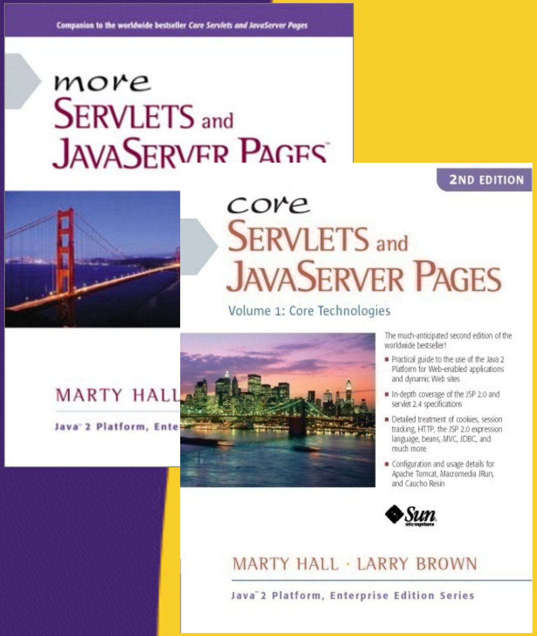
Course Material Usage Rules

- **PowerPoint slides for use only in for-credit courses at degree-granting institutions**
  - Slides not permitted for commercial training courses except when taught by coreservlets.com (see http://courses.coreservlets.com).
- **Slides may be modified by instructor**
  - But please retain reference to coreservlets.com
- **Instructor may give PDF or hardcopy to students**
  - But should protect the PowerPoint files

This slide is suppressed in Slide Show mode

Invoking Java Code with JSP Scripting Elements

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/csajsp2.html



Agenda

- Static vs. dynamic text
- Dynamic code and good JSP design
- JSP expressions
- Servlets vs. JSP pages for similar tasks
- JSP scriptlets
- JSP declarations
- Predefined variables
- Comparison of expressions, scriptlets, and declarations
- XML syntax for JSP pages

Intro

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
- Servlet/JSP combo (MVC)
- MVC with JSP expression language
- Custom tags
- MVC with beans, custom tags, and a framework like JSF 2.0

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
- **Escaping <%**
  - To get <% in output, use <\%

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
- **XML syntax**
  - See slides at end of the lecture for an XML-compatible way of representing JSP pages and scripting elements

JSP Expressions: <%= value %>

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>
  - You cannot mix versions within a single page. You must use XML for *entire* page if you use jsp:expression.
    - See slides at end of this lecture

JSP/Servlet Correspondence

- **Original JSP**

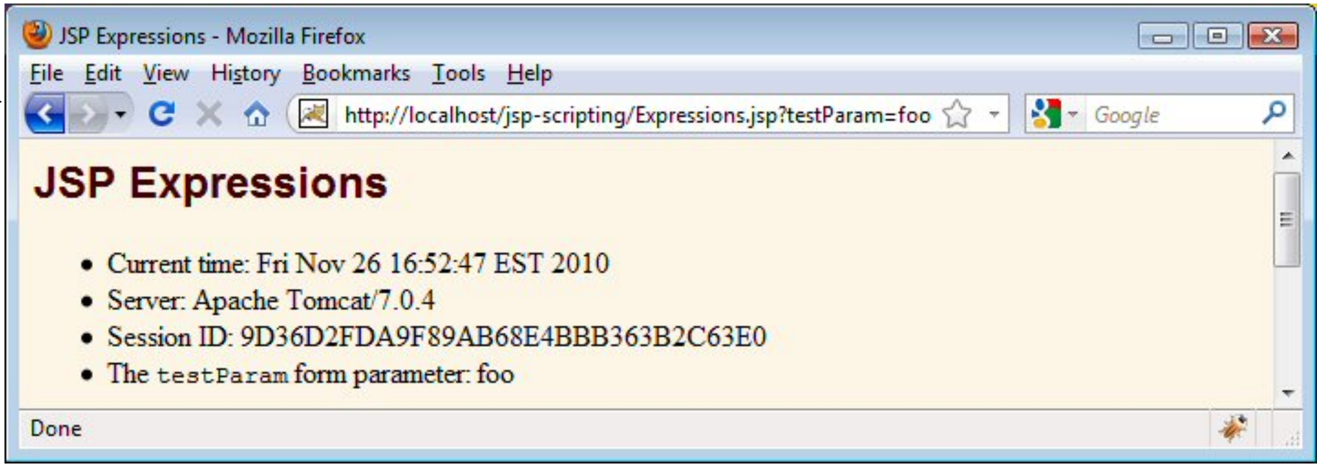
```
<H1>A Random Number</H1>
<%= Math.random() %>
```

