INF/classes/examples/'
- The html files are in folder 'webapps/334/examples-doc'



COMPSCI334

13

- You can create your own directory under webapps to store your own servlet and jsp.
- Your directory must contain a WEB-INF directory. Under WEB-INF, you must have a web.xml file.
 - copy the web.xml file in 334/WEB-INF
- Under WEB-INF, you must have a “classes” directory where all your servlets are stored.

COMPSCI334

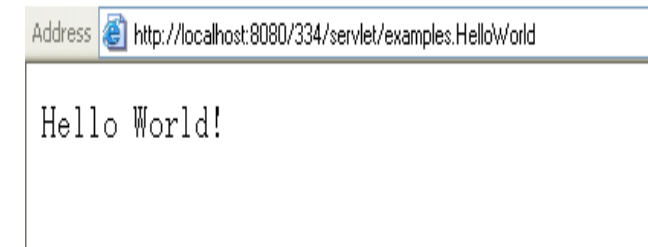
14

Compiling and Invoking HelloWorld

- Place source code HelloWorld.java in directory examples.
- Compile HelloWorld.java
 - javac HelloWorld.java
- Start Tomcat
 - Tomcat listens to http request on port 8080
- Send request
 - <http://localhost:8080/334/servlet/examples.HelloWorld>

COMPSCI334

15



COMPSCI334

16

add the following to web.xml

```
<servlet>
  <servlet-name>HelloWorld</servlet-name>
  <servlet-class>examples.HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloWorld</servlet-name>
  <url-pattern>/servlet/examples.HelloWorld</url-pattern>
</servlet-mapping>
```

COMPSCI334

17

Packaging Servlets

- Move the files to a subdirectory that matches the intended package name
 - For example, HelloWorld.java under examples
 - “examples” is a subdirectory under “classes”
- Insert a package statement in the class file
 - package examples;
- Include package name in URL
 - http://localhost:8080/334/servlet/examples.HelloWorld

COMPSCI334

18

Handling the Client Request

- Chapter 19, 4, 5 and 6 of *Core SERVLETS and JAVASERVER*
- A form for user to fill in three parameters
- When the “submit query” button is clicked, the form is sent back to the server.
- Servlet “DisplayParams” is used to process the form.
 - retrieve the values entered by the user
 - generate a page which shows the three values entered by the user

COMPSCI334

19

Address http://localhost:8080/334/examples-doc/DisplayParams.html

Collecting Three Parameters

First Parameter:

Second Parameter:

Third Parameter:

COMPSCI334

20

```

<HTML>
<HEAD>
<TITLE>Collecting Three Parameters</TITLE>
</HEAD>
<H1 ALIGN="CENTER">Collecting Three Parameters</H1>

<FORM ACTION="/334/servlet/examples.DisplayParams" METHOD="GET">
First Parameter: <INPUT TYPE="TEXT" NAME="First"><BR>
Second Parameter: <INPUT TYPE="TEXT" NAME="Second"><BR>
Third Parameter: <INPUT TYPE="TEXT" NAME="Third"><BR>
<CENTER>
<INPUT TYPE="SUBMIT">
</CENTER>
</FORM>

</BODY>
</HTML>

```

COMPSCI334

21

Address <http://localhost:8080/334/servlet/examples.DisplayParams?First=1&Second=2&Third=3>

Three Parameters

- First: 1
- Second: 2
- Third: 3

COMPSCI334

22

Response in HTML

```

<HTML>
<TITLE>Display Three Parameters</TITLE>
<BODY>
<H1 ALIGN=CENTER> Three Parameters </H1>
<UL>
<LI><B>First</B>: 1
<LI><B>Second</B>: 2
<LI><B>Third</B>: 3
</UL>
</BODY></HTML>

```

COMPSCI334

23

Reading Form Data In Servlets

- `request.getParameter("name")`
 - Returns the value of first occurrence of name in query string
 - Returns null if no such parameter is in query data
- `request.getParameterValues("name")`
 - Returns an array of the values of all occurrences of name in query string
 - Returns null if no such parameter is in query
- `request.getParameterNames()`
 - Returns Enumeration of request params

COMPSCI334

24

```
package examples;

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

```
public class DisplayParams extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
```

output is a HTML file

COMPSCI334

25

```
out.println("<HTML>" +
    "<TITLE>Display Three Parameters</TITLE>\n" +
    "<BODY>" +
    "<H1 ALIGN=CENTER> Three Parameters </H1>\n" +
    "<UL>\n" +
    "  <LI><B> First </B>: "
    + request.getParameter("First") + "\n" +
    "  <LI><B> Second </B>: "
    + request.getParameter("Second") + "\n" +
    "  <LI><B> Third </B>: "
    + request.getParameter("Third") + "\n" +
    "</UL>\n" +
    "</BODY></HTML>");
```

retrieve the values of the parameters according to the names of the parameters

COMPSCI334

26

What if the method is "POST"

```
<HTML>
<HEAD>
  <TITLE>Collecting Three Parameters</TITLE>
</HEAD>
<H1 ALIGN="CENTER">Collecting Three Parameters</H1>
<FORM ACTION="/334/servlet/examples.DisplayParams" METHOD="POST">
  First Parameter: <INPUT TYPE="TEXT" NAME="First"><BR>
  Second Parameter: <INPUT TYPE="TEXT" NAME="Second"><BR>
  Third Parameter: <INPUT TYPE="TEXT" NAME="Third"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT">
  </CENTER>
</FORM>

</BODY>
</HTML>
```

method is post

COMPSCI334

27

If you process the post method in the same way as the get method, you can simply call the doGet method in doPost.

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

call doGet in doPost

COMPSCI334

28

- POST are used to send a large amount of information
- You might want to use a loop to enumerate through the parameters rather than list each one by hand
 - Use method request.getParameterNames()
 - the order of the returned parameters is unspecified

COMPSCI334

29

```
public void doPost(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML><TITLE>Display Parameters</TITLE>" +
               "<BODY>" +
               "<H1 ALIGN=CENTER> Parameters </H1>\n" +
               "<UL>\n");
    Enumeration paramNames = request.getParameterNames();
    while (paramNames.hasMoreElements()) {
        String paramName = (String)paramNames.nextElement();
        out.println("<LI><B>" + paramName + "</B>:" +
                  request.getParameter(paramName));
    }
    out.println("</UL>\n" + "</BODY></HTML>");
}
```

get the names of all the parameters

retrieve and display the value of each parameter

COMPSCI334

30

Address: <http://localhost:8080/334/servlet/examples.DisplayParams>

Parameters

- Second: 2
- First: 1
- Third: 3

COMPSCI334

31

Filtering Strings for HTML-Specific Characters

- HTML files contain some special characters, e.g. <, >, &, etc., which are used as directives in HTML.
- If we want to display a character which is the same as a special HTML character, to make sure that resulting page can be displayed correctly, we need to substitute the character as a sequence of other characters.
 - < → "<," > → ">," & → "&,"

COMPSCI334

32

Address F:\ssh\new_page_1.htm

if (a

```
<html>
<body>
if (a<b)
    doThis();
else
    doThat();
</body>
</html>
```

Address F:\ssh\new_page_1.htm

if (a < b) doThis(); else doThat();

```
<html>
<body>
if (a &lt; b)
    doThis();
else
    doThat();
</body>
</html>
```

COMPSCI334
33

```
public static String filter(String input) {
    StringBuffer filtered = new StringBuffer(input.length());
    char c;
    for(int i=0; i<input.length(); i++) {
        c = input.charAt(i);
        if (c == '<') {
            filtered.append("&lt;");
        } else if (c == '>') {
            filtered.append("&gt;");
        } else if (c == '"') {
            filtered.append("&quot;");
        } else if (c == '&') {
            filtered.append("&amp;");
        } else {
            filtered.append(c);
        }
    }
    return(filtered.toString());
}
```

This method checks each character in the input string. It replaces "<" with "<", etc.

COMPSCI334
34

Common HTTP 1.1 Request Headers

- Accept
 - Indicates MIME types browser can handle
- Accept-Encoding
 - Indicates encodings (e.g., gzip or compress) browser can handle.
- Authorization
 - User identification for password-protected pages.
- Connection
 - Persistent connections mean that the server can reuse the same socket over again for requests very close together from the same client (e.g., the images associated with a page)

COMPSCI334
35

- Cookie
 - Gives cookies previously sent to client.
- Host
 - Indicates host given in original URL
- If-Modified-Since
 - Indicates client wants page only if it has been changed after specified date
- Referer
 - URL of referring Web page
 - Useful for tracking traffic; logged by many servers
- User-Agent
 - for identifying category of client, e.g. Web browser, cell phone, etc.

COMPSCI334
36

Reading Request Headers

- Some time the request headers are important in processing client's requests
 - Compressed page
 - Image formats
- In this example, we want to display the request header being sent in by the browser.

COMPSCI334

37

Some Useful Methods

- General
 - `getHeader`: get the value of a specified request header
 - `getHeaderNames`: get the names of the request headers sent in by the browser
- Related info
 - `getMethod`: GET, POST, etc.
 - `getRequestURI`: part of this requested URL
 - No server name
 - Use `getRequestURL` if want to see the server name etc
 - `getProtocol`: HTTP/1.1, HTTP/1.0

COMPSCI334

38

Address  http://localhost:8080/334/index.html

Servlet Examples

[Hello World \(source\)](#)

[Collecting Parameters with GET \(source\)](#)

[Collecting Parameters with POST \(source\)](#)

[Show Request Headers \(source\)](#)

COMPSCI334

39

Showing Request Headers

Request Method: GET

Request URI: /334/servlet/examples.ShowRequestHeaders

Request Protocol: HTTP/1.1

Header Name	Header Value
User-Agent	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Accept	image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */*
Host	localhost:8080
Accept-Encoding	gzip, deflate
Accept-Language	en-nz
Referer	http://localhost:8080/334/index.html
Connection	Keep-Alive

