Vietnam National University of HCMC
International University
School of Computer Science and Engineering

Web Application Development (IT093IU)

Assoc. Prof. Nguyen Van Sinh
Email: nvsinh@hcmiu.edu.vn

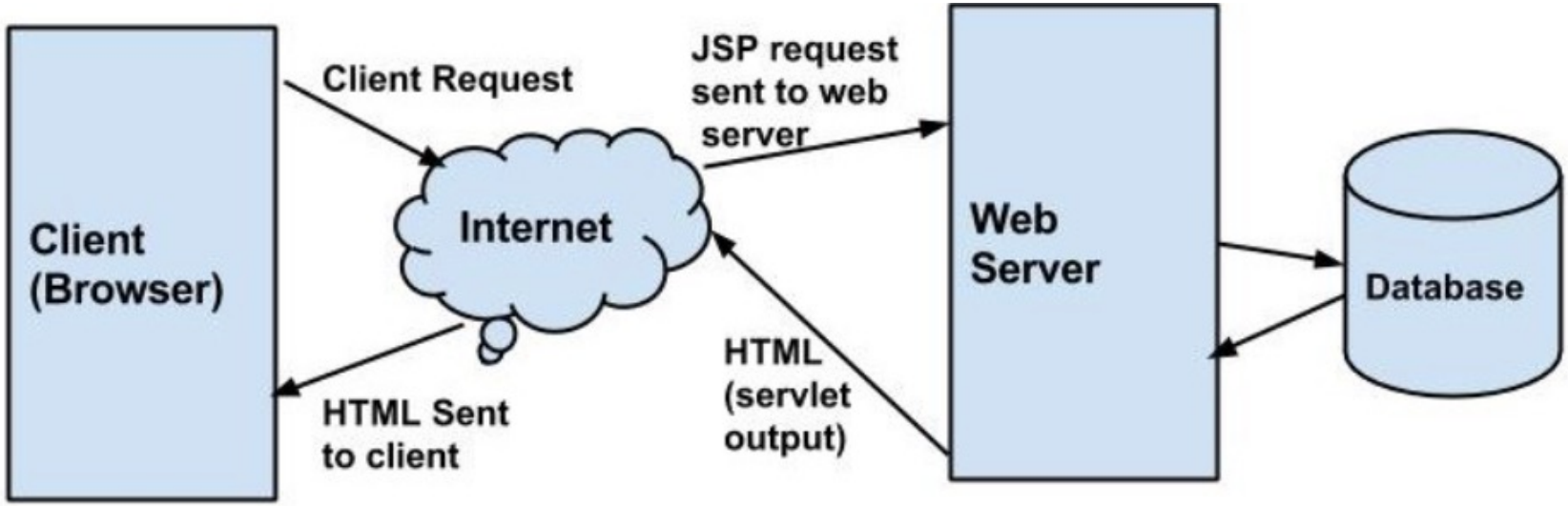
(Semester 2, 2023-2024)

Lecture 5: Java Server Page (JSP)

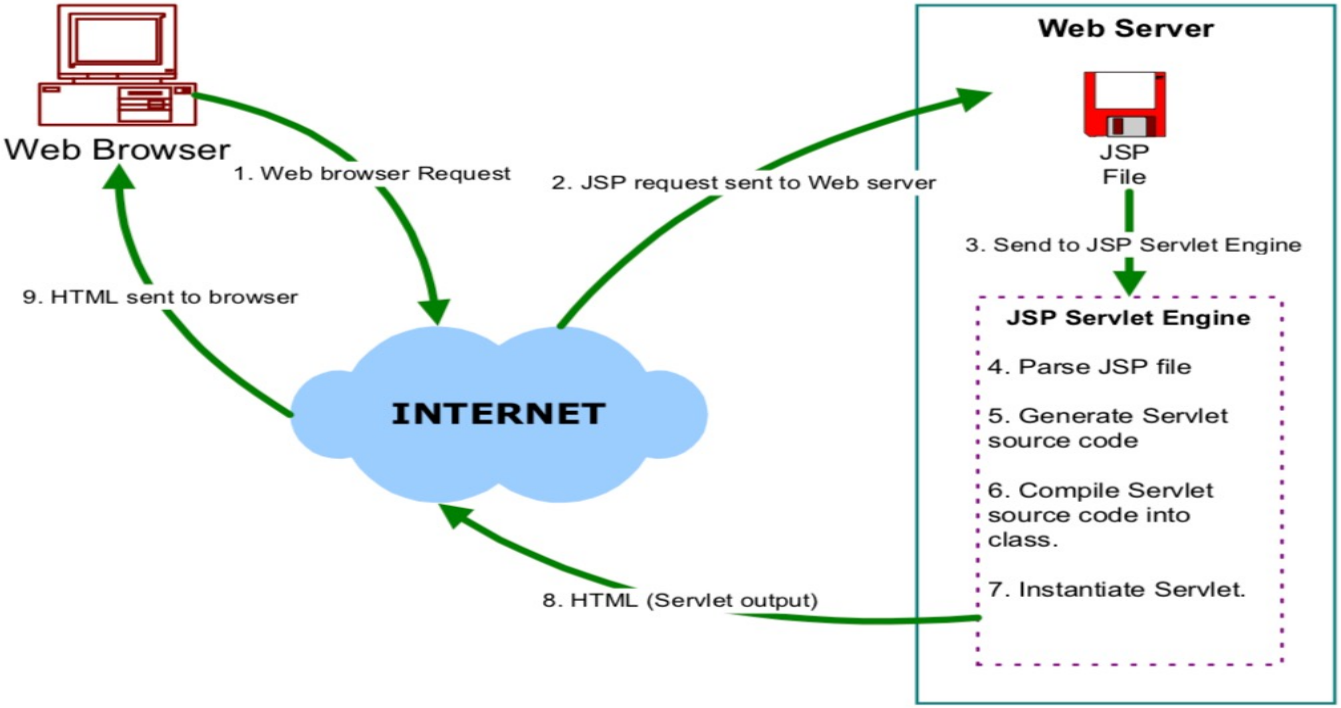
- Introduction to JSP
- Why we need JSP
- How JSP works
- Benefits of JSP
- Setting up your environment for JSP
- A simple example

Introduction to JSP

- JSP Stands for Java Server Pages
- Presents dynamic content to users
- Handles the presentation logic in an MVC architecture



JSP Architecture



Steps required for a JSP request:

1. The user goes to a web site made using JSP. The user goes to a JSP page (ending with .jsp). The web browser makes the request via the Internet.
2. The JSP request gets sent to the Web server.
3. The Web server recognises that the file required is special (.jsp), therefore passes the JSP file to the JSP Servlet Engine.
4. If the JSP file has been called the first time, the JSP file is parsed, otherwise go to step 7.
5. The next step is to generate a special Servlet from the JSP file. All the HTML required is converted to println statements.
6. The Servlet source code is compiled into a class.
7. The Servlet is instantiated, calling the *init* and *service* methods.
8. HTML from the Servlet output is sent via the Internet.
9. HTML results are displayed on the user's web browser.