- **Representative resulting servlet code**

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

JSP Expressions: Example

```
…<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Server: <%= application.getServerInfo() %>
  <LI>Session ID: <%= session.getId() %>
  <LI>The <CODE>testParam</CODE> form parameter:
      <%= request.getParameter("testParam") %>
</UL>
</BODY></HTML>
```


JSP Expressions - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/jsp-scripting/Expressions.jsp?testParam=foo
Google
Done

JSP Expressions

- Current time: Fri Nov 26 16:52:47 EST 2010
- Server: Apache Tomcat/7.0.4
- Session ID: 9D36D2FDA9F89AB68E4BBB363B2C63E0
- The `testParam` form parameter: foo

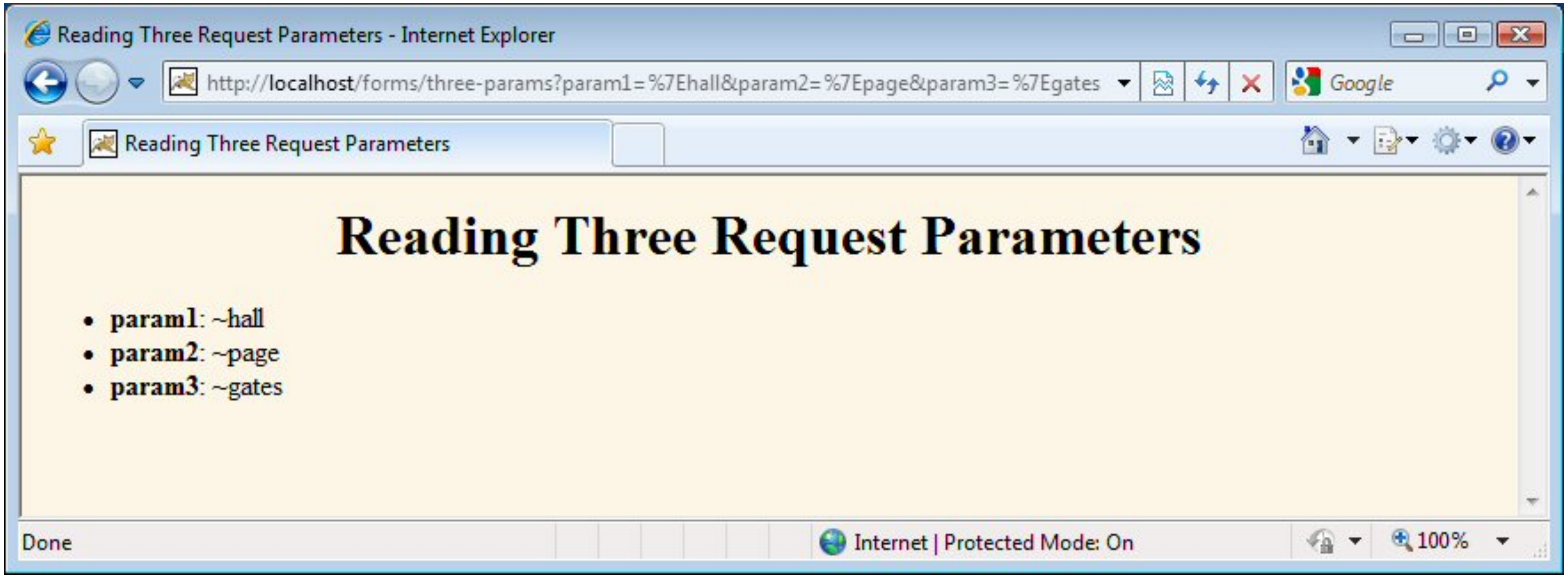
Predefined Variables

- **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().