```

...
<B>Request Method: </B> xxx <BR>
<B>Request URI: </B> xxx <BR>
<B>Request Protocol: </B> xxx <BR><BR>
<TABLE BORDER=1 ALIGN=CENTER>
<TR>
<TH>Header Name<TH>Header Value
<TR><TD> xxx <TD> xxx
<TR><TD> xxx <TD> xxx
...
</TABLE>
...

```

Use the print/println statement to generate the above html file.

COMPSCI334

41

```

public class ShowRequestHeaders extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("< HTML>< BODY><H1 ALIGN=CENTER> "+
            "Showing Request Headers</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" +
            "<TABLE BORDER=1 ALIGN=CENTER>\n" +
            "<TR>\n" +
            "<TH>Header Name<TH>Header Value");
    }
}

```

get various information of this request

COMPSCI334

42

```

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>");
}

```

get the name of all the headers

retrieve the value of each header

COMPSCI334

43

Sending Compressed Pages

- Gzipped files are compressed files. They can reduce the download time of long text pages.
- Recent browsers automatically uncompress the files whose "Content-Encoding" header marked as "gzip".
 - Need to check whether the browser can handle "gzip" file before sending the compressed file to the browser
 - request.getHeader("Accept-Encoding")
- To send a gzipped page, the servlet must use Java's GZIPOutputStream to generate the page.
 - If send a compressed file, need to tell the browser by setting the response header
 - response.setHeader("Content-Encoding", "gzip");

COMPSCI334

44

```

public class EncodedPage extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        String encodings = request.getHeader("Accept-Encoding");
        PrintWriter out;
        if ((encodings != null) &&
            (encodings.indexOf("gzip") != -1)) {
            OutputStream out1 = response.getOutputStream();
            out = new PrintWriter(new GZIPOutputStream(out1), false);
            response.setHeader("Content-Encoding", "gzip");
        } else {
            out = response.getWriter();
        }
    }
}

```

retrieve header value

check whether header exists

accept gzip format?

output will be in gzip format

COMPSCI334

45

```

out.println("<HTML><BODY>" );
String line = "Blah, blah, blah, blah, blah. ";
for(int i=0; i<10000; i++) {
    out.println(line);
}
out.println("</BODY></HTML>");
out.close();
}
}

```

COMPSCI334

46

Redirect pages

- In this example, when a user submits a form containing a search string, servlet SearchEngines processes the form.
 - SearchEngines redirects the search to google.
- Search string must be encoded
 - When browser sends data to the server, the browser replaces the space with "+" and all the other non-alphanumeric characters to "%XY" where XY is the hex value of the ASCII character
 - http://www.google.co.nz/search?q=at+t
 - means containing "at t": space is represented by "+"
 - what if the search string is "at + t"
 - replace "+" with "%2B"
 - http://www.google.co.nz/search?q=at+%2B+t

COMPSCI334

47

- URLCoder.encode(String s, String enc)
 - Translates a string into application/x-www-form-urlencoded format using a specific encoding scheme
 - UTF-8 for enc
- response.sendRedirect(String url)
 - Redirect a request to the specified url



COMPSCI334

48

```

<HTML>
<HEAD>
  <TITLE>Searching the Web</TITLE>
</HEAD>

<H1 ALIGN="CENTER">Searching the Web</H1>

<FORM ACTION="/334/servlet/examples.SearchEngines">
  <CENTER>
    Search String:
    <INPUT TYPE="TEXT" NAME="searchString"><BR>
  </CENTER>
</FORM>

</BODY>
</HTML>

```

COMPSCI334

49

- Retrieve the search string entered by the user
 - request.getParameter("searchString");
 - The string is decoded when retrieved
 - "at+%2B+t" → "at + t"
- Check whether the search string exists
 - null: something wrong with the html form
 - searchString.length() == 0: user did not enter a string
 - sendError(int, String)
 - the error status code
 - the descriptive message
- Encode the search string
 - URLEncoder.encode(searchString,"UTF-8")
- Redirect to google
 - response.sendRedirect("http://www.google.co.nz/search?q="+searchString);

COMPSCI334

50

```

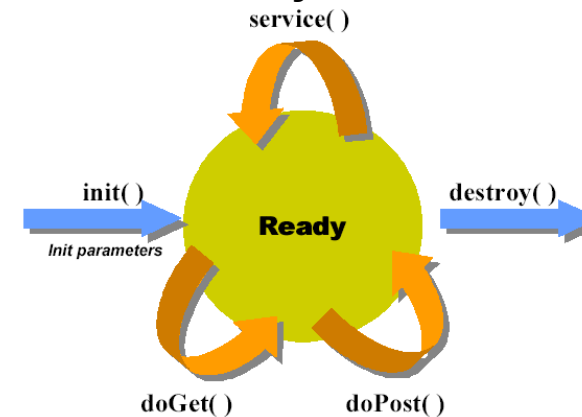
public class SearchEngines extends HttpServlet {
  public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    String searchString = request.getParameter("searchString");
    if ((searchString == null) ||
        (searchString.length() == 0)) {
      response.sendError(response.SC_BAD_REQUEST,
        "Missing search string.");
      return;
    }
    searchString = URLEncoder.encode(searchString,"UTF-8");
    response.sendRedirect("http://www.google.co.nz/search?q="+
      searchString);
  }
}

```

COMPSCI334

51

Servlet Life Cycle



COMPSCI334

52

- **Init()**
 - Executed **once** when the servlet is first loaded.
 - Not called for each request.
 - Perform any set-up in this method.
- **Service()**
 - Do not override this method!
- **doGet(), doPost(), doXxx()**
 - Handles GET, POST, etc. requests.
 - Override these to provide desired behavior.
- **Destroy()**
 - Called when server deletes servlet instance.
 - Perform any clean-up, and save data to be read by the next init().

COMPSCI334

53

Restricting Access to Web Pages

- Step 1
 - Check whether there is Authorization header in the request.
 - Authorization=Basic bWFydHk6bWFydHlwdw==
 - If not, go to Step 2.
 - If so, skip over word “basic” and reverse the base64 encoding of the remaining part.
 - This results in a string of the form username:password.
 - Check the username and password against some stored information.
 - If it matches, return the page.
 - If not, go to Step 2.

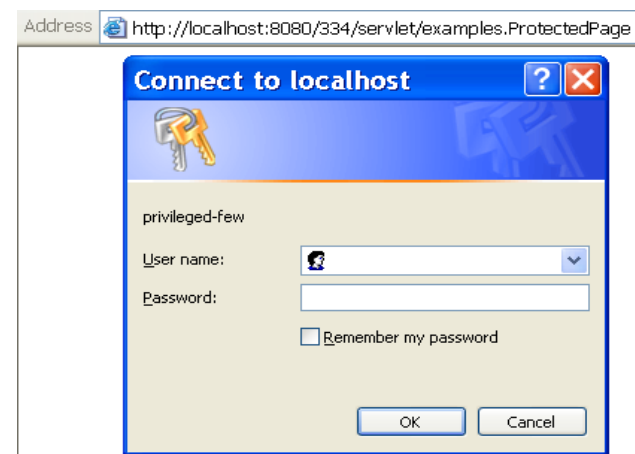
COMPSCI334

54

- Step 2
 - Return a 401 (Unauthorized) response code and a header of the following form:
 - WWW-Authenticate: BASIC realm="some-name"
 - The header instructs browser to pop up a dialog box telling the user to enter a name and password, then to reconnect with that username and password embedded in a single base64 string inside the Authorization header.

COMPSCI334

55



COMPSCI334

56

- Set up the passwords for users
 - Done in the init method when the servlet starts

```
public class ProtectedPage extends HttpServlet {
    private Properties passwords;
```

```
    public void init(ServletConfig config)
        throws ServletException {
        passwords = new Properties();
        passwords.put("abc001", "micky");
        passwords.put("abc002", "goofy");
    }
```

store UPIs and
passwords in a
property variable

COMPSCI334

57

- Obtain the value of the “Authorization” header
 - request.getHeader("Authorization");
- If the “Authorization” header is not empty, retrieve the user name and password
 - String userInfo = authorization.substring(6).trim();
 - Authorization=Basic bWFydHk6bWFydHlwdw==
 - password information starts at position 6 of the header value
 - Decode the BASE64 value to obtain the real value for user name and password
 - BASE64Decoder decoder = new BASE64Decoder();
 - String nameAndPassword = new String(decoder.decodeBuffer(userInfo));
 - Extract the user name and password from the decoded string
 - User name and password are separated by “:”
 - Java String APIs
 - Compare the user name and password with the corresponding ones stored on the server
 - passwords.getProperty(user);

COMPSCI334

58

```
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String authorization = request.getHeader("Authorization");
        if (authorization == null) {
            askForPassword(response);
        } else {
            System.out.println("Authorization="+authorization);
            String userInfo = authorization.substring(6).trim();
            BASE64Decoder decoder = new BASE64Decoder();
            String nameAndPassword =
                new String(decoder.decodeBuffer(userInfo));
```

if no password, ask user
provide password

password
information
starts at
position 6 of
the header
value

decode header value

COMPSCI334

59

```
int index = nameAndPassword.indexOf(":");
String user = nameAndPassword.substring(0, index);
String password = nameAndPassword.substring(index+1);
String realPassword = passwords.getProperty(user);

if ((realPassword != null) &&
    (realPassword.equals(password))) {
    out.println("<HTML><BODY>"+
        "<H1 ALIGN=CENTER>Welcome to the Protected Page"+
        "</H1>\n </BODY></HTML>");
} else {
    askForPassword(response);
}
}
```

retrieve password
and upi received
from the browser

get password
stored on server

COMPSCI334

60

- If the “Authorization” header is empty
 - Set the response status code
 - `response.setStatus(response.SC_UNAUTHORIZED);`
 - Ask browser opens a pop-up for entering user name and password
 - `response.setHeader("WWW-Authenticate", "BASIC realm=\"privileged-few\")");`

COMPSCI334

61

