



# Java Server Page

March, 2019

# Agenda

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

# The Need for JSP

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

# 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>`
- 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

# Advantages of JSP (Continued)

- **Versus Velocity or WebMacro**
  - Standard
- **Versus client-side JavaScript (in browser)**
  - Capabilities mostly do not overlap with JavaScript, but
    - You control server, not client
    - Richer language
- **Versus server-side JavaScript (eg, LiveWire, BroadVision, JRun)**
  - Richer language
- **Versus static HTML**
  - Dynamic features
  - Adding dynamic features no longer "all or nothing" decision

# Setting Up Your Environment

- **Set your CLASSPATH.**
- **Compile your code.**
- **Use packages to avoid name conflicts.**
- **Put JSP page in special directory.**
  - *install\_dir\webapps\ROOT\* (HTML and JSP -- Tomcat)
  - *install\_dir\servers\default\default-app* (JRun)
  - Some servers reserve certain parts of Web hierarchy for JSP pages. Tomcat 3 and JRun (standalone) don't.
- **Use special URL to invoke JSP page.**
- **Caveats**
  - Previous rules about CLASSPATH, install dirs, etc., still apply to regular Java classes used by a JSP page



# Example

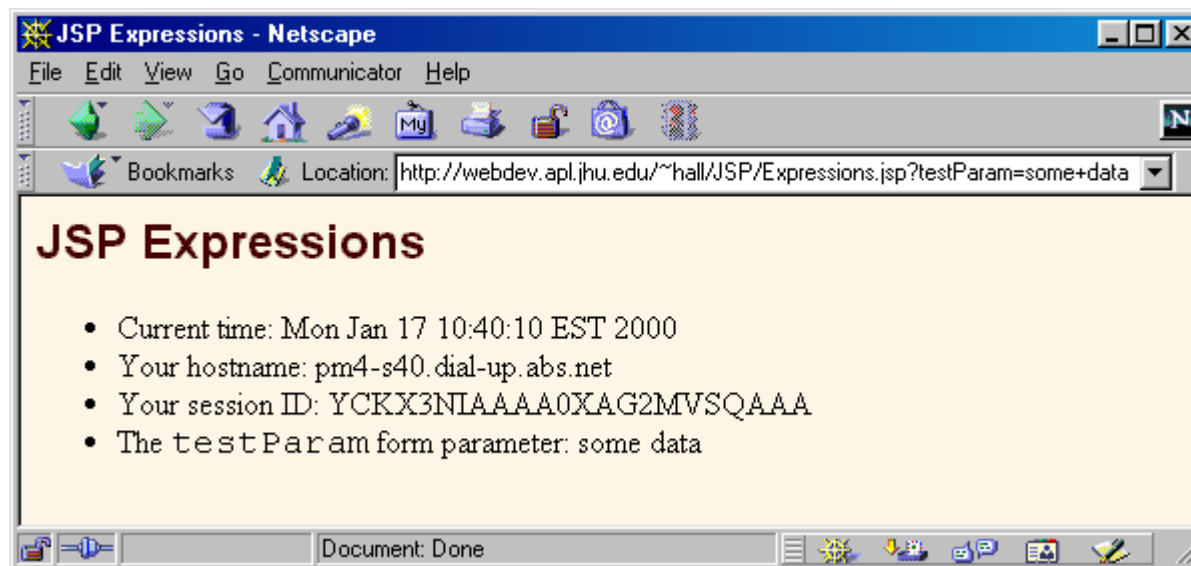
```
<!DOCTYPE HTML PUBLIC
    "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>JSP Expressions</TITLE>
<META NAME="author" CONTENT="Marty Hall">
<META NAME="keywords"
    CONTENT="JSP,expressions,JavaServer,Pages,servlets">
<META NAME="description"
    CONTENT="A quick example of JSP expressions.">
<LINK REL=STYLESHEET
    HREF="JSP-Styles.css"
    TYPE="text/css">
</HEAD>
```

# Example (Continued)

```
<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>
</HTML>
```

# Example Result

- **If location was**
  - C:\jakarta-tomcat-4.0\webapps\ROOT\Expressions.jsp or
  - C:\Program Files\Allaire\JRun\servers\default\default-app\Expressions.jsp
- **URL would be**
  - <http://localhost/Expressions.jsp>



# 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/Shockwave/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

		Request #1	Request #2		Request #3	Request #4		Request #5	Request #6
JSP page translated into servlet	Page first written	Yes	No	Server restarted	No	No	Page modified	Yes	No
Servlet compiled		Yes	No		No	No		Yes	No
Servlet instantiated and loaded into server's memory		Yes	No		Yes	No		Yes	No
init (or equivalent) called		Yes	No		Yes	No		Yes	No
doGet (or equivalent) called		Yes	Yes		Yes	Yes		Yes	Yes

# JSP/Servlets in the Real World

- ofoto.com:  
print and  
manage  
digital and  
conventional  
photos.



# JSP/Servlets in the Real World

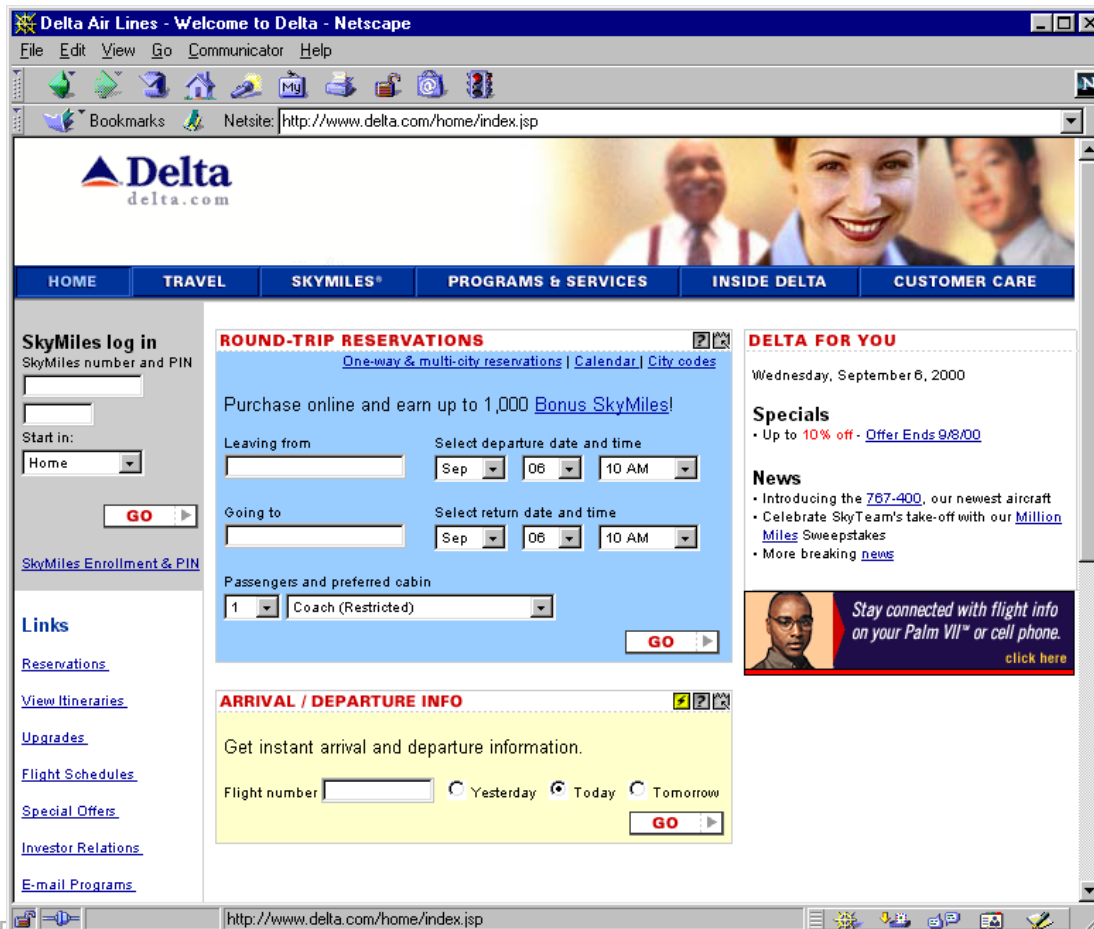
- **First USA Bank: largest credit card issuer in the world; most on-line banking customers**