The Need for JSP

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

The JSP Framework

- Idea:
  - Use regular HTML for most of page
  - Mark servlet code with special tags
  - Entire JSP page gets translated into a servlet (once), and servlet is what actually gets invoked (for each request)
- Example:
  - JSP
    - Thanks for ordering <I><%= request.getParameter("title") %></I>
    - <% out.println(2*5) %>
    - <% out.println(“Result: ” + 2*5) %> //+ is a connection
  - URL
    - http://host/OrderConfirmation.jsp ?title=Core+Web+Programming
  - Result
    - Thanks for ordering *Core Web Programming*

Benefits of JSP

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

Advantages of JSP over Competing Technologies

- Versus ASP or ColdFusion
  - Better language for dynamic part
  - Portable to multiple servers and operating systems
- Versus PHP
  - Better language for dynamic part
  - Better tool support
- Versus pure servlets
  - More convenient to create HTML
  - Can use standard tools (e.g., HomeSite)
  - Divide and conquer
  - JSP programmers still need to know servlet programming

Setting Up Your Environment

- Set your CLASSPATH.
- Compile your code.
- Use packages to avoid name conflicts.
- Put JSP page in special directory.
- Use special URL to invoke JSP page.
- … all of them are existed in the Netbean IDE

Example: Index.html

```
<body>
   <h1>Test JSP Page</h1>
   <hr>
   <form method="post" action="TestJSP.jsp" >
      User name: <input type="text" name="User">
      <br>
      Password:  <input type="pass" name="Pass">
      <br>
                 <input type="submit">
   </form>
</body>
```

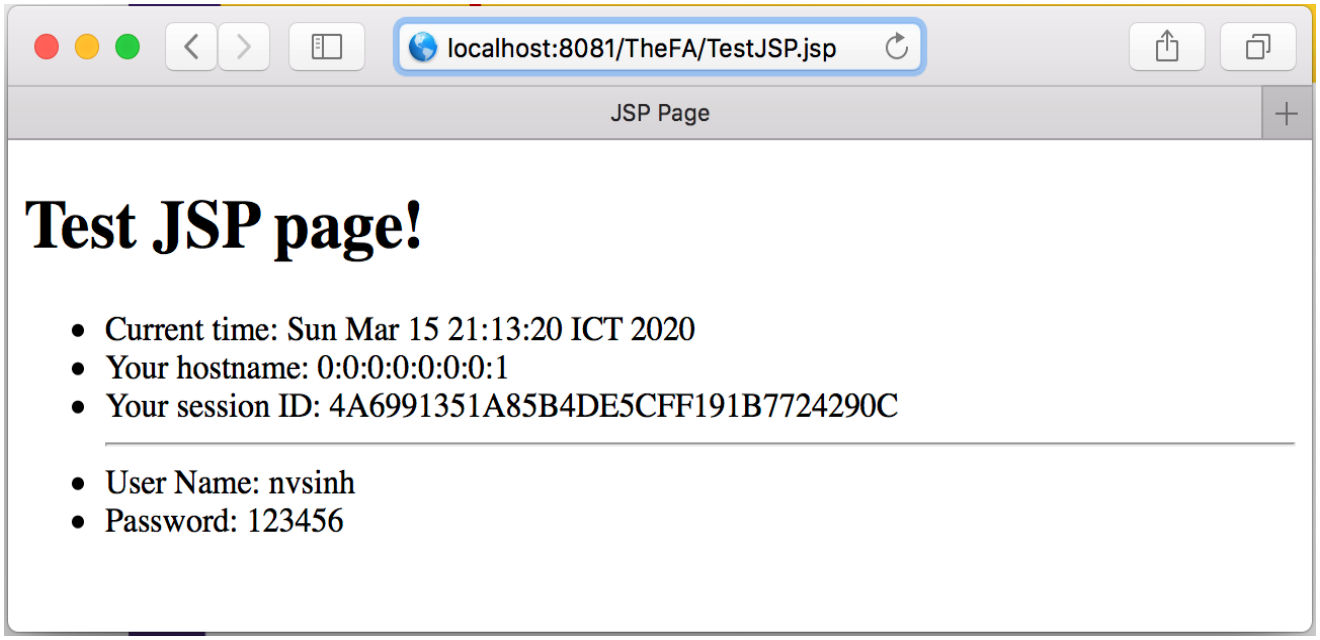
Example: TestJSP.jsp

```
<body>
  <h1>Test JSP page!</h1>
  <UL>
      <LI>Current time: <%= new java.util.Date()%>
      <LI>Your hostname: <%=
  request.getRemoteHost()%>
      <LI>Your session ID: <%= session.getId()%>
      <hr>
      <LI>User Name: <%=
  request.getParameter("User")%>
      <LI>Password: <%=
  request.getParameter("Pass")%>
  </UL>
</body>
```

Example Result

- If location was
  - ../TheFA/TestJSP.jsp
- URL would be

  http://localhost:8081/TheFA/TestJSP.jsp



Login page (html code)

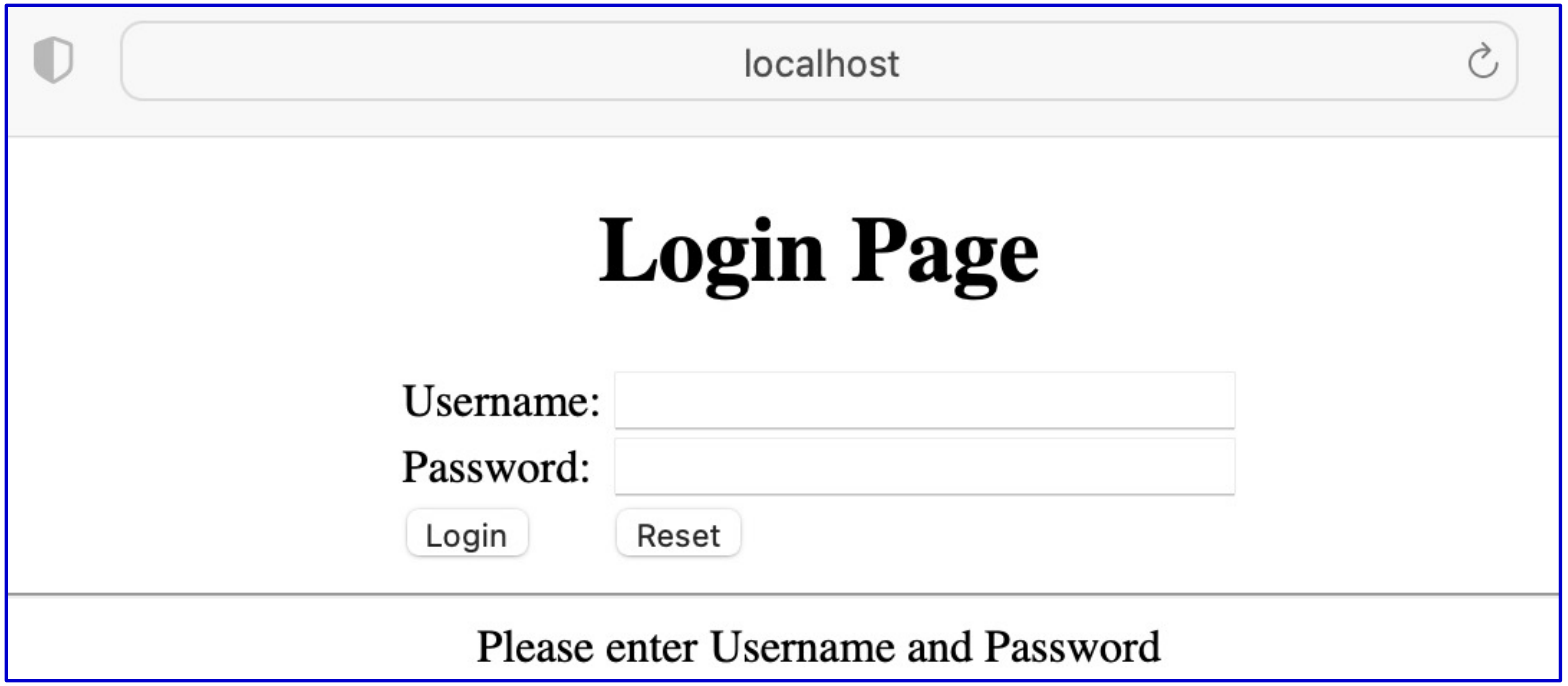
```
<h1>Login Page</h1>
<form action="NewForm.jsp" method="post">
    <table border="0">
        <tr>
            <td>Username: </td>
            <td><input type="text" name="User" size="30"> </td>
        </tr>
        <tr>
            <td>Password: </td>
            <td><input type="text" name="Pass" size="30"></td>
        </tr>
        <tr>
            <td><input type="submit" name= "Login" value="Login"></td>
            <td><input type="reset" size="30"></td>
        </tr>
    </table>
</form>
```