```
private void askForPassword(HttpServletResponse response) {
    response.setStatus(response.SC_UNAUTHORIZED); // Ie 401
    response.setHeader("WWW-Authenticate",
        "BASIC realm=\"privileged-few\"");
}
```

ask browser opens a pop-up
for entering user name and
password

COMPSCI334

62

Some HTTP 1.1 Response Headers

- Chapter 7 of *Core SERVLETS and JAVASERVER*
- Cache-Control
 - public: document is cacheable
 - private: document is for a single user and can only be stored in private caches
 - no-cache: document should never be cached
- Content-Disposition
 - request the browser ask the user to save the response to disk in a file of the given name
 - Content-Disposition: attachment; filename=file-name

COMPSCI334

63

- Content-Encoding
 - The way document is encoded.
- Content-Length
 - The number of bytes in the response.
- Content-Type
 - The MIME type of the document being returned.
- Expires
 - The time at which document should be considered out-of-date and thus should no longer be cached.
- Refresh
 - The number of seconds until browser should reload page.

COMPSCI334

64

- Set-Cookie
 - The cookies that browser should remember.
- WWW-Authenticate
 - The authorization type and realm needed in Authorization header.

COMPSCI334

65

Setting Arbitrary Response Headers

- `public void setHeader(String headerName, String headerValue)`
 - Sets an arbitrary header
- `public void setIntHeader(String name, int headerValue)`
 - Prevents need to convert int to String before calling `setHeader`

COMPSCI334

66

Setting Special Response Headers

- `setContentType`
 - Sets the Content-Type header.
 - text/html, application/msword
- `setContentLength`
 - Sets the Content-Length header.
- `addCookie`
 - Adds a value to the Set-Cookie header.

COMPSCI334

67

This program requests the browser refreshes its page every 2 seconds.

```
public class Refresh extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        response.setIntHeader("Refresh", 2); } set Refresh header
        out.println("<HTML><BODY>");
        out.println("Current Time is "+(new java.util.Date()));
        out.println("</BODY></HTML>");
    }
}
```

COMPSCI334

68

Handling Cookies

- Chapter 8 of *Core SERVLETS and JAVASERVER*
- Cookies are textual information that a web server sends to a browser.
 - Cookies are used by the web server to record information about the clients.
- When a browser sends a request to a site, it sends the cookies received from the site to the web server.

COMPSCI334

69

The Potential of Cookies

- Identifying a user during an e-commerce session
- Avoiding username and password
- Customizing a site
 - user can select the information to be displayed and the server remembers the user's choices
- Targeted advertising
 - advertisements based on user's interests

COMPSCI334

70

Problems with Cookies

- The problem is mainly privacy, not security
- Servers can remember your previous actions
- Poorly designed sites store sensitive information like credit card numbers directly in cookie

COMPSCI334

71

Sending Cookies to the Client

- Create a Cookie object
 - Call the Cookie constructor with a cookie name and a cookie value, both of which are strings.
 - `Cookie c = new Cookie("userID", "a1234");`
- Set the maximum age
 - To tell browser to store cookie on disk
 - `c.setMaxAge(60*60*24*7);` // One week
 - `c.setMaxAge(0)` tells the browser to delete the cookie
- Place the Cookie into the HTTP response
 - Use `response.addCookie`

COMPSCI334

72

Using Cookie Attributes

- `getMaxAge/setMaxAge`
 - Gets/sets the cookie expiration time (in seconds). If you fail to set this, cookie applies to current browsing session only.
- `getName`
 - Gets the cookie name. There is no `setName` method; you supply name to constructor. For incoming cookie array, you use `getName` to find the cookie of interest.
- `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. If you set the value of an incoming cookie, you still have to send it back out with `response.addCookie`.

COMPSCI334

73

Reading Cookies from the Client

- Call `request.getCookies`
 - This yields an array of `Cookie` objects
- Loop down the array, calling `getName` on each entry until you find the cookie of interest

```
String cookieName = "userID";
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for(int i=0; i<cookies.length; i++) {
        Cookie cookie = cookies[i];
        if (cookieName.equals(cookie.getName())) {
            doSomethingWith(cookie.getValue());
        }
    }
}
```

COMPSCI334

74

Address  http://localhost:8080/334/servlet/examples.SearchEnginesFrontEnd

Searching the Web

Search String:

COMPSCI334

75

- In this example, when a user visit servlet `SearchEnginesFrontEnd`, a form is shown.
 - Generate the form using `out.println`
- `SearchEnginesFrontEnd` fills in the “Search String” field with the string typed in by the user when the servlet is visited last time by the user (if any).
 - The search string typed in by the user last time is sent back to the server in a cookie
 - Retrieve the cookie containing the search string
 - `request.getCookies()`
 - `getCookieValue(cookies, "searchString");`
- When the “search” button is clicked, servlet “`CustomizedSearchEngines`” is called to redirect the search to google.

COMPSCI334

76

In the doGet method:

```
Cookie[] cookies = request.getCookies();  
String searchString = getCookieValue(cookies, "searchString");  
if (searchString==null) searchString="";
```

read cookies to find out
whether there is any search
string from previous visit

if no string found,
initialize the string to
empty

COMPSCI334

77

- Use println to generate the html code of the form
- The ACTION attribute of the FORM tag should be set to
"/334/servlet/examples.CustomizedSearchEngines"
 - CustomizedSearchEngines is the servlet which processes the search request
- The VALUE attribute of the INPUT tag should be set to the value of variable searchString
 - The INPUT tag shows the input field
 - "<INPUT TYPE=\"TEXT\" NAME=\"searchString\" VALUE=\"\"+searchString+\"\">
\n"

COMPSCI334

78

This method goes through all the cookies returned from the client to find out whether the cookie with the specified name exists

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

COMPSCI334

79

CustomizedSearchEngines

- Check the search string
 - Same as the redirecting page example
- Store the newly entered search string in a cookie and return the cookie to the client.
 - Create a cookie
 - Cookie searchStringCookie = new Cookie("searchString", searchString);
 - Set cookie expire time
 - searchStringCookie.setMaxAge(60*60*24);
 - Send cookie back
 - response.addCookie(searchStringCookie);
- Redirect search to google
 - Same as the redirecting page example

COMPSCI334

80

```

public class CustomizedSearchEngines extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        String searchString = request.getParameter("searchString");
        if ((searchString == null) || (searchString.length() == 0)) {
            response.sendError(response.SC_BAD_REQUEST,
                "Missing search string.");
        }
        return;
    }
}

```

search string cannot be empty

COMPSCI334

81

```

Cookie searchStringCookie =
    new Cookie("searchString", searchString);
searchStringCookie.setMaxAge(60*60*24);
response.addCookie(searchStringCookie);
searchString = URLEncoder.encode(searchString, "UTF-8");
response.sendRedirect("http://www.google.co.nz/
    search?q="+searchString);
}

```

send the cookie back to the client for next visit

redirect the search to google

COMPSCI334

82

Session Tracking

- Chapter 9 of *Core SERVLETS and JAVASERVER*
- Why session tracking?
 - HTTP protocol is stateless.
 - When doing on-line shopping, a customer might browse many pages and make many orders on different pages.
 - When a customer checks out, we need to find out the orders the customer made while in the shop.

COMPSCI334

83

Techniques for session tracking

- Using cookies
 - Send a cookie to the client when the client first visit the site.
 - Whenever the client comes back the same site, the cookie is sent back by the client's browser.
 - Problem
 - Client might disallow cookie

COMPSCI334

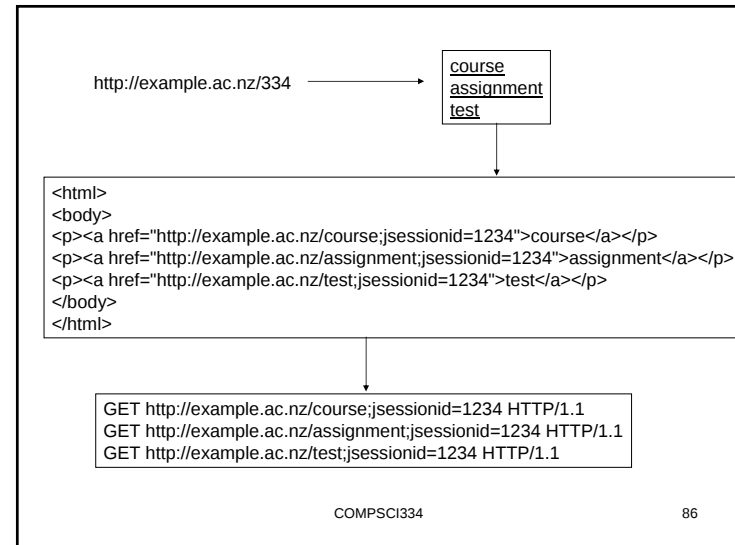
84

- URL-Rewriting

- Appends some extra data that identifies the session on the end of each URL that refers to the sever site
 - `http://host/path/file.html;session=xyz`
- Server associates that identifier with data it has stored about that session
- Advantage
 - Works even if cookies are disabled or unsupported
- Disadvantage
 - Must encode all URLs to include the session information that refer to your own site
 - all pages which contain reference to your site must be generated dynamically

COMPSCI334

85



COMPSCI334

86

- Hidden Form Fields

- `<INPUT TYPE="HIDDEN" NAME="session" VALUE="1234">`
 - the field is invisible to the user
 - when the form is submitted, the name, i.e. session, and the value, i.e. 1234, are automatically sent to the server
- when generating form, the servlet needs to generate the hidden field

COMPSCI334

87

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.

COMPSCI334

88

- Hashtable-like mechanism lets you store arbitrary objects inside session
 - setAttribute stores values
 - getAttribute retrieves values

COMPSCI334

89

Session Tracking Basics

- Access the session object
 - Call request.getSession to get HttpSession object
 - This is a hashtable associated with the user
- Look up information associated with a session.
 - Call getAttribute on the HttpSession object, cast the return value to the appropriate type, and check whether the result is null.
- Store information in a session.
 - Use setAttribute with a key and a value.
- Discard session data.
 - Call removeAttribute discards a specific value.
 - Call invalidate to discard an entire session.

COMPSCI334

90

- The session object is like a hash table
- setAttribute(String name, Object o)
- getAttribute(String name)
- getAttributeNames()

COMPSCI334

91

Session Tracking Basics

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

COMPSCI334

92

What if Server Uses URL Rewriting?

- Encode the URL that links back to the same site as the servlet:
 - Pass URL through `response.encodeURL()`.
 - If server is using cookies, this returns URL unchanged
 - If server is using URL rewriting, this appends the session info to the URL
- ```
String url = "order-page.html";
url = response.encodeURL(url);
```

COMPSCI334

93

- This example is a simple servlet that shows basic information about the client's session.
- When the client connects, the servlet retrieves the existing session. If there is no session, create a new one.
  - `HttpSession session = request.getSession(true);`
- The servlet looks for an attribute called `accessCount` of type `Integer`.
  - `Integer accessCount = (Integer)session.getAttribute("accessCount");`
  - `accessCount` does not exist, use 0 as the number of previous accesses
    - `accessCount = new Integer(0);`
  - Otherwise, increment the attribute by one
    - `accessCount = new Integer(accessCount.intValue() + 1);`
  - Store the new value to the session
    - `session.setAttribute("accessCount", accessCount);`

COMPSCI334

94

- Use a table to display various information about the session object
  - Session ID: `session.getId()`
  - Creation time: `session.getCreationTime()`
  - Last accessed time: `session.getLastAccessedTime()`

COMPSCI334

95