# JSP/Servlets in the Real World

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



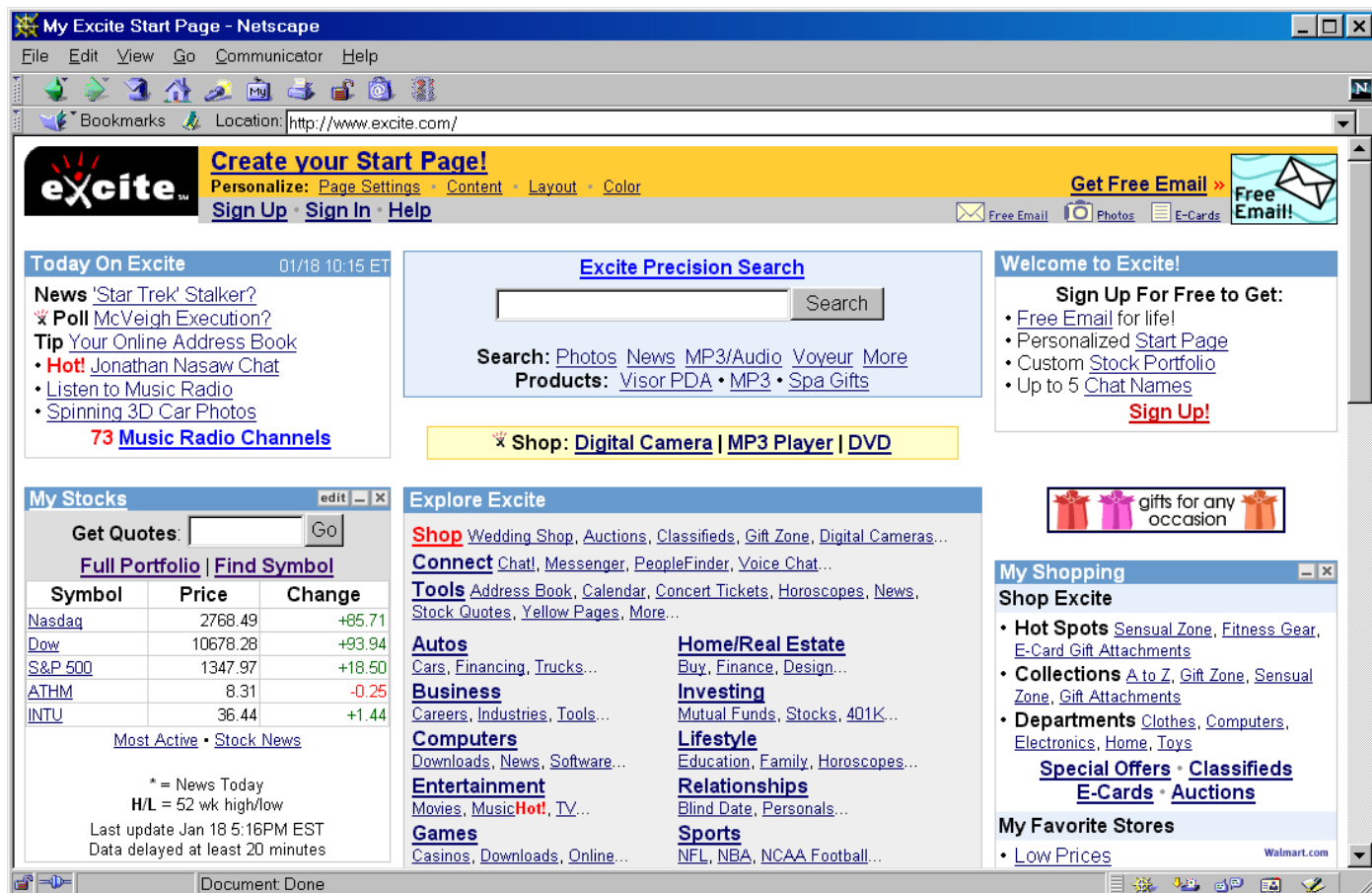
# JSP/Servlets in the Real World

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



# JSP/Servlets in the Real World

- **Excite:** one of the top five Internet portals; one of the ten busiest sites on the Web



# 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() %>`

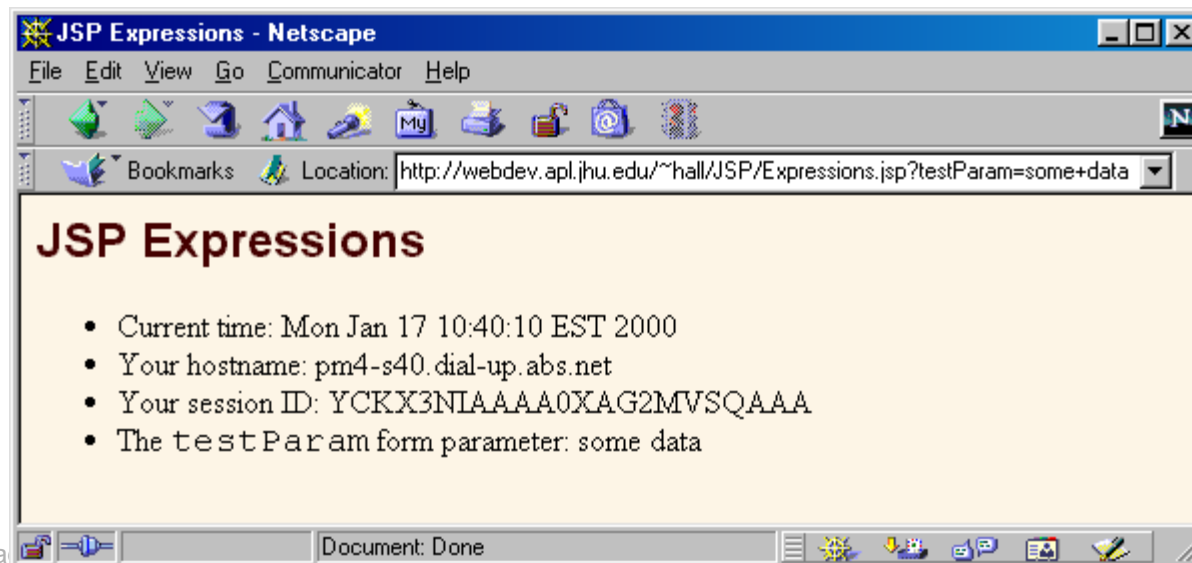
<LI>Your session ID: `<%= session.getId() %>`