Login page (interface)

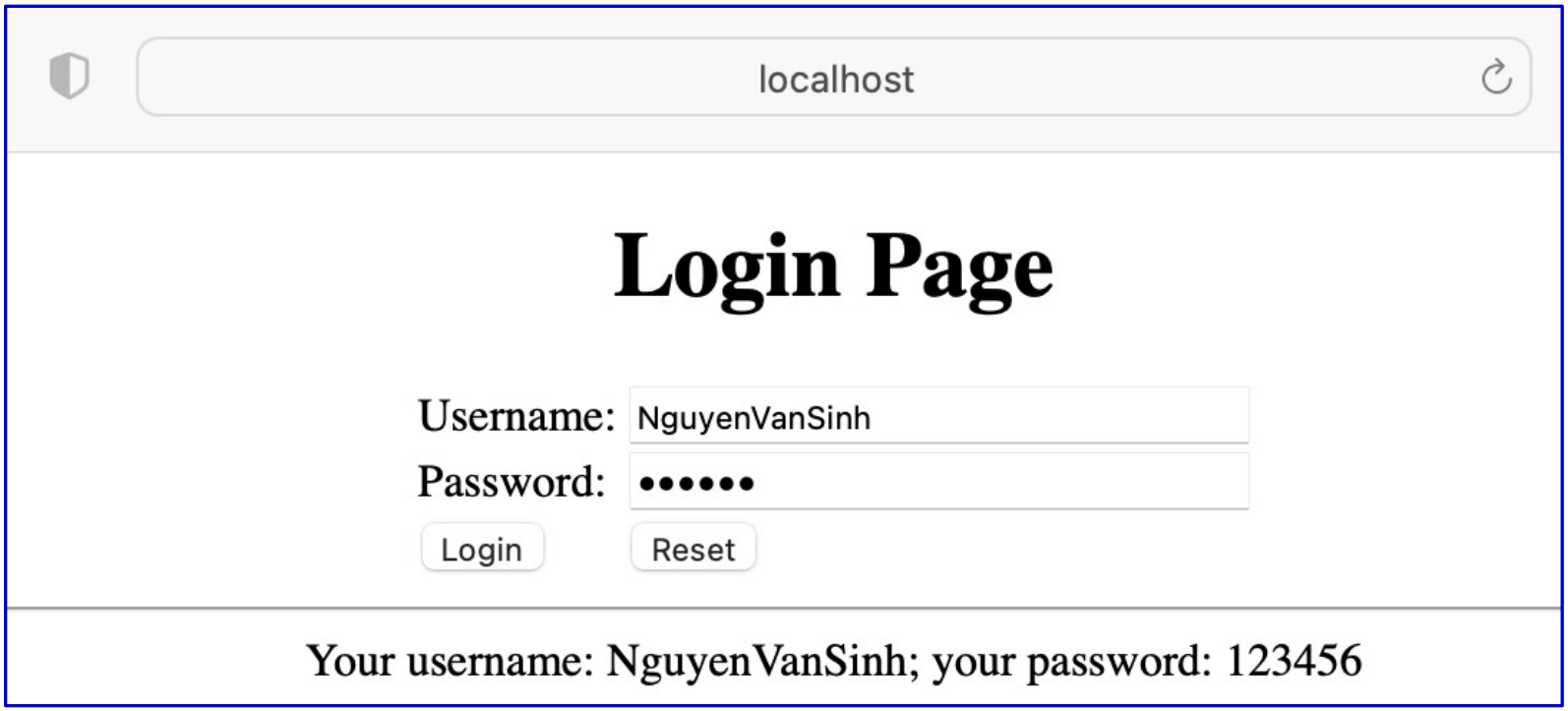
Login page (jsp code)

```
<%
    if (request.getParameter("Login") != null) {
        String name = request.getParameter("User");
        String password = request.getParameter("Pass");
        if (name.equals("") && password.equals("")) {
            out.print("Please enter Username and Password");
        } else {
            out.print("Your username: " + name + "; your password: " + password);
            }
    }
%>
```

Login page (result)



Login page (result)



Most common misunderstanding forgetting JSP is server-side technology

- Very common question
  - I can't do such and such with HTML.
    Will JSP let me do it?
- Why doesn't this question make sense?
  - JSP runs entirely on server
  - It doesn't change content the client (browser) can handle
- Similar questions
  - How do I put a normal applet in a JSP page?
    Answer: send an <APPLET…> tag to the client
  - How do I put an image in a JSP page?
    Answer: send an <IMG …> tag to the client
  - How do I use JavaScript/Acrobat/etc., ?
    Answer: send the appropriate HTML tags

2nd most common misunderstanding translation/request time confusion

- What happens at page translation time?
  - JSP constructs get translated into servlet code.
- What happens at request time?
  - Servlet code gets executed. *No* interpretation of JSP occurs at request time. The original JSP page is totally ignored at request time; only the servlet that resulted from it is used.
- When does page translation occur?
  - Typically, the first time JSP page is accessed after it is modified. This should never happen to real user (developers should test all JSP pages they install).
  - Page translation does *not* occur for each request.

The JSP Lifecycle

- JSP Page Translation: The container validates the JSP page and parse it to generate the servlet file.
- JSP Page Compilation: The generated servlet file is compiled into a servlet class.
- Class Loading: Container loads the generated servlet class into memory.
- Instantiation: Container invokes the no argument constructor of generated servlet class to load it into memory and instantiate it.
- Initialization: After creating the instance, jspInit() method is called to initialize it.
- Request Processing: The _jspService() is called by web container every time when a request is come for the jsp.
  Syntax: public void _jspService() throws ServletException,IOException
- **Note:** In this method underscore (_) represents that implementation of this method is provided by auto generated servlet and it can't be overridden by developer.
- Destroy: The jspDestroy() method is called by web container to unload JSP from memory.

JSP/Servlets in the Real World

- First USA Bank: largest credit card issuer in the world; most on-line banking customers

JSP/Servlets in the Real World

- Delta Airlines: entire Web site, including real-time schedule info

JSP/Servlets in the Real World

- American Century Investments: more than 70 mutual funds, $90 billion under management, two million investors

Summary

- JSP makes it easier to create and maintain HTML, while still providing full access to servlet code
- JSP pages get translated into servlets
  - It is the servlets that run at request time
  - Client does not see anything JSP-related
- You still need to understand servlets
  - Understanding how JSP really works
  - Servlet code called from JSP
  - Knowing when servlets are better than JSP
  - Mixing servlets and JSP
- Other technologies use similar approach, but aren't as portable and don't let you use Java for the "real code"

Agenda - scripting elements

- Basic syntax
- Types of JSP scripting elements
- Expressions
- Predefined variables
- Scriptlets
- Declarations

Uses of JSP constructs

Simple Application

Complex Application

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Custom tags
- Servlet/JSP combo (MVC), with beans and possibly custom tags

Design strategy: Limit java code in JSP pages

- You have two options
  - Put 25 lines of Java code directly in the JSP page
  - Put those 25 lines in a separate Java class and put 1 line in the JSP page that invokes it
- Why is the second option *much* better?
  - **Development**. You write the separate class in a Java environment (editor or IDE), not an HTML environment
  - **Debugging**. If you have syntax errors, you see them immediately at compile time. Simple print statements can be seen.
  - **Testing**. You can write a test routine with a loop that does 10,000 tests and reapply it after each change.
  - **Reuse**. You can use the same class from multiple pages.