```
public class ShowSession extends HttpServlet {
 public void doGet(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 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);
 }
}
```

retrieve session

find attribute

initialize the attribute if it does not exist

increment the attribute if it already exists

COMPSCI334

96

```
// URL rewriting
String url = response.encodeURL
("http://localhost:8080/334/servlet/examples.ShowSession");
out.println("<HTML><BODY>\n" +
 "<H1 ALIGN=\"CENTER\">" + heading + "</H1>\n" +
 "<H2>Information on Your Session:</H2>\n" +
 "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
 "<TR >\n" +
 " <TH>Info Type <TH>Value\n" +
 "<TR>\n" +
 " <TD>ID\n <TD>" + session.getId() + "\n" +
 "<TR>\n" +
 " <TD>Creation Time\n "+
 " <TD>" + new Date(session.getCreationTime()) + "\n" +
 "</TABLE>\n" +
 "<CENTER>Refresh</CENTER>" +
 "</BODY></HTML>");
```

gain session ID

gain session creation time

COMPSCI334 97

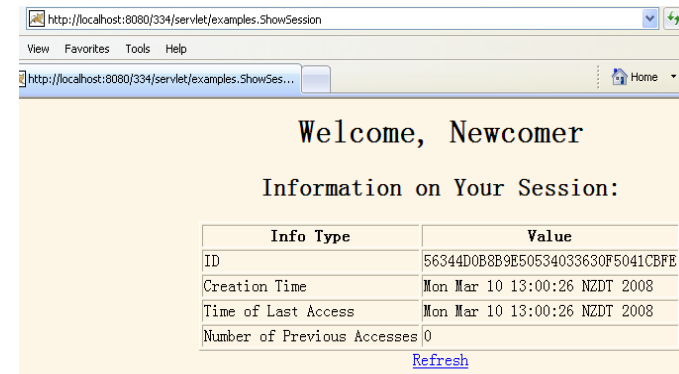
```
"<TR>\n" +
" <TD>Time of Last Access\n" +
" <TD>" + new Date(session.getLastAccessedTime()) + "\n" +
"<TR>\n" +
" <TD>Number of Previous Accesses\n" +
" <TD>" + accessCount + "\n" +
"</TABLE>\n" +
"<CENTER>Refresh</CENTER>" +
"</BODY></HTML>");
}
```

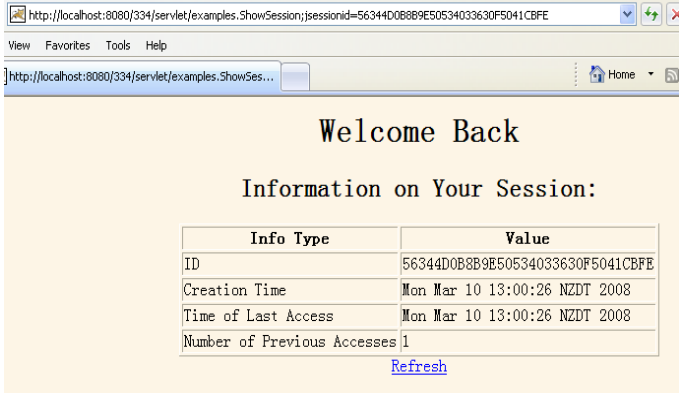
number of previous access

retrieve the time that the session was last sent from client

COMPSCI334 98

- To see how URL rewriting works, you need to disable the cookie
  - In IE browser, you can add localhost to the “Restricted sites” using “Internet Options” → security. Then, click “Restricted sites” → click the “Sites” button and use the pop up panel to enter the “http://localhost”.
- COMPSCI334 99





A screenshot of a web browser window. The address bar shows `http://localhost:8080/334/servlet/examples.ShowSession;jsessionid=56344D0B8B9E50534033630F5041CBFE`. The page content includes the text "Welcome Back" and "Information on Your Session:". Below this is a table with session details.

| Info Type                   | Value                            |
|-----------------------------|----------------------------------|
| ID                          | 56344D0B8B9E50534033630F5041CBFE |
| Creation Time               | Mon Mar 10 13:00:26 NZDT 2008    |
| Time of Last Access         | Mon Mar 10 13:00:26 NZDT 2008    |
| Number of Previous Accesses | 1                                |

Below the table is a [Refresh](#) link. At the bottom of the slide, the text "COMPSCI334" and "101" are visible.

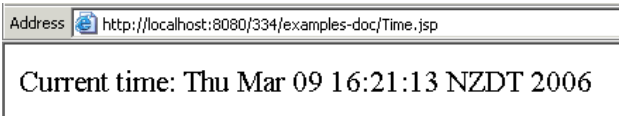
## The Need for JSP

- Chapter 10 and 11 of *Core SERVLETS and JAVASERVER*
- The main drawback of using servlet is that all the HTML text must be generated by `println` statement.
  - Web page writers must know programming.
- The idea behind JSP
  - A page contains a mixture of Java and HTML code
  - Use regular HTML for most of page
  - Mark Java code with special tags
  - Entire JSP page gets translated into a servlet (once), and servlet is what actually gets invoked (for each request)

COMPSCI334

102

```
<HTML>
<BODY>
Current time: <%= new java.util.Date() %>
</BODY>
</HTML>
```



A screenshot of a web browser window. The address bar shows `http://localhost:8080/334/examples-doc/Time.jsp`. The page content displays "Current time: Thu Mar 09 16:21:13 NZDT 2006". At the bottom of the slide, the text "COMPSCI334" and "103" are visible.

- JSP makes it easier to create and maintain HTML, while still providing full access to servlet code
- JSP pages get translated into servlets the first time the pages are requested
  - Servlet code gets executed. The original JSP page is totally ignored.
  - Page translation does not occur for each request.

COMPSCI334

104

## Basic Syntax

- HTML Text
  - `<H1>Blah</H1>`
  - Passed through to client. Really turned into servlet code that looks like
    - `out.print("<H1>Blah</H1>");`
- JSP Comments
  - `<%-- Comment --%>`
  - Not sent to client
  - To get `<%` in output, use `<\%`

COMPSCI334

105

- Expressions
  - `<%= Java Expression %>`
  - Expression evaluated, converted to String, and placed into HTML page at the place it occurred in JSP page
  - `<%= new java.util.Date() %>`

COMPSCI334

106

## JSP/Servlet Correspondence

- **Original JSP**

```
<H1>A Random Number</H1>
<%= Math.random() %>
```

- **Possible resulting servlet code**

```
public void _jspService(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 HttpSession session = request.getSession(true);
 JspWriter out = response.getWriter();
 out.println("<H1>A Random Number</H1>");
 out.println(Math.random());
 ...
}
```

## Predefined Variables

- request
  - The `HttpServletRequest`
- response
  - The `HttpServletResponse`
- out
  - The `JspWriter` used to send output to the client
- session
  - The `HttpSession` associated with the request

COMPSCI334

108

## JSP Scriptlets

- A collections of Java code enclosed with `<% %>`
  - `<% Java code %>`
  - `<%`  
String queryData = request.getQueryString();  
out.println("Attached GET data: " + queryData);  
`%>`

COMPSCI334

109

- Scriptlets need not be complete Java expressions
- Complete expressions are usually clearer and easier to maintain.

### Example

```
- <% if (Math.random() < 0.5) { %>
 Have a nice day!
 <% } else { %>
 Have a lousy day!
 <% } %>
```

### Representative result

```
- if (Math.random() < 0.5) {
 out.println("Have a nice day!");
} else {
 out.println("Have a lousy day!");
}
```

COMPSCI334

110

## JSP Declarations

```
<HTML>
<BODY>
<H1>JSP Declarations</H1>

<%! private int accessCount = 0; %>
<H2>Accesses to page since server reboot:
<%= ++accessCount %></H2>

</BODY>
</HTML>
```

The scope of the declaration is in a JSP file.

COMPSCI334

111

## JSP Declarations

Accesses to page since server reboot: 1

## JSP Declarations

Accesses to page since server reboot: 2

COMPSCI334

112

## JSP page Directive:

- By default, the servlet imports java.lang.\*, javax.servlet.\*, javax.servlet.jsp.\*, javax.servlet.http.\* packages.
  - Different servers might also import various other Java packages.
- To import other packages, the following forms are used:
  - `<%@ page import="package.class" %>`
  - `<%@ page import="package.class1,...,package.classN" %>`

COMPSCI334

113