<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

```
<!DOCTYPE HTML PUBLIC
    "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
    <TITLE>Color Testing</TITLE>
</HEAD>
<%
String bgColor = request.getParameter("bgColor") ;
boolean hasExplicitColor;
if (bgColor != null) {
    hasExplicitColor = true;
} else {
    hasExplicitColor = false;
    bgColor = "WHITE";
}
%>
```

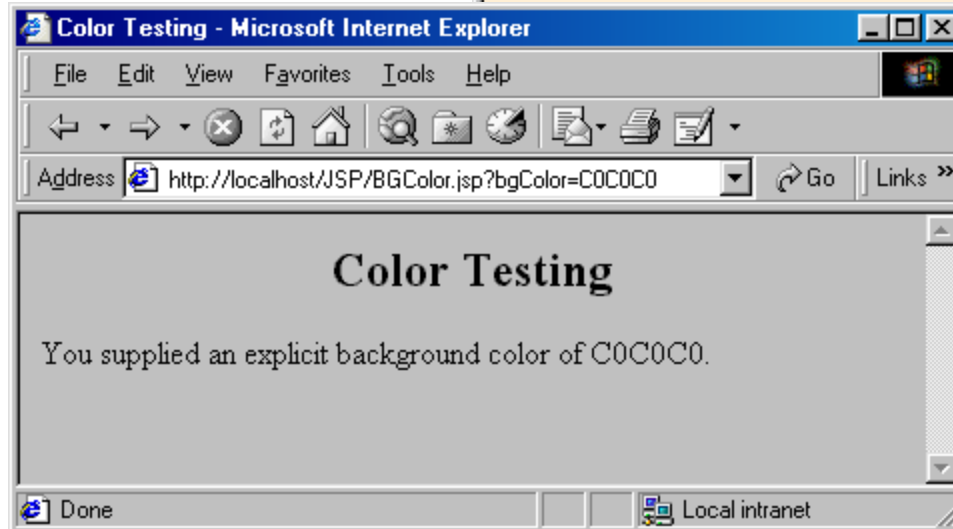
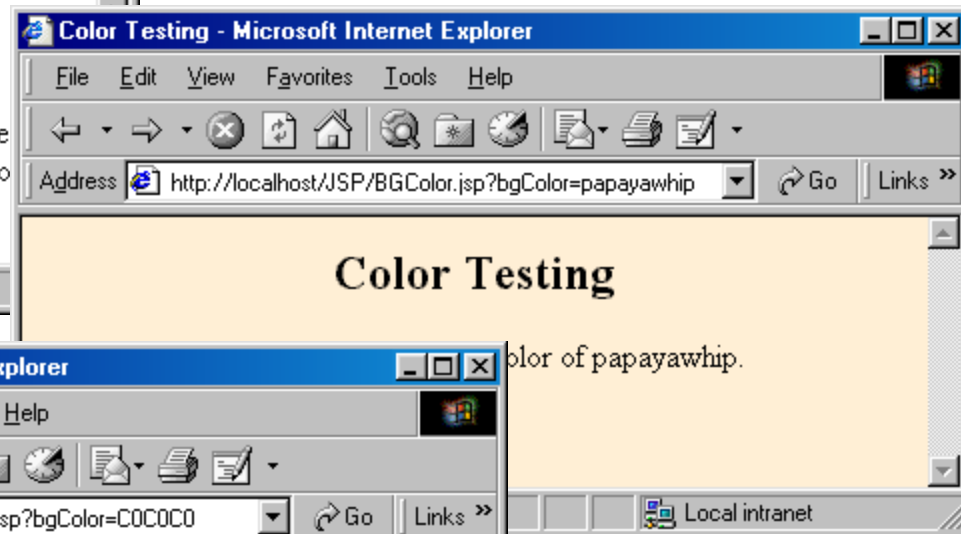
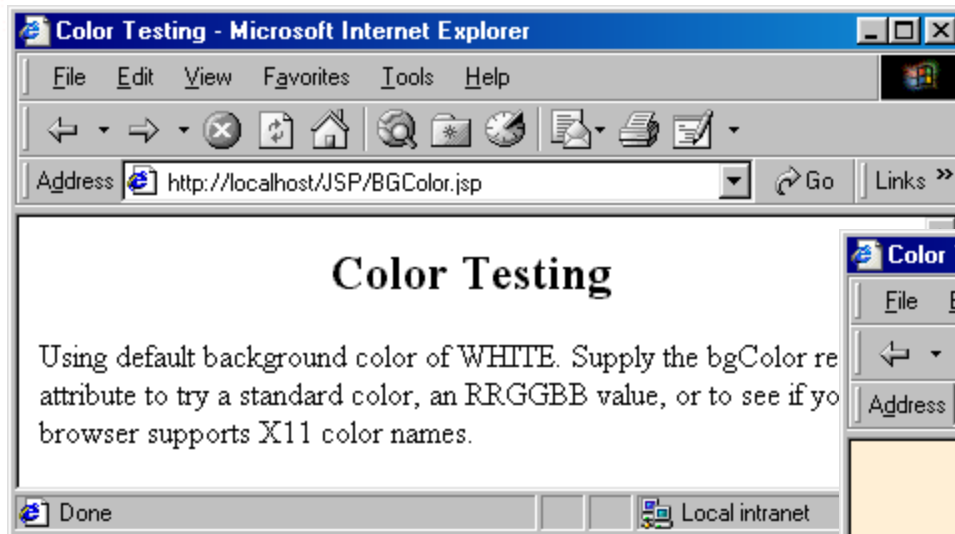


# Example Using JSP Scriptlets (Continued)

```
<BODY BGCOLOR="<%= bgColor %>">
<H2 ALIGN="CENTER">Color Testing</H2>
<%
if (hasExplicitColor) {
    ...
} else {
    ...
}
%>

</BODY>
</HTML>
```

# JSP Scriptlets: Results



color of papayawhip.