Basic syntax

- HTML Text
  - <H1>Blah</H1>
  - Passed through to client. Really turned into servlet code that looks like
    - out.print("<H1>Blah</H1>");
- HTML Comments
  - <!-- Comment -->
  - Same as other HTML: passed through to client
- JSP Comments
  - <%-- Comment --%>
  - Not sent to client

Types of scripting elements

- Expressions
  - Format: <%= expression %>
  - Evaluated and inserted into the servlet's output. I.e., results in something like out.print(expression)
- Scriptlets
  - Format: <% code %>
  - Inserted verbatim into the servlet's _jspService method (called by service)
- Declarations
  - Format: <%! code %>
  - Inserted verbatim into the body of the servlet class, outside of any existing methods

JSP expressions

- Format
  - <%= Java Expression %>
- Result
  - Expression evaluated, converted to String, and placed into HTML page at the place it occurred in JSP page
  - That is, expression placed in _jspService inside out.print
- Examples
  - Current time: <%= new java.util.Date() %>
  - Your hostname: <%= request.getRemoteHost() %>
- XML-compatible syntax
  - <jsp:expression>Java Expression</jsp:expression>
  - XML version not supported by Tomcat 3. Until JSP 1.2, servers are not required to support it. Even then, you cannot mix versions within a single page.

JSP/Servlet Correspondence

- Original JSP

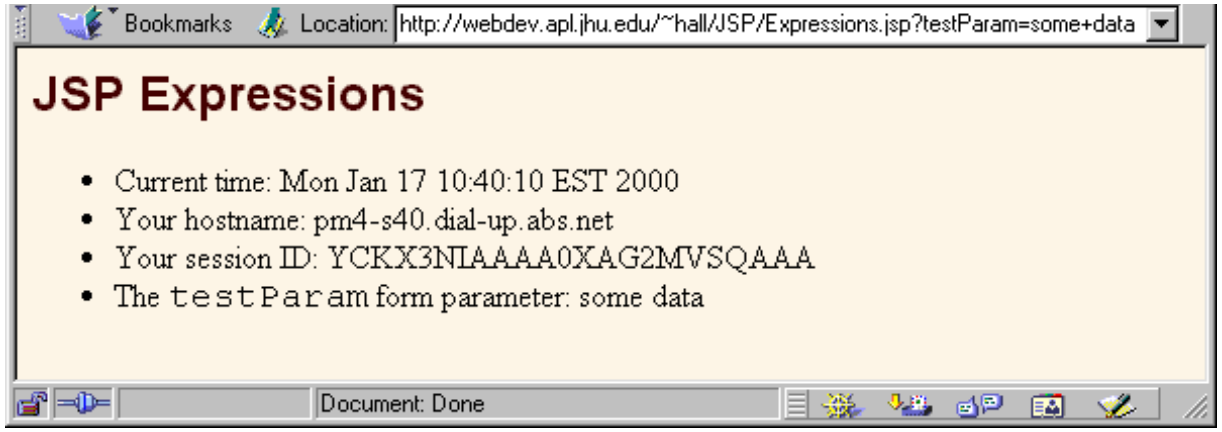
```
<H1>A Random Number</H1>
<%= Math.random() %>
```

- 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());
 ...
}
```

Example using JSP expressions

```
<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Your hostname: <%= request.getRemoteHost() %>
  <LI>Your session ID: <%= session.getId() %>
  <LI>The <CODE>testParam</CODE> form parameter:
      <%= request.getParameter("testParam") %>
</UL>
</BODY>
```



Predefined variables

- request
  - The HttpServletRequest (1st argument to service/doGet)
- response
  - The HttpServletResponse (2nd arg to service/doGet)
- out
  - The Writer (a buffered version of type JspWriter) used to send output to the client
- session
  - The HttpSession associated with the request (unless disabled with the session attribute of the page directive)
- application
  - The ServletContext (for sharing data) as obtained via getServletContext().

JSP scriptlets

- Format
  - <% Java Code %>
- Result
  - Code is inserted verbatim into servlet's _jspService
- Example
  - <%
    String queryData = request.getQueryString();
    out.println("Attached GET data: " + queryData);
    %>
  - <% response.setContentType("text/plain"); %>
- XML-compatible syntax
  - <jsp:scriptlet>Java Code</jsp:scriptlet>

JSP/Servlet correspondence

- Original JSP

```
<%= foo() %>
<% bar(); %>
```

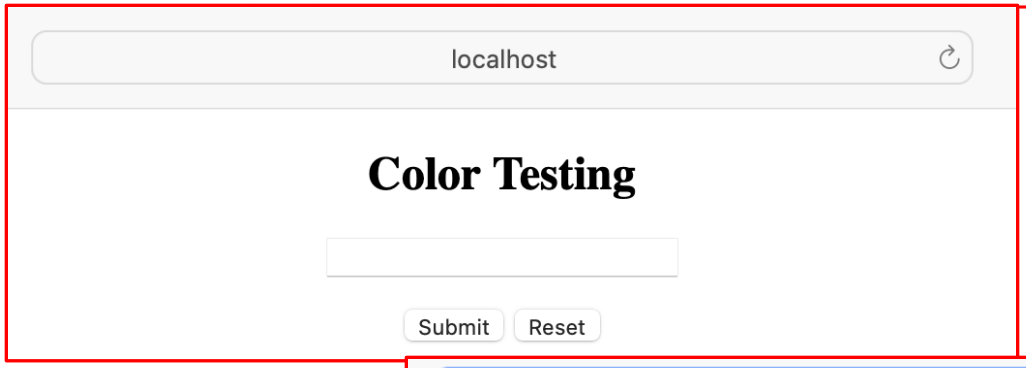
- Possible resulting servlet code

```
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(foo());
 bar();
 ...
}
```

Example using JSP scriptlets

```
<HEAD>
    <TITLE>Color Testing</TITLE>
</HEAD>
<%
    String bgColor = "BLUE";
    String myColor = request.getParameter("color");

    if (myColor != null) {
        myColor = myColor;
    } else {
        myColor = bgColor;
    }
%>
<BODY BGCOLOR="<%= myColor%>">
    <H2 ALIGN="CENTER">Color Testing</H2>
    <center>
    <form>
        <input type="text" name="color" size="25">
        <p></p>
        <input type="submit" value="Submit">
        <input type="reset" value="Reset">
    </form>
    </center>
</BODY>
```

JSP Scriptlets: results



Using scriptlets to make parts of the JSP file conditional

- Point
  - Scriplets are inserted into servlet exactly as written
  - Need not be complete Java expressions
  - Complete expressions are usually clearer and easier to maintain, however
- Example
  - ```
    <% if (Math.random() < 0.5) { %>
    Have a <B>nice</B> day!
    <% } else { %>
    Have a <B>busy</B> day!
    <% } %>
    ```
- Representative result
  - ```
    if (Math.random() < 0.5) {
      out.println("Have a <B>nice</B> day!");
    } else {
      out.println("Have a <B>busy</B> day!");
    }
    ```

JSP declarations

- Format
  - <%! Java Code %>
- Result
  - Code is inserted verbatim into servlet's class definition, outside of any existing methods
- Examples
  - <%! private int someField = 5; %>
  - <%! private void someMethod(...) {...} %>
- Design consideration
  - Fields are clearly useful. For methods, it is usually better to define the method in a separate Java class.
- XML-compatible syntax
  - <jsp:declaration>Java Code</jsp:declaration>

JSP/Servlet correspondence

- Original JSP