```
<HTML>
<BODY>
<%@ page import="java.math.BigInteger" %>
<%! BigInteger bi = new java.math.BigInteger("12345678901234567890"); %>
Number is <%= bi.toString() %>
</BODY>
</HTML>
```

COMPSCI334

114

## Including Files in JSP Documents

- Chapter 13 of *Core SERVLETS and JAVASERVER*
- To reuse JSP, HTML, or plain text content
- Use `jsp:include` tag
  - `<jsp:include page="file-name" />`
  - `<jsp:include page="COMPSCI334.html" />`
- Use include directive
  - `<%@ include file="file-name" %>`
  - `<%@ include file="AccessCount.jsp" %>`

COMPSCI334

115

```
<HTML>
<BODY>

<jsp:include page="COMPSCI334.html" />
<jsp:include page="COMPSCI734.html" />

</BODY>
</HTML>
```

Address  http://localhost:8080/334/examples-doc/Include.jsp

1. This is COMPSCI334
2. This is COMPSCI734

COMPSCI334

116

```

<%!
private int accessCount = 0;
private String accessHost = null;
%>
<P>
<HR>
This page has been accessed <%= ++accessCount %>
times since server reboot.
<% if (accessHost != null) { %>
 It was last accessed from
 <%= accessHost %>.
<% }
 accessHost = request.getRemoteHost(); %>

```

} JSP file to be included

COMPSCI334

117

```

<HTML>
<BODY>
<%@ page import="java.util.Date" %>
Current time: <%= new java.util.Date() %>
<%@ include file="AccessCount.jsp" %>
</BODY>
</HTML>

```

COMPSCI334

118

Current time: Thu Apr 10 10:11:22 NZST 2008

This page has been accessed 1 times since server reboot.

COMPSCI334

119

Current time: Thu Apr 10 10:14:06 NZST 2008

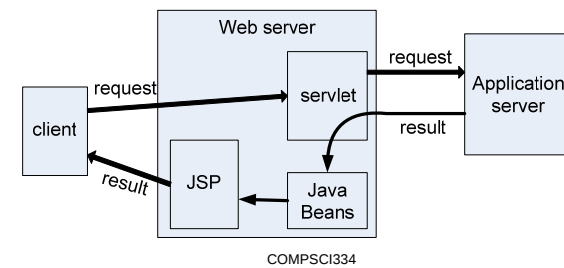
This page has been accessed 2 times since server reboot.  
It was last accessed from 127.0.0.1.

COMPSCI334

120

## Using JavaBeans with JSP

- Chapter 14 of *Core SERVLETS and JAVASERVER*
- In JSP pages, we should avoid using complicated programming logic.



120

## What Are Beans?

- Java classes that follow certain conventions
  - Must have a zero-argument (empty) constructor
    - You can satisfy this requirement either by explicitly defining such a constructor or by omitting all constructors
  - Should have no public instance variables (fields)
  - Class variables should be accessed through methods called *getXxx* and *setXxx*
    - If class has method *getTitle* that returns a *String*, class is said to have a *String property* named *title*
    - Boolean properties use *isXxx* instead of *getXxx*

COMPSCI334

121

```
package examples;
public class StringBean {
 private String message = "No message specified";
 public String getMessage() {
 return(message);
 }
 public void setMessage(String message) {
 this.message = message;
 }
}
```

- The file should be placed in `webapps\334\WEB-INF\classes\examples` folder
- Compile the file with `javac StringBean.java`

COMPSCI334

122

## Basic Bean Use in JSP

- Format
  - `<jsp:useBean id="name" class="package.Class" />`
  - If an **existing bean** is to be accessed, it is **retrieved**. If a bean needs to be created, it is **instantiated** without explicit Java programming.
- Example
  - `<jsp:useBean id="sb" class="examples.StringBean" />`
  - Same as

```
<% examples.StringBean sb = new examples.StringBean ();
pageContext.setAttribute("sb", sb, PageContext.PAGE_SCOPE);
%>
```

COMPSCI334

123

## Accessing Bean Properties

- Format
  - `<jsp:getProperty name="name" property="property" />`
- Example
  - `<jsp:getProperty name="sb" property="message" />`
  - Same as `<%= sb.getMessage() %>`

COMPSCI334

124

## Setting Bean Properties:

- Format

- `<jsp:setProperty name="name" property="property" value="value" />`

- Example

- `<jsp:setProperty name="sb" property="message" value="welcome" />`
  - Same as `<% sb.setMessage("welcome"); %>`

COMPSCI334

125

```
<HTML>
<BODY>
<jsp:useBean id="stringBean" class="examples.StringBean" />
```

```

```

```
Initial value (getProperty):
```

```
<I><jsp:getProperty name="stringBean"
 property="message" /></I>
```

```
Initial value (JSP expression):
```

```
<I><%= stringBean.getMessage() %></I>
```

COMPSCI334

126

```
<jsp:setProperty name="stringBean"
 property="message"
 value="Welcome 1" />
```

Value after setting property with setProperty:

```
<I><jsp:getProperty name="stringBean"
 property="message" /></I>
```

```
<%= stringBean.setMessage("Welcome 2"); %>
```

Value after setting property with scriptlet:

```
<I><%= stringBean.getMessage() %></I>
```

```

```

```
</BODY>
```

```
</HTML>
```

COMPSCI334

127

Address  http://localhost:8080/334/examples-doc/StringBean.jsp

1. Initial value (getProperty): *No message specified*
2. Initial value (JSP expression): *No message specified*
3. Value after setting property with setProperty: *Welcome 1*
4. Value after setting property with scriptlet: *Welcome 2*

COMPSCI334

128

## Associating Individual Properties with Input Parameters

- Format
  - `<jsp:setProperty name="Bean's ID" property="Bean's property" param="form's parameter name" />`
- Example
  - `<jsp:setProperty name="stringBean" property="message" param="input" />`

COMPSCI334

129

```
<HTML>
<BODY>
<FORM ACTION="property.jsp">
Enter message here: <INPUT TYPE="text" NAME="input">

<INPUT TYPE="SUBMIT" VALUE="Submit">
</FORM>
</BODY>
</HTML>
```

Address  http://localhost:8080/334/examples-doc/Property.html

Enter message here:

COMPSCI334

130

```
<HTML>
<BODY>
<jsp:useBean id="stringBean" class="examples.StringBean" />
<jsp:setProperty name="stringBean" property="message" param="input" />
New value in Bean: <jsp:getProperty name="stringBean" property="message" />
</BODY>
</HTML>
```

Address  http://localhost:8080/334/examples-doc/property.jsp?input=abcd

New value in Bean: abcd

COMPSCI334

131

## Associating All Properties with Input Parameters

- If the names of the parameters in a form are the same as the names of the properties of a Bean, you can assign the values of the parameters directly to the corresponding properties in the Bean.
  - no action is taken when an input parameter is missing
- Format
  - `<jsp:setProperty name="Bean's ID" property="*" />`

COMPSCI334

132

```

public class StringBean2 {
 private String message1 = "No message specified";
 private String message2 = "No message specified";

 public String getMessage1() {
 return(message1);
 }
 public String getMessage2() {
 return(message2);
 }

 public void setMessage1(String message) {
 this.message1 = message;
 }
 public void setMessage2(String message) {
 this.message2 = message;
 }
}

```

Bean with two properties

COMPSCI334

133

```

<HTML>
<BODY>
<FORM ACTION="property2.jsp">
Enter message1 here: <INPUT TYPE="text" NAME="message1" size="20">

Enter message2 here: <INPUT TYPE="text" NAME="message2" size="20">

Enter message3 here: <INPUT TYPE="text" NAME="message3" size="20">

<INPUT TYPE="SUBMIT" VALUE="Submit">
</FORM>
</BODY>
</HTML>

```

Address  http://localhost:8080/334/examples-doc/Property2.html

Enter message1 here:

Enter message2 here:

Enter message3 here:

COMPSCI334

134

```

<HTML>
<BODY>
<jsp:useBean id="stringBean" class="examples.StringBean2" />
<jsp:setProperty name="stringBean" property="*" />
New value in message1: <jsp:getProperty name="stringBean"
 property="message1" />

New value in message2: <jsp:getProperty name="stringBean"
 property="message2" />
</BODY>
</HTML>

```

Address  http://localhost:8080/334/examples-doc/Property2.html

Enter message1 here:

Enter message2 here:

Enter message3 here:

Address  http://localhost:8080/334/examples-doc/Property2.html

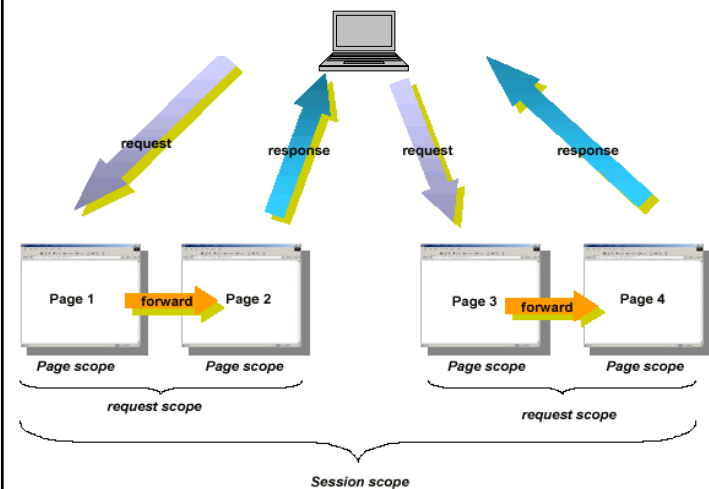
New value in message1: 1

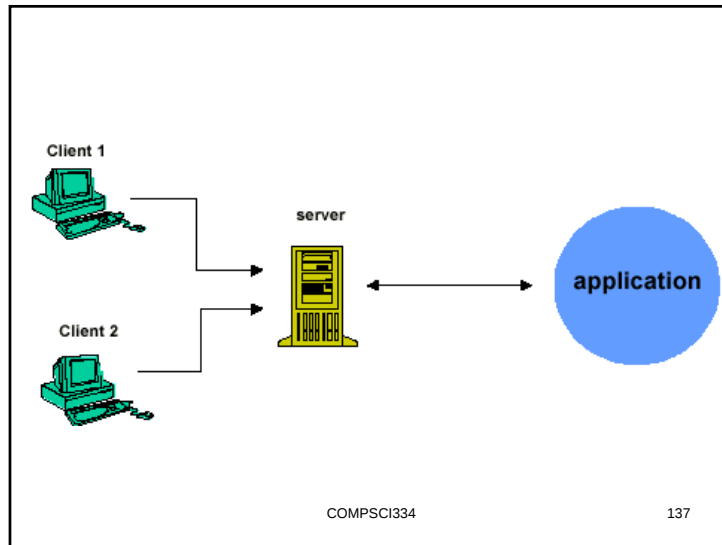
New value in message2: 2

COMPSCI334

135

## The Scope of JavaBean





- When you store beans in different scopes, you should use different names for each bean. Otherwise, servers might be confused and retrieve the wrong bean.

COMPSCI334

138