# Using Scriptlets to Make Parts of the JSP File Conditional

- **Point**

- Scriptlets 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>lousy</B> day!
<% } %>
```

- **Representative result**

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

```
<!DOCTYPE HTML PUBLIC
    "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>JSP Declarations</TITLE>
<LINK REL=STYLESHEET
    HREF="JSP-Styles.css"
    TYPE="text/css">
</HEAD>

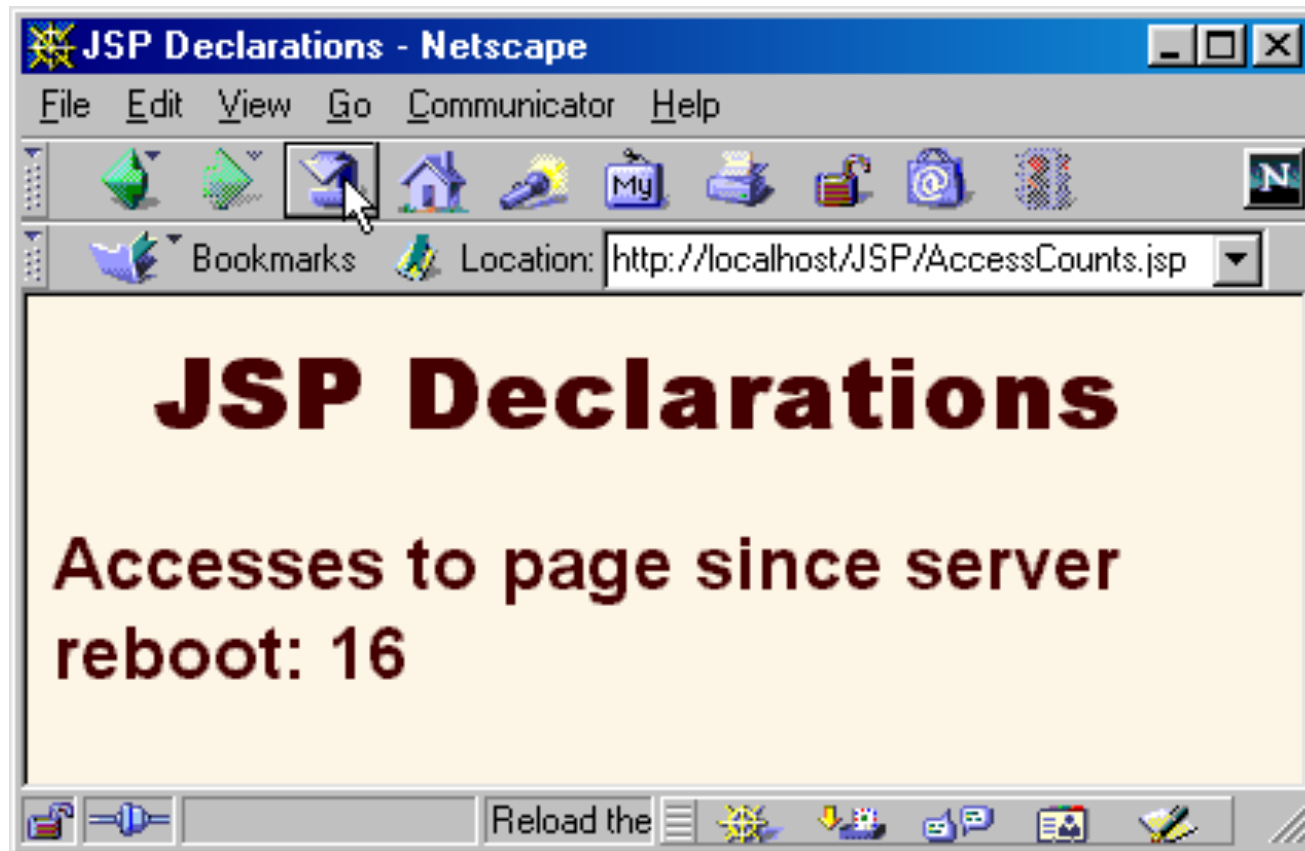
<BODY>
<H1>JSP Declarations</H1>

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

</BODY>
</HTML>
```

# JSP Declarations: Result

- After 15 total visits by an arbitrary number of different clients





# 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
- For Tomcat, this is  
*install dir\webapps\ROOT\WEB-INF\classes* or  
*...\ROOT\WEB-INF\classes\directoryMatchingPackage*
  - **Always use packages for utilities that will be used by JSP!**



# Example of import Attribute

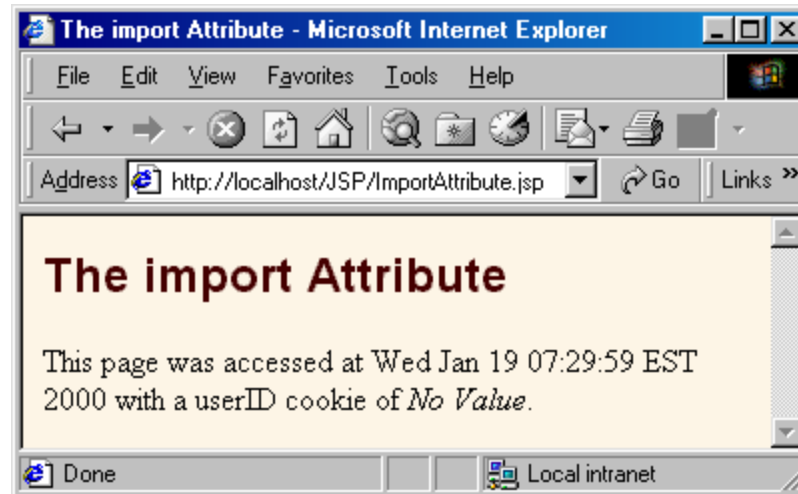
```
...  
<BODY>  
<H2>The import Attribute</H2>  
<%-- JSP page directive --%>  
<%@ page import="java.util.*,coreservlets.*" %>  
  
<%-- JSP Declaration --%>  
<%!  
private String randomID() {  
    int num = (int) (Math.random()*10000000.0);  
    return("id" + num);  
}  
  
private final String NO_VALUE = "<I>No Value</I>";  
%>
```

# Example of import Attribute (cont)

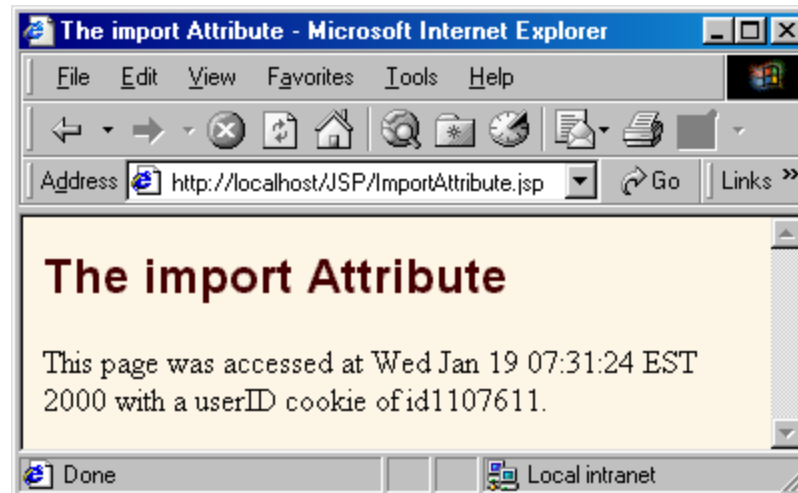
```
<%-- JSP Scriptlet --%>
<%
Cookie[] cookies = request.getCookies();
String oldID =
    ServletUtilities.getCookieValue(cookies, "userID", NO_VALUE);
String newID;
if (oldID.equals(NO_VALUE)) {
    newID = randomID();
} else {
    newID = oldID;
}
LongLivedCookie cookie = new LongLivedCookie("userID", 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