```
<H1>Some Heading</H1>
<%!
  private String randomHeading() {
    return("<H2>" + Math.random() + "</H2>");
  }
%>
<%= randomHeading() %>
```

- (Alternative: make randomHeading a static method in a separate Java class)

JSP/Servlet correspondence

- Possible resulting servlet code

```
public class xxxx implements HttpJspPage {
 private String randomHeading() {
   return("<H2>" + Math.random() + "</H2>");
 }

 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>Some Heading</H1>");
   out.println(randomHeading());
   ...
 } ...
}
```

Example using JSP to count visitors

```
<BODY bgcolor="pink">
    <center>
        <H1>Visiting Counter</H1> <hr>
        <%! private int accessCount = 0;%>
        <H2>Accesses to page since server reboot:
            <%= ++accessCount%></H2>
    </center>
</BODY>
```

JSP declarations: the jspInit and jspDestroy methods

- JSP pages, like regular servlets, sometimes want to use init and destroy
- Problem: the servlet that gets built from the JSP page might already use init and destroy
  - Overriding them would cause problems.
  - Thus, it is illegal to use JSP declarations to declare init or destroy.
- Solution: use jspInit and jspDestroy.
  - The auto-generated servlet is guaranteed to call these methods from init and destroy, but the standard versions of jspInit and jspDestroy are empty (placeholders for you to override).

JSP declarations and predefined variables

- Problem
  - The predefined variables (request, response, out, session, etc.) are *local* to the _jspService method. Thus, they are not available to methods defined by JSP declarations or to methods in helper classes. What can you do about this?
- Solution: pass them as arguments. E.g.

```
<%!
private void someMethod(HttpSession s)
{
  doSomethingWith(s);
}
%>
<% someMethod(session); %>
```

- Note that the println method of JspWriter throws IOException
  - Use “throws IOException” for methods that use println

Using JSP expressions as attribute values

- Static Value
  - ```
    <jsp:setProperty
      name="author"
      property="firstName"
      value="Marty" />
    ```
- Dynamic Value
  - ```
    <jsp:setProperty
      name="user"
      property="id"
      value='<%= "UserID" + Math.random()
    %>' />
    ```

Attributes that permit JSP expressions

- The name and value properties of jsp:setProperty
  - See upcoming section on beans
- The page attribute of jsp:include
  - See upcoming section on including files and applets
- The page attribute of jsp:forward
  - See upcoming section on integrating servlets and JSP
- The value attribute of jsp:param
  - See upcoming section on including files and applets

Summary

- JSP Expressions
  - Format: <%= expression %>
  - Wrapped in out.print and inserted into _jspService
- JSP Scriptlets
  - Format: <% code %>
  - Inserted verbatim into the servlet's _jspService method
- JSP Declarations
  - Format: <%! code %>
  - Inserted verbatim into the body of the servlet class
- Predefined variables
  - request, response, out, session, application
- Limit the Java code that is directly in page
  - Use helper classes, beans, custom tags, servlet/JSP combo

Agenda - controlling the structure of generated servlets

- The import attribute
- The contentType attribute
- Generating plain text and Excel documents
- The isThreadSafe attribute
- The session attribute
- The buffer attribute
- The extends attribute
- The errorPage attribute
- The isErrorPage attribute

Purpose of the page directive

- Give high-level information about the servlet that will result from the JSP page
- Can control
  - Which classes are imported
  - What class the servlet extends
  - What MIME type is generated
  - How multithreading is handled
  - If the servlet participates in sessions
  - The size and behavior of the output buffer
  - What page handles unexpected errors

The import attribute

- Format
  - <%@ page import="package.class" %>
  - <%@ page import="package.class1,...,package.classN" %>
- Purpose
  - Generate import statements at top of servlet definition
- Notes
  - Although JSP pages can be almost anywhere on server, classes used by JSP pages must be in normal servlet dirs
  - Always use packages for utilities that will be used by JSP!

Example of import attribute

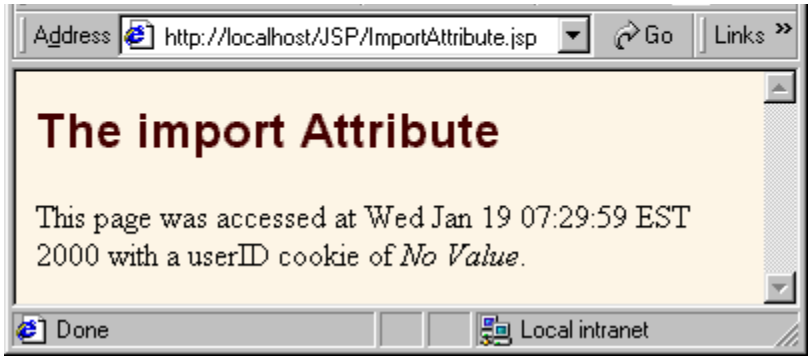
```
<BODY>
<H2>The import Attribute</H2>
<%-- JSP page directive --%>
<%@ page import="java.util.*,coreservlets.*" %>

<%-- JSP Declaration --%>
<%!
private String randomID() {
  int num = (int)(Math.random()*10000000.0);
  return("id" + num);
}
private final String NO_VALUE = "<I>No Value</I>";
%>
```

Example of import attribute (cont)

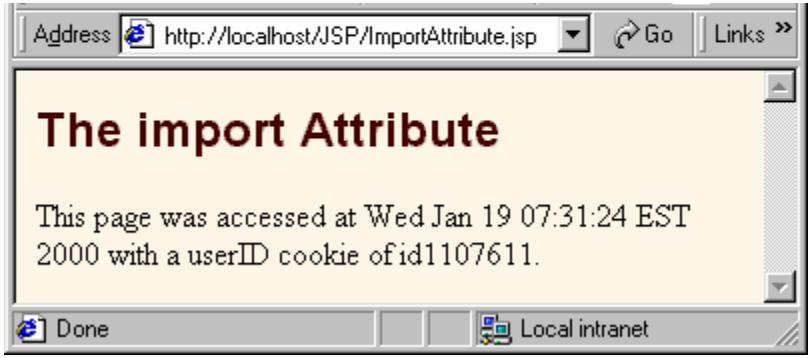
```
<%
Cookie[] cookies = request.getCookies();
String oldID =
  ServletUtilities.getCookieValue(cookies, "userID", NO_VALUE);
String newID;
if (oldID.equals(NO_VALUE)) {
  newID = randomID();
} else {
  newID = oldID;
}
LongLivedCookie cookie = new LongLivedCookie("userID", newID);
response.addCookie(cookie);
%>
<%-- JSP Expressions --%>
This page was accessed at <%= new Date() %> with a userID
cookie of <%= oldID %>.
 </BODY></HTML>
```

Example of import attribute: result

- First access



- Subsequent accesses



The contentType attribute

- Format
  - <%@ page contentType="MIME-Type" %>
  - <%@ page contentType="MIME-Type; charset=Character-Set" %>
- Purpose
  - Specify the MIME type of the page generated by the servlet that results from the JSP page
- Notes
  - Attribute value cannot be computed at request time
  - See section on response headers for table of the most common MIME types

Using contentType to generate plain text documents