```
<HTML>
<BODY>
<jsp:useBean id="acb_p" class="examples.AccessCountBean" scope="page" />
This site has been visited
<jsp:getProperty name="acb_p" property="accessCount" />
Times.
<jsp:setProperty name="acb_p" property="accessCountIncrement" value="1" />
</BODY>
</HTML>
```

Address http://localhost:8080/334/examples-doc/ScopePage.jsp

This site has been visited 1 Times.

COMPSCI334

139

```
<HTML>
<BODY>
<jsp:useBean id="acb_s" class="examples.AccessCountBean" scope="session" />
This site has been visited
<jsp:getProperty name="acb_s" property="accessCount" />
Times.
<jsp:setProperty name="acb_s" property="accessCountIncrement" value="1" />
</BODY>
</HTML>
```

load and refresh once

Address http://localhost:8080/334/examples-doc/ScopeSession.jsp

This site has been visited 2 Times.

Address http://localhost:8080/334/examples-doc/ScopeSession.jsp

This site has been visited 1 Times.

load in another window

COMPSCI334

140

```

<HTML>
<BODY>
<jsp:useBean id="acb_a" class="examples.AccessCountBean" scope="application" />
This site has been visited
<jsp:getProperty name="acb_a" property="accessCount" />
Times.
<jsp:setProperty name="acb_a" property="accessCountIncrement" value="1" />
</BODY>
</HTML>

```

load and refresh once



COMPSCI334

141

### Difference between page and request scope

```

<HTML>
<BODY>
<jsp:useBean id="acb" class="examples.AccessCountBean" scope="page" />
This site has been visited
<jsp:getProperty name="acb" property="accessCount" />
Times.
<jsp:setProperty name="acb" property="accessCountIncrement" value="1" />

<jsp:include page="ScopeIncludePage.jsp" />
</BODY>
</HTML>

```

ScopeIncludePage.jsp

From included page: <BR>  
 <jsp:useBean id="acb" class="examples.AccessCountBean" scope="page" />  
 This site has been visited  
 <jsp:getProperty name="acb" property="accessCount" />  
 Times.  
 <jsp:setProperty name="acb" property="accessCountIncrement" value="1" />

COMPSCI334

142

Address <http://localhost:8080/334/examples-doc/ScopeDiff.jsp>

This site has been visited 1 Times.  
 From included page:  
 This site has been visited 1 Times.

the count from the included page starts from 1

COMPSCI334

143

```

<HTML>
<BODY>
<jsp:useBean id="acb" class="examples.AccessCountBean" scope="request" />
This site has been visited
<jsp:getProperty name="acb" property="accessCount" />
Times.
<jsp:setProperty name="acb" property="accessCountIncrement" value="1" />

<jsp:include page="ScopeInclude.jsp" />
</BODY>
</HTML>

```

ScopeInclude.jsp

From included page: <BR>  
 <jsp:useBean id="acb" class="examples.AccessCountBean" scope="request" />  
 This site has been visited  
 <jsp:getProperty name="acb" property="accessCount" />  
 Times.  
 <jsp:setProperty name="acb" property="accessCountIncrement" value="1" />

COMPSCI334

144

Address  http://localhost:8080/334/examples-doc/ScopeRequest.jsp

This site has been visited 1 Times.  
From included page:  
This site has been visited 2 Times.

the bean used in the included page is the same as the one used in ScopeRequest.jsp

COMPSCI334

145

## Integrating Servlets and JSP

- Chapter 15 of *Core SERVLETS and JAVASERVER*
- Why Combine Servlets & JSP?
  - The assumption behind JSP is that a *single* page gives a *single* basic look
  - For complex processing, starting with JSP is awkward

COMPSCI334

146

## Joint servlet/JSP process

- Original request is answered by a servlet
- Servlet processes request data, does database lookup, business logic, etc.
- Results are placed in beans
- Request is forwarded to a JSP page to format result
- Different JSP pages can be used to handle different types of presentation

COMPSCI334

147

## Dispatching Requests from Servlets to JSP Pages

- First obtain the servlet context.
  - There is one context per "web application".
  - A "web application" is a collection of servlets and content installed under the server's URL namespace.
    - E.g. all the servlets, JSP file, etc. under the 334 directory.
  - `getServletContext()`

COMPSCI334

148

- Call the `getRequestDispatcher` method of `ServletContext`

```
String url = "/examples-doc/Integrate11.jsp";
RequestDispatcher dispatcher =
 getServletContext().getRequestDispatcher(url);
```
- Call forward to completely transfer control to destination page (no communication with client)

```
dispatcher.forward(request, response);
```

COMPSCI334

149

## jsp:useBean

- The JSP page should not create the objects
  - The servlet, not the JSP page, should create all the data objects. So, to guarantee that the JSP page will not create objects, you should use

```
<jsp:useBean ... type="package.Class" />
```

 instead of

```
<jsp:useBean ... class="package.Class" />
```

COMPSCI334

150

## Storing Data for Later Use: The Current Request

- Purpose
  - Storing data that servlet looked up and that JSP page will use only in this request.
- Servlet syntax to store data

```
SomeClass value = new SomeClass(...);
request.setAttribute("key", value);
```
- JSP syntax to retrieve data

```
<jsp:useBean
 id="key"
 type="somepackage.SomeClass"
 scope="request" />
```

COMPSCI334

151

## Storing Data for Later Use: The Current Session

- Purpose
  - Storing data that servlet looked up and that JSP page will use in this request and in later requests from same client.
- Servlet syntax to store data

```
SomeClass value = new SomeClass(...);
HttpSession session = request.getSession(true);
session.setAttribute("key", value);
```
- JSP syntax to retrieve data

```
<jsp:useBean
 id="key"
 type="somepackage.SomeClass"
 scope="session" />
```

COMPSCI334

152

## Storing Data for Later Use: The Application

- Purpose
  - Storing data that servlet looked up and that JSP page will use in this request and in later requests from *any* client.
- Servlet syntax to store data

```
SomeClass value = new SomeClass(...);
getServletContext().setAttribute("key", value);
```
- JSP syntax to retrieve data

```
<jsp:useBean
 id="key"
 type="somepackage.SomeClass"
 scope="application" />
```

COMPSCI334

153

- This example shows the information of a student using the JSP page chosen by the user (assume different pages lay out the information differently)
- The servlet stores the retrieved information in a StudentBean and forwards the requests to the selected JSP page

COMPSCI334

154

Address http://localhost:8080/334/examples-doc/Integrate.html

ID:

Select Page:

☐ Page1

☐ Page2

Address http://localhost:8080/334/servlet/examples.Integrate?id=1&page=page1

This is Page 1.  
id = 1  
record = abc

```
package examples;
public class Student {
 private String record;
 private int id;

 public Student(String record, int id) {
 this.record = record;
 this.id = id;
 }
 public String getRecord() {
 return record;
 }
 public int getId() {
 return id;
 }
}
```

```
package examples;
public class StudentBean {
 private String record;
 private int id;

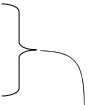
 public String getRecord() {
 return record;
 }
 public int getId() {
 return id;
 }
 public void setRecord(String record) {
 this.record = record;
 }
 public void setId(int id) {
 this.id = id;
 }
}
```

COMPSCI334

156

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

```
public class Integrate extends HttpServlet {
 private Student[] student= new Student[2];
 public void init() throws ServletException {
 student[0] = new Student("abc", 1);
 student[1] = new Student("def", 2);
 }
}
```



create two Student  
objects for the servlet

COMPSCI334

157

```
public void doGet(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
 int id = Integer.parseInt(request.getParameter("id"));
```

```
 // create a bean for holding student's data
 StudentBean sb = new StudentBean();
```

```
 // set up the values in the bean
 // the values of the bean are from the Student object
 // with index given by the user
 sb.setRecord(student[id-1].getRecord());
 sb.setId(student[id-1].getId());
```

COMPSCI334

158

```
 // store the bean in session object
 HttpSession session = request.getSession(true);
 session.setAttribute("student", sb);

 // determine which JSP page to forward to
 // according to user's choice
 String url;
 if (request.getParameter("page").equals("page1"))
 url = "/examples-doc/Integrate1.jsp";
 else url = "/examples-doc/Integrate2.jsp";

 // forward to the selected JSP page
 RequestDispatcher dispatcher =
 getServletContext().getRequestDispatcher(url);
 dispatcher.forward(request, response);
}
}
```

COMPSCI334

159

```
<HTML>
This is Page 1.