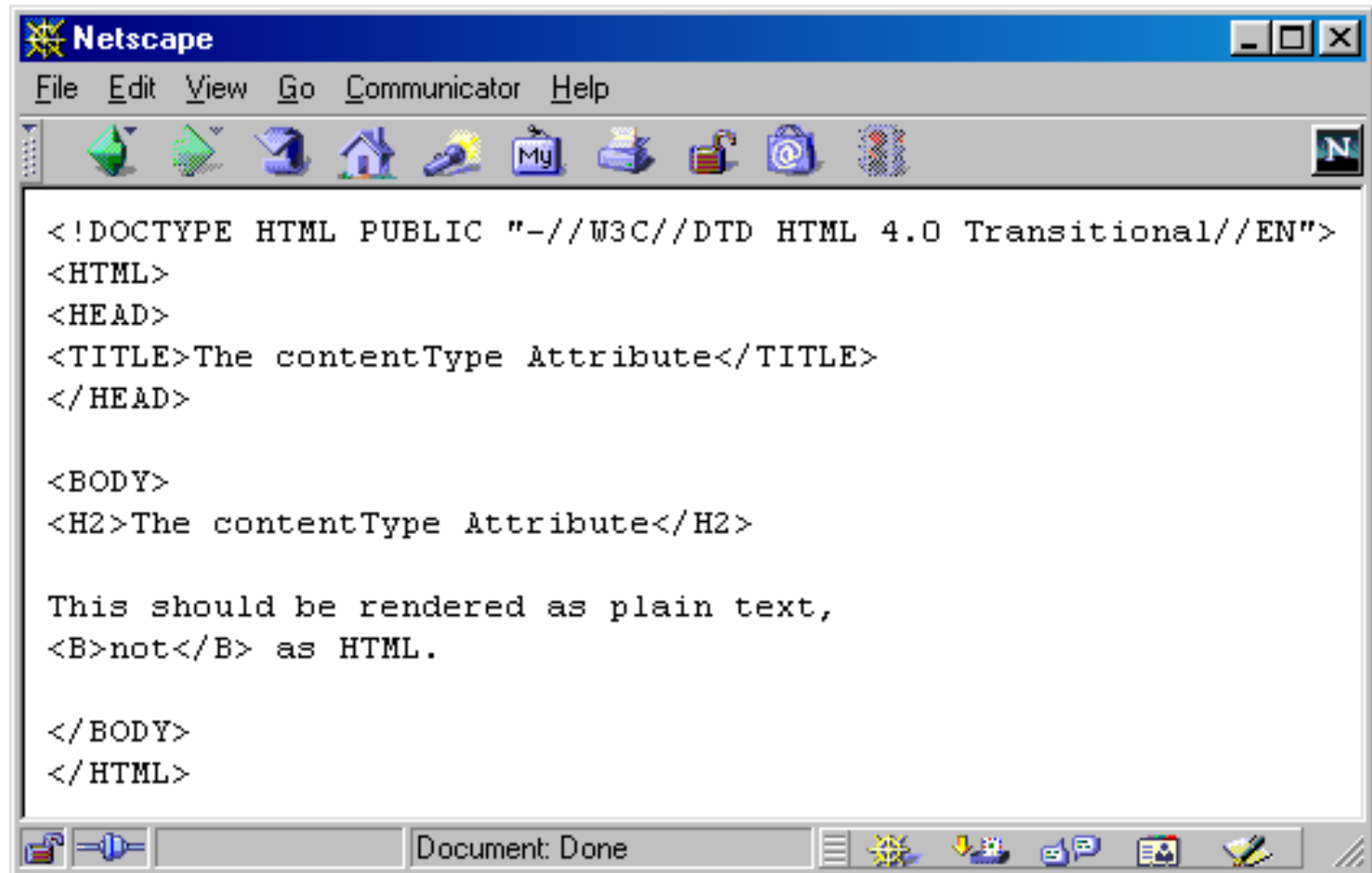
# Using contentType to Generate Plain Text Documents

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

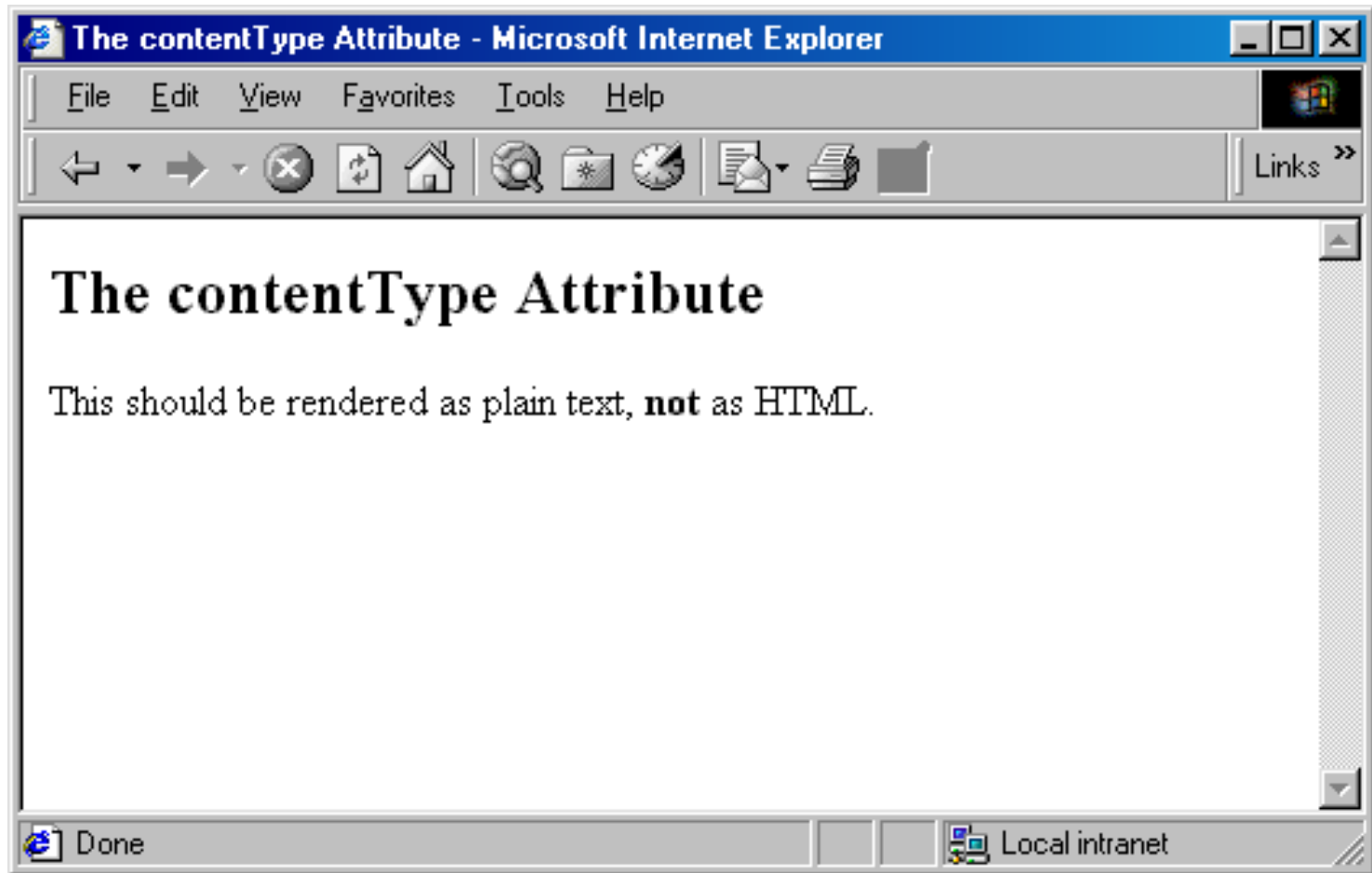
<H2>The contentType Attribute</H2>
<%@ page contentType="text/plain" %>
This should be rendered as plain text,
<B>not</B> as HTML.