Comparing Servlets to JSP: Reading Three Params (Servlet)

```
@WebServlet("/three-params")
public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    …
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>"+title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI><B>param1</B>: "
                + request.getParameter("param1") + "\n" +
                "  <LI><B>param2</B>: "
                + request.getParameter("param2") + "\n" +
                "  <LI><B>param3</B>: "
                + request.getParameter("param3") + "\n" +
                "</UL>\n" +
                "</BODY></HTML>");
  }
}
```

Reading Three Params (Servlet): Result


Reading Three Request Parameters - Internet Explorer
Google
Reading Three Request Parameters
Done
Internet | Protected Mode: On
100%

http://localhost/forms/three-params?param1=%7Ehall&param2=%7Epage&param3=%7Egates

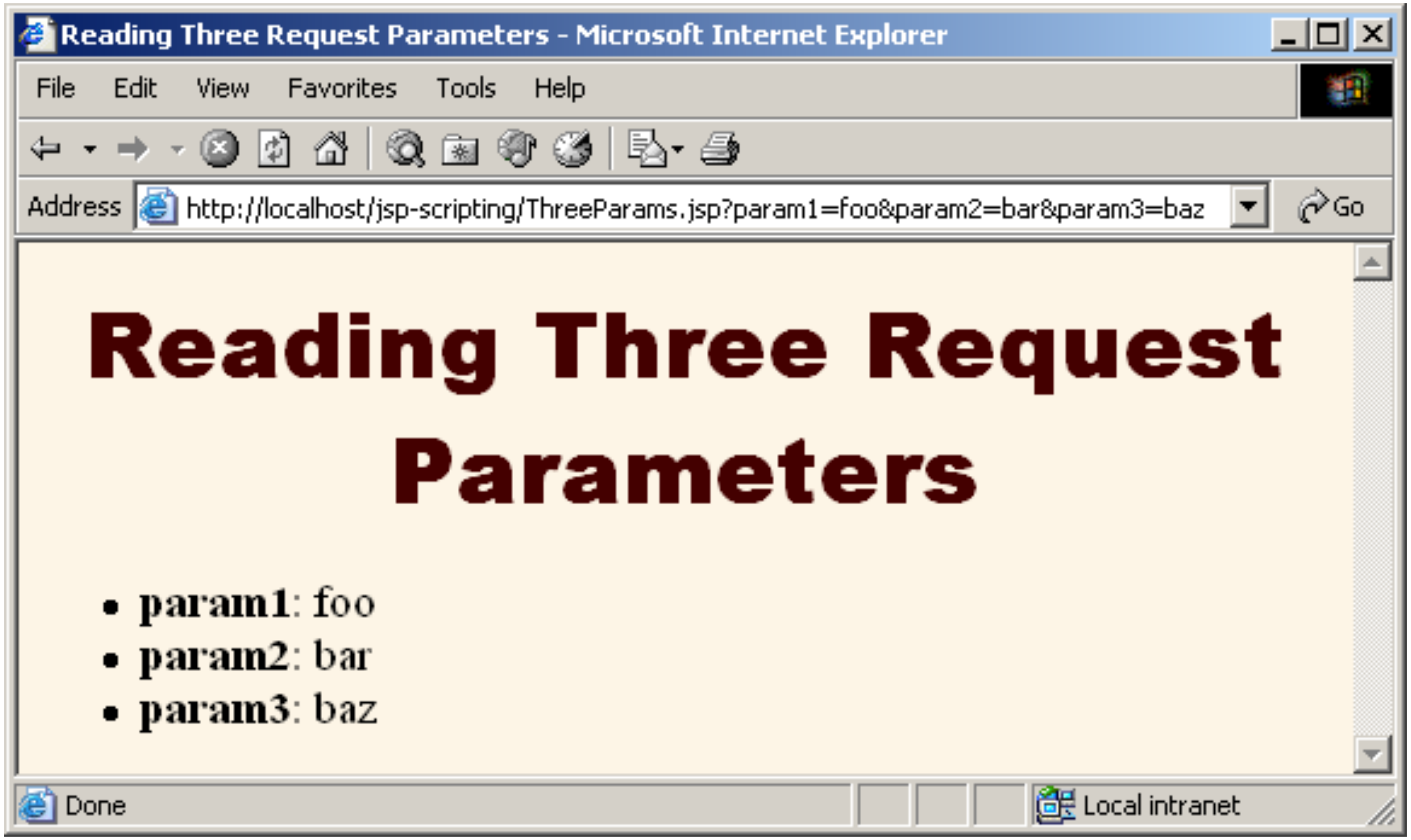
Reading Three Request Parameters

- **param1**: ~hall
- **param2**: ~page
- **param3**: ~gates

Comparing Servlets to JSP: Reading Three Params (JSP)

```
<!DOCTYPE …>
<HTML>
<HEAD>
<TITLE>Reading Three Request Parameters</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>Reading Three Request Parameters</H1>
<UL>
  <LI><B>param1</B>:
      <%= request.getParameter("param1") %>
  <LI><B>param2</B>:
      <%= request.getParameter("param2") %>
  <LI><B>param3</B>:
      <%= request.getParameter("param3") %>
</UL>
</BODY></HTML>
```

Reading Three Params (Servlet): Result


Reading Three Request Parameters - Microsoft Internet Explorer

http://localhost/jsp-scripting/ThreeParams.jsp?param1=foo&param2=bar&param3=baz

Reading Three Request Parameters

- **param1**: foo
- **param2**: bar
- **param3**: baz

JSP Scriptlets: <% Code %>

JSP Scriptlets

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

JSP/Servlet Correspondence

- **Original JSP**

```
<H2>foo</H2>
<%= bar() %>
<% baz(); %>
```

- **Representative resulting servlet code**

```
 public void _jspService(HttpServletRequest request,
                         HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("text/html");
  HttpSession session = request.getSession();
  JspWriter out = response.getWriter();
  out.println("<H2>foo</H2>");
  out.println(bar());
  baz();
  ...
}
```

JSP Scriptlets: Example

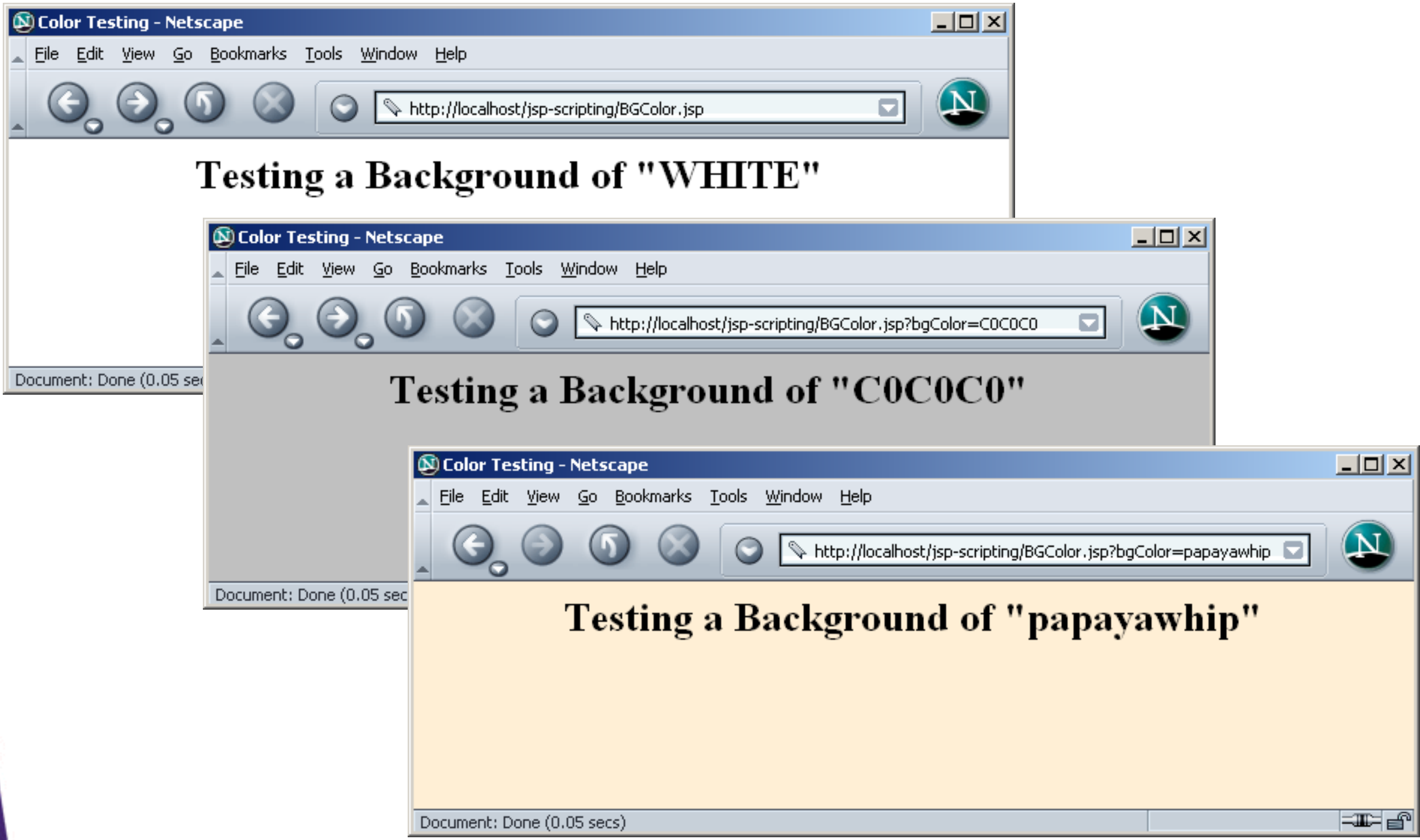
- **Suppose you want to let end users customize the background color of a page**
  - What is wrong with the following code?

```
<BODY BGCOLOR=
  "<%= request.getParameter("bgColor") %>">
```