<BODY>
<jsp:useBean id="student"
 type="examples.StudentBean"
 scope="session" />
id = <jsp:getProperty name="student"
 property="id" />

record = <jsp:getProperty name="student"
 property="record" />

</BODY>
</HTML>
```

COMPSCI334

160

## Including Pages Instead of Forwarding to Them

- With the forward method of RequestDispatcher
  - Control is permanently transferred to new page
  - Original page cannot generate any output
- With the include method of RequestDispatcher
  - Control is temporarily transferred to new page
  - Original page can generate output before and after the included page

COMPSCI334

161

```
public class ServletInclude extends HttpServlet {
 String url = "/examples-doc/Secret.jsp";
 public void doGet(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
 // tell the browser not to cache the page
 response.setHeader("Cache-Control", "no-cache");
 RequestDispatcher dispatcher =
 getServletContext().getRequestDispatcher(url);
 dispatcher.include(request, response);
 }
}
```

COMPSCI334

162

## JSP expression language

- Main drawbacks of using JSP tag, e.g. `jsp:useBean`, `jsp:getProperty`
  - Cannot access bean subproperties
- An expression language makes it possible to easily access application data stored in JavaBeans components.
  - More concise, succinct, and readable syntax
  - Ability to access subproperties
  - Ability to access collections

COMPSCI334

163

- The JSP expression language allows a page author to access a bean using simple syntax such as `${name}` for a simple variable or `${name.id}` for a nested property
  - `${stringBean}`
  - `${stringBean.message}`
    - `<jsp:useBean id="stringBean" class="examples.StringBean" />`
    - `<jsp:getProperty name="stringBean" property="message" />`

COMPSCI334

164

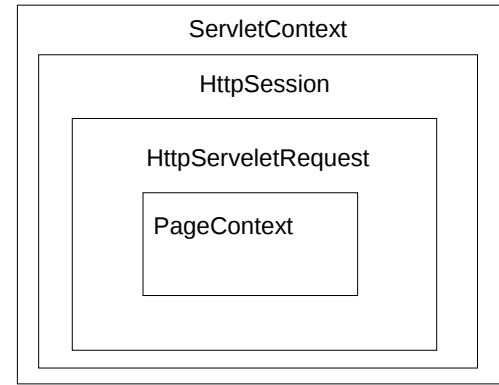
## Accessing scoped variables

- A servlet invokes code that create data, then forwards the request to a JSP page
- The servlet needs to use **setAttribute** to store data in one of the following locations
  - HttpServletRequest
  - HttpSession
  - ServletContext
- A JSP page can access the value as `${name}`

COMPSCI334

165

## Retrieving values in JSP



COMPSCI334

166

- JSP searches for an attribute value in the following order
  - PageContext
  - HttpServletRequest
  - HttpSession
  - ServletContext

COMPSCI334

167

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

 // store "attribute1" in HttpServletRequest object
 request.setAttribute("attribute1", "First Value");
 HttpSession session = request.getSession();
 // store "attribute2" in HttpSession object
 session.setAttribute("attribute2", "Second Value");
 // store "attribute3" in ServletContext object
 ServletContext application = getServletContext();
 application.setAttribute("attribute3",
 new java.util.Date());
}
```

COMPSCI334

168

```
// store "repeated" in HttpServletRequest object
request.setAttribute("repeated", "Request");
// store "repeated" in HttpSession object
session.setAttribute("repeated", "Session");
// store "repeated" in ServletContext object
application.setAttribute("repeated", "ServletContext");
// forward request to a JSP page
RequestDispatcher dispatcher =
 request.getRequestDispatcher("/examples-doc/scoped-vars.jsp");
dispatcher.forward(request, response);
}
```

COMPSCI334

169

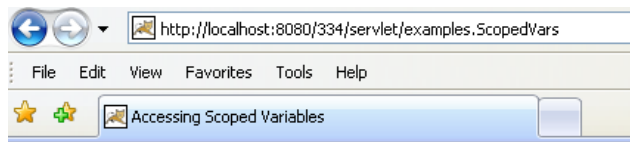
```
<HTML>
<HEAD><TITLE>Accessing Scoped Variables</TITLE>
</HEAD>
<BODY>

 attribute1: ${attribute1}
 attribute2: ${attribute2}
 attribute3: ${attribute3}
 Source of "repeated" attribute: ${repeated}

</BODY>
</HTML>
```

COMPSCI334

170



- **attribute1:** First Value
- **attribute2:** Second Value
- **attribute3:** Fri Mar 14 15:15:14 NZDT 2008
- **Source of "repeated" attribute:** Request

COMPSCI334

171

## Creating Custom JSP Tag Libraries

- Chapter 7 & 8 of *Core Servlets and JavaServer Pages, Volume 2: Advanced Technologies, Second Edition*
- Improve the flexibility and power of JSP pages
- Separate JSP page content from behavior:
  - Content authors author content.
  - Code developers create tags.

COMPSCI334

172

## Components That Make Up a Tag Library

- The Tag Handler Class
  - Java code that says how to actually translate tag into code
  - Usually extends [javax.servlet.jsp.tagext.SimpleTagSupport](#)
  - Goes in same directories as servlet class files
    - On Tomcat, the tag class files must be in a package.

COMPSCI334

173

- The Tag Library Descriptor File
  - XML file describing tag name, attributes, and tag handler class
  - Normally stored with the JSP file
- The JSP File
  - Imports a tag library (referencing URL of descriptor file)
  - Defines tag prefix, e.g. 334:simplePrime
  - Uses tags

COMPSCI334

174

JSP

<334:simpleTag />

browser

Welcome

COMPSCI334

175

- This example shows how to implement a tag handler class
  - Display a message
  - Body is empty: <334:simpleTag />
- The tag handler class should extend [SimpleTagSupport](#) class
  - Need to overwrite the doTag method
- In the [doTag](#) method
  - Display the message corresponds to the tag
  - Obtain a JspWriter object for sending response back to the browser
    - `JspWriter out = getJspContext().getOut();`
  - Send response using print
    - `out.print("Welcome");`

COMPSCI334

176

```

package examples.tags;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;

public class SimpleTag extends SimpleTagSupport
{
 public void doTag() throws JspException, IOException
 {
 JspWriter out = getJspContext().getOut();
 out.print("Welcome");
 }
}

```

needed packages

COMPSCI334

177

## Defining a Simple Tag Library Descriptor

- Start with XML header and DOCTYPE
- Top-level element is taglib
- Each tag defined by tag element with:
  - name, whose body defines the base tag name.
    - <name>simpleTag</name>
  - tagclass, which gives the fully qualified class name of the tag handler.
    - <tagclass>examples.tags.SimpleTag</tagclass>
  - bodycontent, which specifies if the tag contains content between start and end tag
    - <bodycontent>empty</bodycontent>
  - info, which gives a short description.
    - <info>Outputs a welcome message.</info>

COMPSCI334

178

```

<taglib>
...
 <tag>
 <name>simpleTag</name>
 <tagclass>examples.tags.SimpleTag</tagclass>
 <bodycontent>empty</bodycontent>
 <info>Outputs a welcome message.</info>
 </tag>
...
</taglib>

```

COMPSCI334

179

## Accessing Custom Tags From JSP Files

- Import the tag library
  - Specify location of TLD file
  - Define a tag prefix (namespace)
  - **<%@ taglib uri="334-taglib.tld" prefix="334" %>**
- Use the tags
  - <prefix:tagName />
  - Tag name comes from TLD file
  - Prefix comes from taglib directive
  - <334:simpleTag />

COMPSCI334

180

```

<HTML>
<BODY>
<H1>Simple Tag Example</H1>
<%@ taglib uri="334-taglib.tld" prefix="334" %>
<334:simpleTag />
</BODY>
</HTML>

```

Address  http://localhost:8080/334/examples-doc/SimpleTag.jsp

# Simple Tag Example

Welcome

COMPSCI334

181

## Assigning Attributes to Tags

- Allowing tags like
 

```

<prefix:name
 attribute1="value1"
 attribute2="value2"
 ...
 attributeN="valueN"
/>

```

COMPSCI334

182

## Attributes: The Tag Handler Class

- Use of an attribute called attribute1 simply results in a call to a method called setAttribute1
  - Attribute value is supplied to method as a String
  - Use a class variable to hold the value of the attribute
- To support <prefix:tagName attribute1="x" />, add the following to tag handler class:
 

```

public void setAttribute1(String value1) {
 doSomethingWith(value1);
}

```

COMPSCI334

183

- In this example, we want to provides two attributes to the welcomeTag
  - colour: the colour of the word being displayed
  - repeat: the number of times that the word to be displayed
- Declare two class variables to hold the values of the two attributes
  - colour, repeat
- Need to implement two methods to set the values of the two variables
  - setColour, setRepeat
- The doTag method needs to output the html code to display word "Welcome" in the specified colour and repeat the html code according to the value in the repeat variable
  - <font color="black">Welcome </font>
  - <font color="black">Welcome </font>< <font color="black">Welcome </font><

COMPSCI334

184

```

package examples.tags;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;

public class WelcomeTag extends SimpleTagSupport {
 // there are two attribute, repeat and colour
 private int repeat = 1;
 private String colour = "black";

 // method for setting up the value of "colour"
 public void setColour(String colour) {
 this.colour = colour;
 }
}

```

COMPSCI334

185

```

// method for setting up the value of "repeat"
public void setRepeat(String repeat) {
 try {
 this.repeat = Integer.parseInt(repeat);
 } catch (NumberFormatException nfe) {
 this.repeat = 2;
 }
}
}

```

COMPSCI334

186

<334:welcomeTag />

With no attributes: <font color="black">Welcome </font><BR>

<334:welcomeTag colour="red" />

With "colour": <font color="red">Welcome </font><BR>

<334:welcomeTag colour="blue" repeat="3" />

With both attributes: <font color="blue">Welcome </font><font color="blue">Welcome </font><font color="blue">Welcome </font><BR>

COMPSCI334

187

html code for displaying "Welcome" in blue  
<font color="blue">Welcome </font>

```

public void doTag() throws JspException, IOException
{
 JspWriter out = getJspContext().getOut();
 for (int i = 0; i < repeat; i++)
 out.print("" + "Welcome " + "");
 out.println("
");
}
}

```

COMPSCI334

188

## Attributes: The Tag Library Descriptor File

- The tag element must contain a nested attribute element
- The attribute element has three further nested elements
  - name, a required element that defines the case-sensitive attribute name. `<name>colour</name>`
  - required, a required element that stipulates whether the attribute must always be supplied (true) or is optional (false). E.g. `<required>false</required>`
  - rtexprvalue, an optional attribute that indicates whether the attribute value can be a JSP expression like `<%= expression %>` (true) or whether it must be a fixed string (false). The default value is false.