</BODY>
</HTML>
```

# Plain Text Documents in Netscape (Correct)



# Plain Text Documents in Internet Explorer (Incorrect)

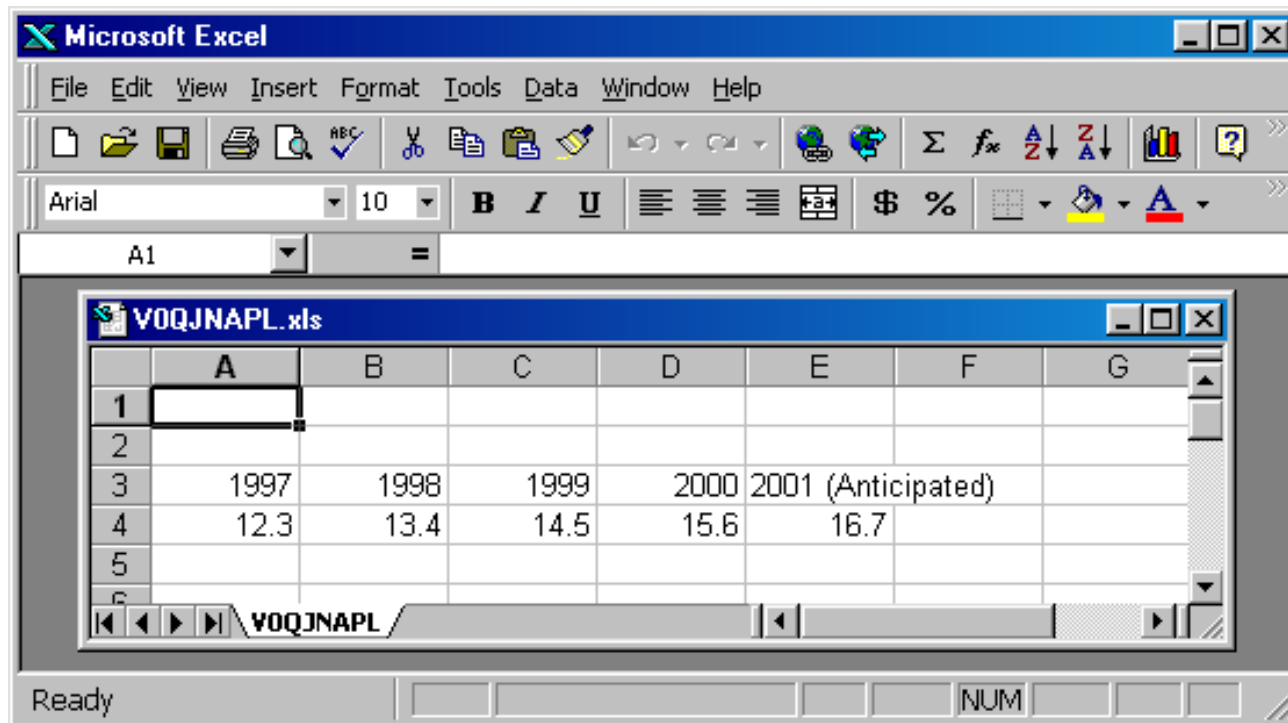


# Generating Excel Spreadsheets

```
<%@ 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 Spreadsheets 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");  
}  
%>  
<!DOCTYPE ...>  
<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
```

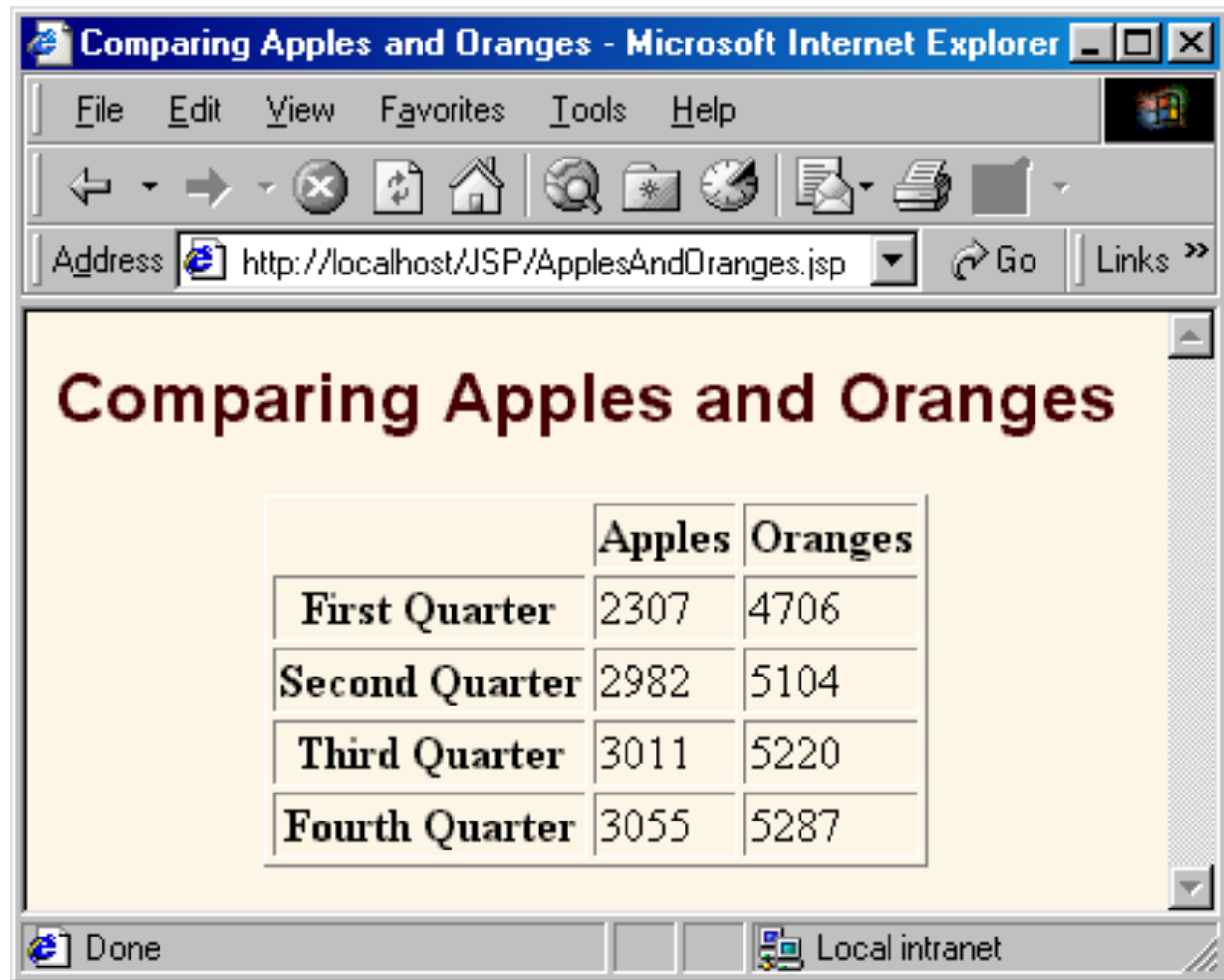
```
</TABLE>
```