```
<HTML>
<HEAD>
<TITLE>The contentType Attribute</TITLE>
</HEAD>
<BODY>
<H2>The contentType Attribute</H2>
<%@ page contentType="text/plain" %>
This should be rendered as plain text,
<B>not</B> as HTML.
</BODY>
</HTML>
```

Plain text documents in Netscape

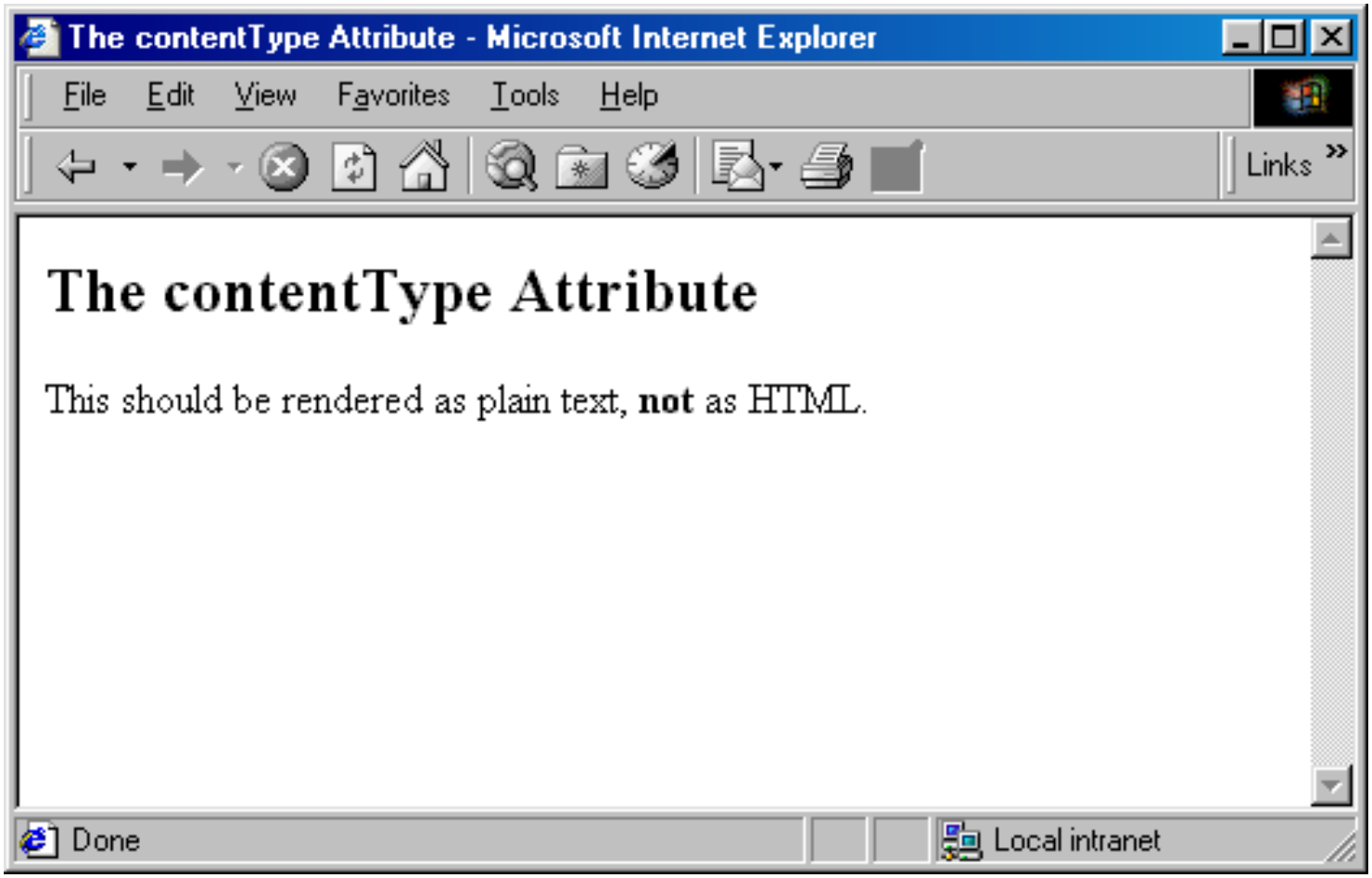
Netscape

File Edit View Go Communicator Help

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>The contentType Attribute</TITLE>
</HEAD>

<BODY>
<H2>The contentType Attribute</H2>

This should be rendered as plain text,
<B>not</B> as HTML.

</BODY>
</HTML>
```

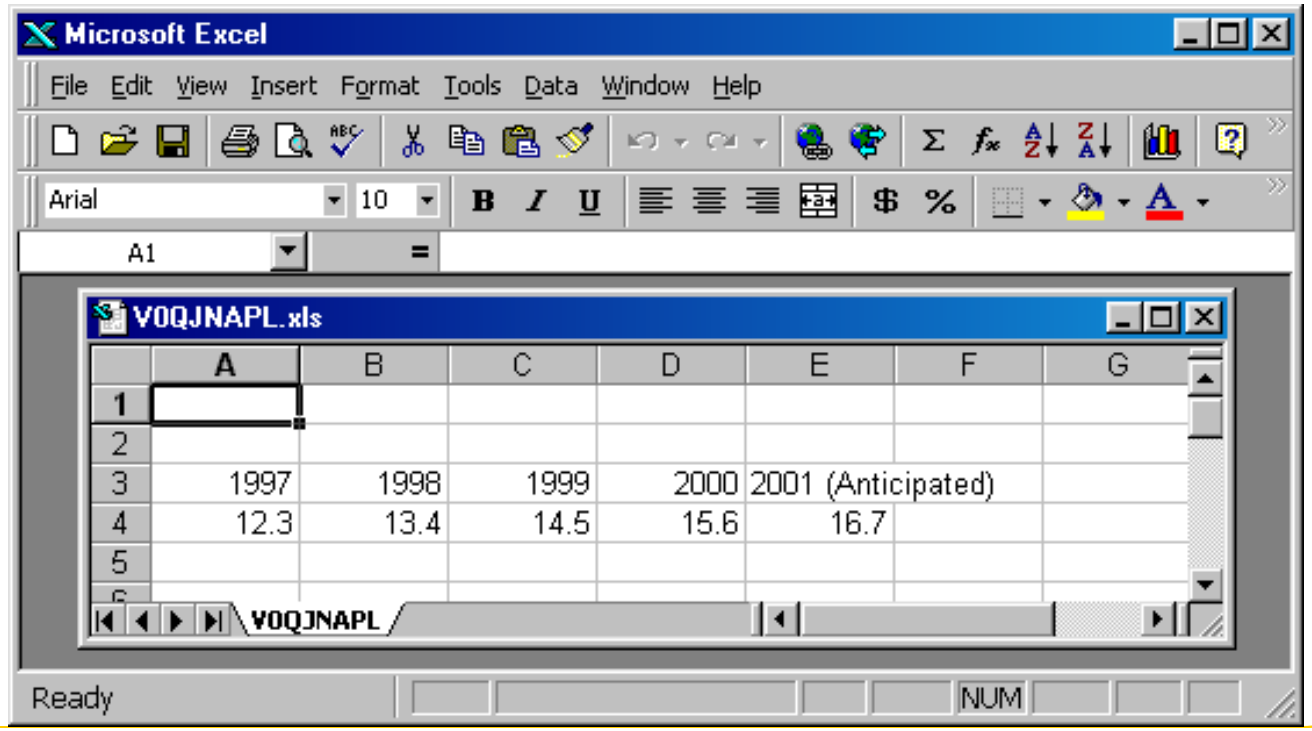
Document: Done

Plain text documents in IE



Generating excel spreadsheets

```
<%@ page contentType="application/vnd.ms-excel" %>
<%-- Note that there are tabs,
     not spaces, between columns. --%>
1997    1998    1999    2000    2001 (Anticipated)
12.3    13.4    14.5    15.6    16.7
```