COMPSCI334

189

```
<tag>
 <name>welcomeTag</name>
 <tagclass>examples.tags.WelcomeTag</tagclass>
 <bodycontent>empty</bodycontent>
 <info>Outputs a welcome message.</info>
 <attribute>
 <name>colour</name>
 <required>false</required>
 </attribute>
 <attribute>
 <name>repeat</name>
 <required>false</required>
 </attribute>
</tag>
```

COMPSCI334

190

## Using welcomeTag Tag

<HTML>

<BODY>

<H1>Simple Tag Example</H1>

<%@ taglib uri="334-taglib.tld" prefix="334" %>

With no attributes: `<334:welcomeTag />`

With "colour": `<334:welcomeTag colour="red" />`

With both attributes: `<334:welcomeTag colour="blue" repeat="3" />`

</BODY>

</HTML>

COMPSCI334

191



COMPSCI334

192

## Including the Tag Body

- Simplest tags
  - `<prefix:tagName />`
- Tags with attributes
  - `<prefix:tagName att1="val1" ... />`
- Now

```
<prefix:tagName>
JSP Content
</prefix:tagName>

<prefix:tagName att1="val1" ... >
JSP Content
</prefix:tagName>
```

COMPSCI334

193

## Using Tag Body: The Tag Handler Class

- `doTag`
  - Output the opening html font tag
    - `<font color="blue">`
  - Output the word “Welcome” according to the attribute “repeat”
  - Include the tag body by
    - `getJspBody().invoke(null);`
  - Output the closing html font tag: `</font>`
- The two methods for handling the attributes are the same as before.

COMPSCI334

194

```
public void doTag() throws JspException, IOException {
 JspWriter out = getJspContext().getOut();
 out.print("");
 for (int i = 0; i < repeat; i++)
 out.print("Welcome ");
 getJspBody().invoke(null);
 out.print("");
}
```

```
Welcome
xxx

```

COMPSCI334

195

## Using Tag Body: The Tag Library Descriptor File

- Only difference is bodycontent element
    - Should be **scriptless** instead of empty:
- ```
<tag>
  <name>...</name>
  <tagclass>...</tagclass>
  <bodycontent>scriptless</bodycontent>
  <info>...</info>
</tag>
```

COMPSCI334

196

```

<tag>
  <name>welcomeTagB</name>
  <tagclass>examples.tags.WelcomeTagB</tagclass>
  <bodycontent>scriptless</bodycontent>
  <info>Outputs a welcome message.</info>
  <attribute>
    <name>colour</name>
    <required>false</required>
  </attribute>
  <attribute>
    <name>repeat</name>
    <required>false</required>
  </attribute>
</tag>

```

COMPSCI334

197

Using welcomeTagB Tag

```

<HTML>

<BODY>
<H1>Simple Tag Example</H1>

<%@ taglib uri="334-taglib.tld" prefix="334" %>

<334:welcomeTagB colour="red">
everybody
</334:welcomeTagB>
</BODY>
</HTML>

```

COMPSCI334

198

Address  http://localhost:8080/334/examples-doc/welcomeTagB.jsp

Simple Tag Example

Welcome everybody

COMPSCI334

199

Manipulating Tag Body

- Modifying tag body


```

<334:filterTag>
  if (a<b)
    doThis();
  else
    doThat();
</334:filterTag>

```
- Use a StringWriter object to hold the tag body
 - `StringWriter stringWriter = new StringWriter();`
 - `getJspBody().invoke(stringWriter);`
- Manipulate the tag body
 - `String modifiedBody = modifyString(stringWriter.toString());`
- Output the modified body using JspWriter object
 - `JspWriter out = getJspContext().getOut();`
 - `out.print(output);`

COMPSCI334

200

```

public void doTag() throws JspException, IOException {
    StringWriter stringWriter = new StringWriter();
    getJspBody().invoke(stringWriter);
    String output = filter(stringWriter.toString());
    JspWriter out = getJspContext().getOut();
    out.print(output);
}

```

COMPSCI334

201

```

<tag>
  <name>filterTag</name>
  <tagclass>examples.tags.FilterTag</tagclass>
  <bodycontent>scriptless</bodycontent>
  <info>Replaces HTML-specific characters in body.</info>
</tag>

```

COMPSCI334

202

```

<HTML>
<BODY>

<%@ taglib uri="334-taglib.tld" prefix="334" %>

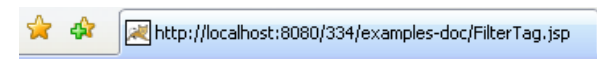
<334:filterTag>
if (a<b)
    doThis();
else
    doThat();
</334:filterTag>

</BODY>
</HTML>

```

COMPSCI334

203



```

if (a<b) doThis(); else doThat();

```

COMPSCI334

204

Obtaining request time information

- This example defines a tag for displaying the values of the request headers
- The request time information are stored in HttpServletRequest object.
- To obtain the HttpServletRequest object
`HttpServletRequest request = (HttpServletRequest)pageContext.getRequest();`
- The method for retrieving the values of the request headers are the same as the servlet examples
 - `request.getHeaderNames(), request.getHeader(header)`

COMPSCI334

205

Address  http://localhost:8080/334/examples-doc/RequestInfo.jsp

 Go  Links

Simple Tag Example

User-Agent : Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705)
Accept : image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, /*/*
Host : localhost:8080
Accept-Encoding : gzip, deflate
Accept-Language : en-nz
Connection : Keep-Alive

COMPSCI334

206

```
public class RequestInfo extends SimpleTagSupport {  
    public void doTag() throws JspException, IOException {  
        PageContext pageContext =  
            (PageContext)getJspContext();  
        HttpServletRequest request =  
            (HttpServletRequest)pageContext.getRequest();  
        Enumeration headers = request.getHeaderNames();  
        JspWriter out = getJspContext().getOut();  
        while (headers.hasMoreElements()) {  
            String header = (String)headers.nextElement();  
            out.print(header + " : "  
                + request.getHeader(header) + "<BR>");  
        }  
    }  
}
```

COMPSCI334

207

```
<tag>  
    <name>RequestInfo</name>  
    <tagclass>examples.tags.RequestInfo</tagclass>  
    <bodycontent>Empty</bodycontent>  
    <info>Display request information.</info>  
</tag>
```

COMPSCI334

208

```

<HTML>
<BODY>
<H1>Simple Tag Example</H1>

<%@ taglib uri="334-taglib.tld" prefix="334" %>

<334:RequestInfo />
</BODY>
</HTML>

```

COMPSCI334

209

Looping Tag

- A tag that outputs its body more than once
 - the body has different values each time

```
<334:forEach items="${courses}" var="course">
<LI>${course}
</334:forEach>
```
- Attribute items
 - Attribute should be defined with rtexprvalue as true
 - items="\${courses}"
 - Usually supply value with the JSP EL
 - The value of the attribute value is a collection
 - items, e.g. {314, 334, 335}
 - An array of Object type variable to hold the attribute value

COMPSCI334

210

- Attribute var
 - Define the name of the bean holding the value to be displayed

Some COMPSCI courses

- 313
- 314
- 334
- 335
- 340

COMPSCI334

211

```

private Object[] items;
private String attributeName;
public void setItems(Object[] items)
{
    this.items = items;
}

public void setVar(String attributeName)
{
    this.attributeName = attributeName;
}

```

COMPSCI334

212

- Body should have access to each item in collection
 - Retrieve each item in the items array
 - assign “course” as the name of the retrieved item and store it on the server
 - `getJspContext().setAttribute(key, object)`

COMPSCI334

213

```
public void doTag() throws JspException, IOException
{
    for (int i = 0; i < items.length; i++)
    {
        getJspContext().setAttribute(attributeName, items[i]);
        getJspBody().invoke(null);
    }
}
```

COMPSCI334

214

forEach: The Tag Library Descriptor File

```
<tag>
  <name>forEach</name>
  <tag-class>examples.tags.ForEachTag</tag-class>
  <body-content>scriptless</body-content>
  <attribute>
    <name>items</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>var</name>
    <required>true</required>
  </attribute>
</tag>
```

COMPSCI334

215

Servlet for testing forEach Tag

```
public class ForTagTest extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        String[] courses = {"313", "314", "334", "335", "340"};
        request.setAttribute("courses", courses);
        String address = "/examples-doc/ForTag.jsp";
        RequestDispatcher dispatcher =
            request.getRequestDispatcher(address);
        dispatcher.forward(request, response);
    }
}
```

COMPSCI334

216

Another example of using forEach Tag

Assignment Marks

Name	Asg1	Asg2
abc	100	90
def	80	70

COMPSCI334

217

```
<H2>Assignment Marks</H2>
<TABLE BORDER=1>
<334:forEach items="${records}" var="row">
  <TR>
    <334:forEach items="${row}" var="col">
      <TD>${col}</TD>
    </334:forEach>
  </TR>
</334:forEach>
</TABLE>
```

COMPSCI334

218

- The outer tag displays row
- The inner tag displays columns in each row
- records is a two dimensional array
 - { { "Name", "Asg1", "Asg2" }, { "abc", "100", "90" }, { "def", "80", "70" } }
- row is a one dimensional array
 - { "Name", "Asg1", "Asg2" }
- col is a simple value
 - "Name",

COMPSCI334

219

Another servlet for testing forEach Tag

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    String[][] records = { { "Name", "Asg1", "Asg2" },
                          { "abc", "100", "90" }, { "def", "80", "70" } };
    request.setAttribute("records", records);
    String address = "/examples-doc/ForTagB.jsp";
    RequestDispatcher dispatcher =
        request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
}
```

COMPSCI334

220