```
</CENTER>
```

```
</BODY>
```

```
</HTML>
```

# Apples and Oranges: Default Result

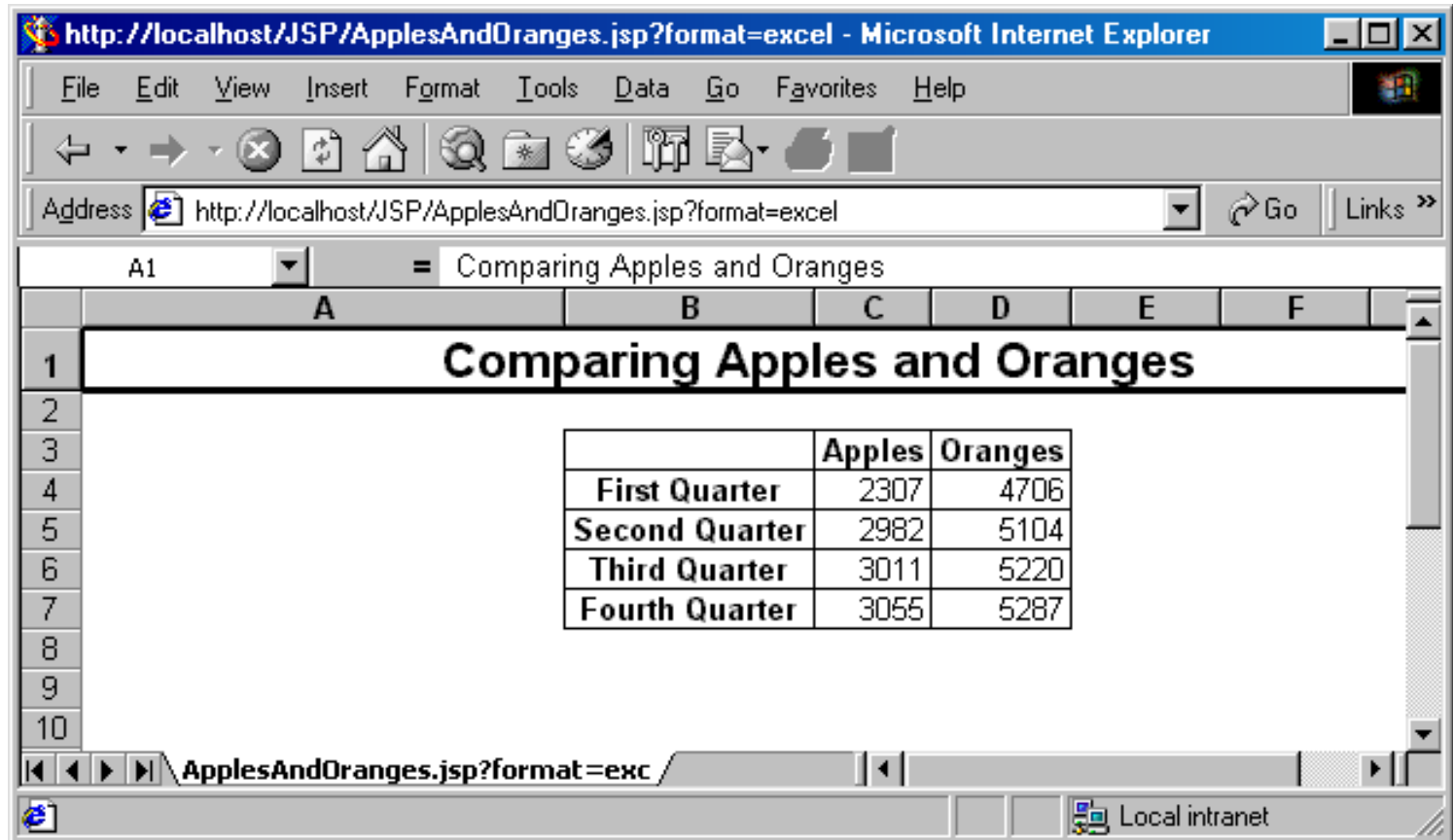


The screenshot shows a Microsoft Internet Explorer window with the title 'Comparing Apples and Oranges - Microsoft Internet Explorer'. The address bar displays 'http://localhost/JSP/ApplesAndOranges.jsp'. The main content area features a large heading 'Comparing Apples and Oranges' and a table with the following data:

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

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

# Apples and Oranges: Result with format=excel



The screenshot shows a Microsoft Internet Explorer browser window with the address bar displaying `http://localhost/JSP/ApplesAndOranges.jsp?format=excel`. The browser's menu bar includes File, Edit, View, Insert, Format, Tools, Data, Go, Favorites, and Help. The address bar also contains a 'Go' button and a 'Links' button. The main content area displays an Excel spreadsheet. The spreadsheet has a title bar that reads 'A1 = Comparing Apples and Oranges'. The first row of the spreadsheet is a header row with the title 'Comparing Apples and Oranges' centered across columns A through F. Below the header, there is a table with four rows and three columns. The first column lists the quarters, and the second and third columns list the counts for Apples and Oranges respectively. The data is as follows:

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

The spreadsheet interface includes a row and column grid on the left and bottom. The status bar at the bottom of the browser window shows 'Local intranet'.

# 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="sizekb" %>`
- `<%@ 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 Cont.)