Generating excel Sspreadsheets conditionally

- Excel can interpret HTML tables
  - Change MIME type based on request parameters
- You cannot use page directive
  - It does not use request-time values.
- Solution
  - Use predefined request variable and call setContentType

```
<%
if (someCondition) {
  response.setContentType("type1");
} else {
  response.setContentType("type2");
}
%>
```

Generating excel spreadsheets conditionally

```
<%
String format = request.getParameter("format");
if ((format != null) && (format.equals("excel"))) {
  response.setContentType("application/vnd.ms-
   excel");}
%>
<HTML><HEAD>
<TITLE>Comparing Apples and Oranges</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<CENTER>
<H2>Comparing Apples and Oranges</H2>
```

Generating excel spreadsheets conditionally (continued)

```
<TABLE BORDER=1>
  <TR><TH></TH><TH>Apples<TH>Oranges
  <TR><TH>First Quarter<TD>2307<TD>4706
  <TR><TH>Second Quarter<TD>2982<TD>5104
  <TR><TH>Third Quarter<TD>3011<TD>5220
  <TR><TH>Fourth Quarter<TD>3055<TD>5287
</TABLE>

</CENTER>
</BODY>
</HTML>
```

Apples and oranges: default result

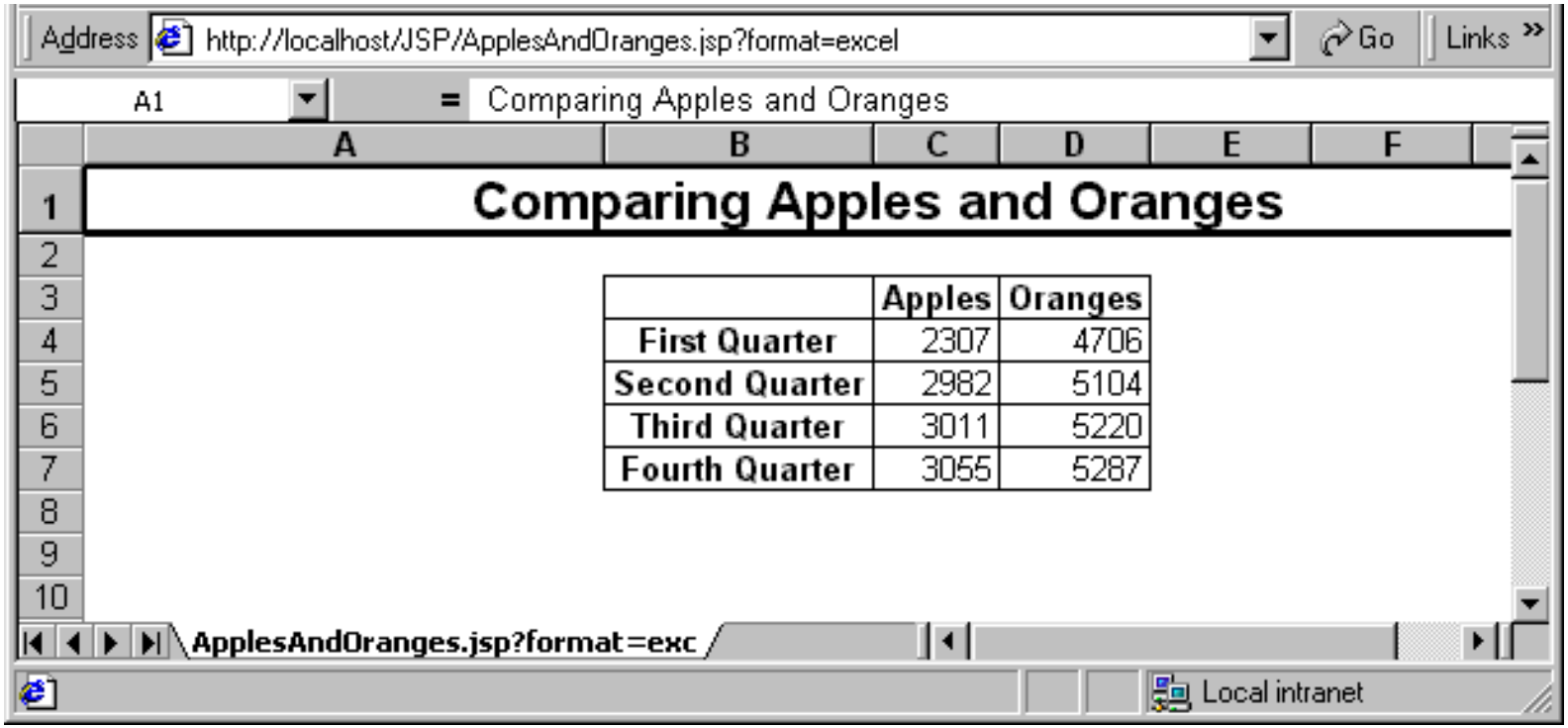
Address http://localhost/JSP/ApplesAndOranges.jsp Go Links »

Comparing Apples and Oranges

	Apples	Oranges
First Quarter	2307	4706
Second Quarter	2982	5104
Third Quarter	3011	5220
Fourth Quarter	3055	5287

Done Local intranet

Apples and oranges: result with format=excel



	Apples	Oranges
First Quarter	2307	4706
Second Quarter	2982	5104
Third Quarter	3011	5220
Fourth Quarter	3055	5287

The session attribute

- Format
  - <%@ page session="true" %> <%--Default-%>
  - <%@ page session="false" %>
- Purpose
  - To designate that page not be part of a session
- Notes
  - By default, it is part of a session
  - Saves memory on server if you have a high-traffic site
  - *All* related pages have to do this for it to be useful

The buffer attribute

- Format
  - <%@ page buffer="*size*kb" %>
  - <%@ page buffer="none" %>
- Purpose
  - To give the size of the buffer used by the out variable
- Notes
  - Buffering lets you set HTTP headers even after some page content has been generated (as long as buffer has not filled up or been explicitly flushed)
  - Servers are allowed to use a larger size than you ask for, but not a smaller size
  - Default is system-specific, but must be at least 8kb

The extends attribute

- Format
  - <%@ page extends="package.class" %>
- Purpose
  - To specify parent class of servlet that will result from JSP page
- Notes
  - Use with extreme caution
  - Can prevent system from using high-performance custom superclasses
  - Real purpose is to let you extend classes *that come from the server vendor* (e.g., to support personalization features), not to extend your own classes.