```
<%  
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>  
...
```

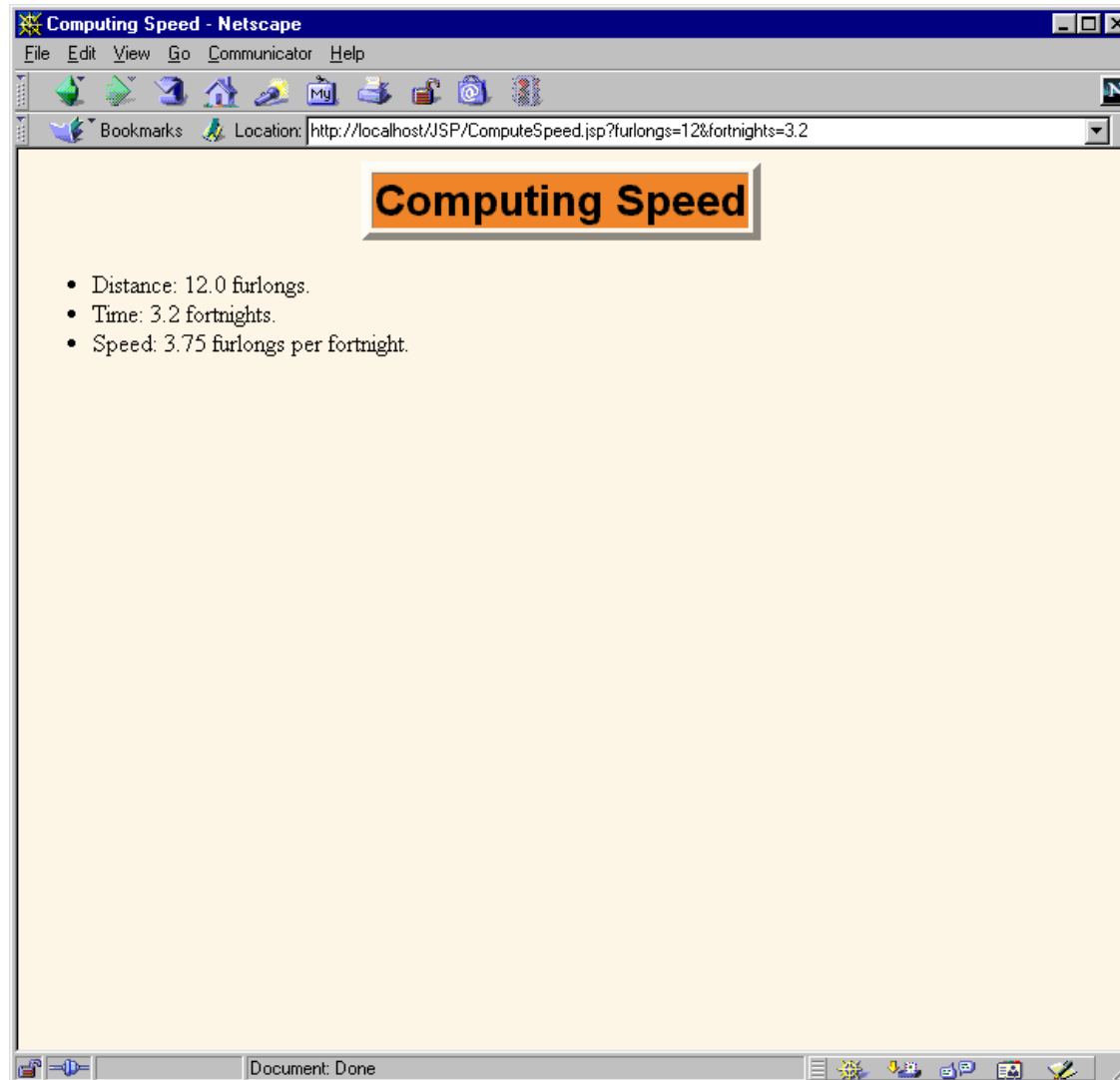
# Error Pages: Example (SpeedErrors.jsp)

```
...
<BODY>

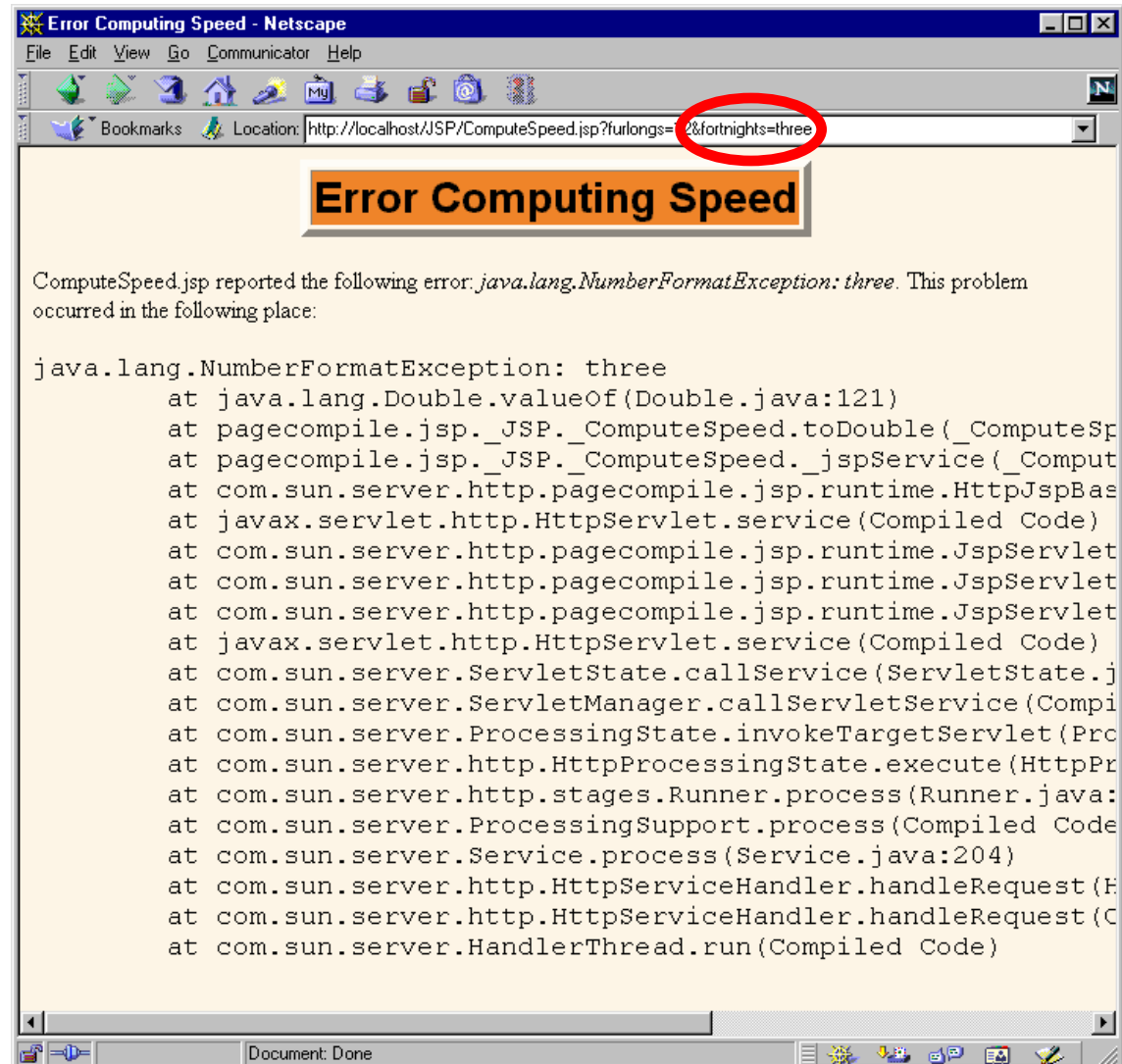
<%@ page isErrorPage="true" %>

<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



# 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