The errorPage attribute

- Format
  - <%@ page errorPage="Relative URL" %>
- Purpose
  - Specifies a JSP page that should process any exceptions thrown but not caught in the current page
- Notes
  - The exception thrown will be automatically available to the designated error page by means of the "exception" variable
  - The web.xml file lets you specify *application-wide* error pages that apply whenever certain exceptions or certain HTTP status codes result.
    - The errorPage attribute is for *page-specific* error pages

The isErrorPage attribute

- Format
  - <%@ page isErrorPage="true" %>
  - <%@ page isErrorPage="false" %> <%-- Default --%>
- Purpose
  - Indicates whether or not the current page can act as the error page for another JSP page
- Notes
  - Use this for emergency backup only; explicitly handle as many exceptions as possible
  - Don't forget to always check query data for missing or malformed values
  - The web.xml file can designate general error pages rather than page-specific ones like this

Error Pages: example (computeSpeed.jsp)

```
<BODY>
<%@ page errorPage="SpeedErrors.jsp" %>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Computing Speed
</TABLE>
<%!
// Note lack of try/catch for
  NumberFormatException
private double toDouble(String value) {
  return(Double.valueOf(value).doubleValue());
}
%>
```

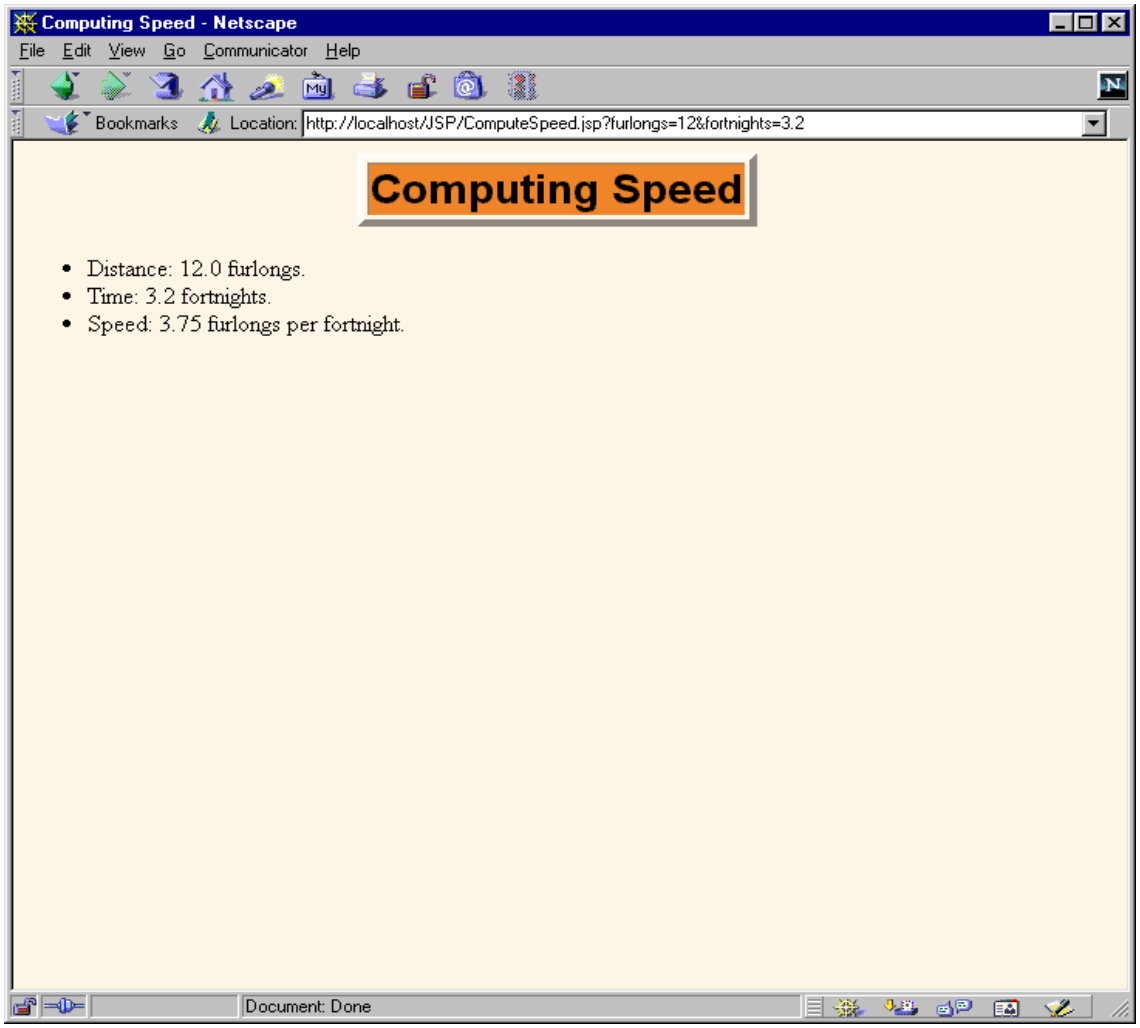
Error Pages: example (computeSpeed.jsp)

```
<%
double furlongs =
   toDouble(request.getParameter("furlongs"));
double fortnights =
   toDouble(request.getParameter("fortnights"));
double speed = furlongs/fortnights;
%>
<UL>
  <LI>Distance: <%= furlongs %> furlongs.
  <LI>Time: <%= fortnights %> fortnights.
  <LI>Speed: <%= speed %> furlongs per fortnight.
</UL>
...
```

Error Pages: example (SpeedErrors.jsp)

```
<BODY>

<%@ page isErrorPage="true" %>

<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Error Computing Speed</TABLE>
<P>
ComputeSpeed.jsp reported the following error:
<I><%= exception %></I>. This problem occurred in
   the
following place:
<PRE>
<% exception.printStackTrace(
                  new java.io.PrintWriter(out)); %>
</PRE>
...
```

Error pages: example



Error pages: example

Error Computing Speed - Netscape

File Edit View Go Communicator Help

Bookmarks Location: http://localhost/JSP/ComputeSpeed.jsp?furlongs=12&fortnights=three

Error Computing Speed

ComputeSpeed.jsp reported the following error: *java.lang.NumberFormatException: three*. This problem occurred in the following place:

```
java.lang.NumberFormatException: three
        at java.lang.Double.valueOf(Double.java:121)
        at pagecompile.jsp._JSP._ComputeSpeed.toDouble(_ComputeSp
        at pagecompile.jsp._JSP._ComputeSpeed._jspService(_Comput
        at com.sun.server.http.pagecompile.jsp.runtime.HttpJspBas
        at javax.servlet.http.HttpServlet.service(Compiled Code)
        at com.sun.server.http.pagecompile.jsp.runtime.JspServlet
        at com.sun.server.http.pagecompile.jsp.runtime.JspServlet
        at com.sun.server.http.pagecompile.jsp.runtime.JspServlet
        at javax.servlet.http.HttpServlet.service(Compiled Code)
        at com.sun.server.ServletState.callService(ServletState.j
        at com.sun.server.ServletManager.callServletService(Compi
        at com.sun.server.ProcessingState.invokeTargetServlet(Prc
        at com.sun.server.http.HttpProcessingState.execute(HttpPr
        at com.sun.server.http.stages.Runner.process(Runner.java:
        at com.sun.server.ProcessingSupport.process(Compiled Code
        at com.sun.server.Service.process(Service.java:204)
        at com.sun.server.http.HttpServiceHandler.handleRequest(H
        at com.sun.server.http.HttpServiceHandler.handleRequest(C
        at com.sun.server.HandlerThread.run(Compiled Code)
```

Document: Done

Summary

- The import attribute
  - Changes the packages imported by the servlet that results from the JSP page
    - *Always* use packages for utility classes!
- The contentType attribute
  - Specifies MIME type of result
  - Cannot be used conditionally
    - Use <% response.setContentType(...); %> instead
- The isThreadSafe attribute
  - Turns off concurrent access
  - Use explicit synchronization instead
- The errorPage and isErrorPage attributes
  - Specifies "emergency" error handling pages