JSP Scriptlets: Example

```
<!DOCTYPE …>
<HTML>
<HEAD>
  <TITLE>Color Testing</TITLE>
</HEAD>
<%
String bgColor = request.getParameter("bgColor");
if ((bgColor == null)||(bgColor.trim().equals(""))){
  bgColor = "WHITE";
}
%>
<BODY BGCOLOR="<%= bgColor %>">
<H2 ALIGN="CENTER">Testing a Background of
"<%= bgColor %>".</H2>
</BODY></HTML>
```

JSP Scriptlets: Result



http://localhost/jsp-scripting/BGColor.jsp

Testing a Background of "WHITE"

Document: Done (0.05 se

http://localhost/jsp-scripting/BGColor.jsp?bgColor=C0C0C0

Testing a Background of "C0C0C0"

Document: Done (0.05 sec

http://localhost/jsp-scripting/BGColor.jsp?bgColor=papayawhip

Testing a Background of "papayawhip"

Document: Done (0.05 secs)

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>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: <%! Code %>

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() %>
```

- **Better alternative:**
  - Make randomHeading a static method in a separate class

JSP/Servlet Correspondence

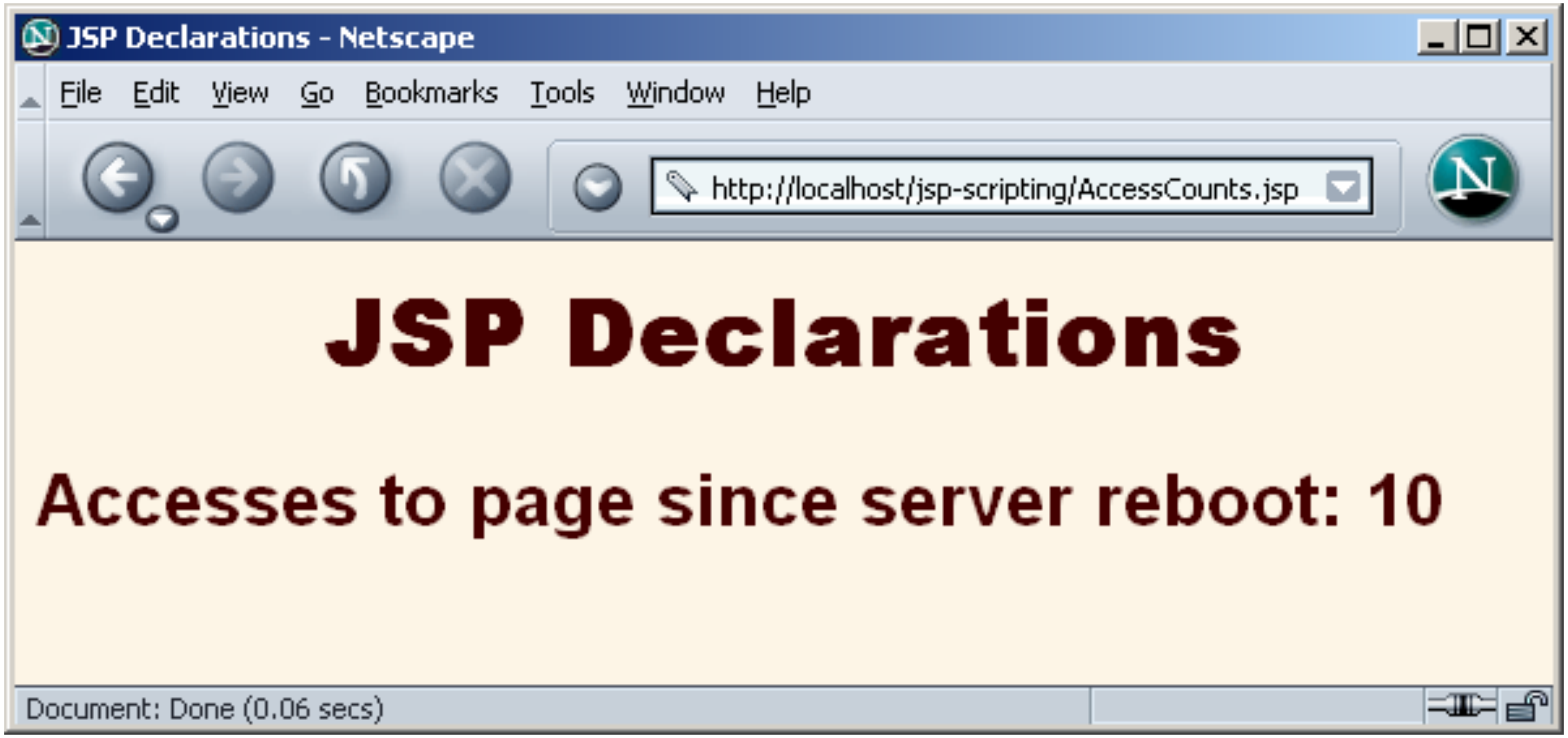
- **Possible resulting servlet code**

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

JSP Declarations: Example

```
<!DOCTYPE …>
<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


JSP Declarations - Netscape
File Edit View Go Bookmarks Tools Window Help

http://localhost/jsp-scripting/AccessCounts.jsp

JSP Declarations

Accesses to page since server reboot: 10

Document: Done (0.06 secs)

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.**

```
public class SomeClass {
  public static void someMethod(HttpSession s) {
    doSomethingWith(s);
  }
}
…
<% somePackage.SomeClass.someMethod(session); %>
```

- **Notes**
  - Same issue if you use methods in JSP declarations
    - But separate classes preferred over JSP declarations
  - println of JSPWwriter throws IOException
    - Use “throws IOException” for methods that use println

Comparing JSP Scripting Elements

Using Expressions, Scriptlets and Declarations

- **Task 1**
  - Output a bulleted list of five random ints from 1 to 10.
    - Since the structure of this page is fixed and we use a separate helper class for the randomInt method, JSP expressions are all that is needed.
- **Task 2**
  - Generate a list of between 1 and 10 entries (selected at random), each of which is a number between 1 and 10.
    - Because the number of entries in the list is dynamic, a JSP scriptlet is needed.
- **Task 3**
  - Generate a random number on the first request, then show the same number to all users until the server is restarted.
    - Instance variables (fields) are the natural way to accomplish this persistence. Use JSP declarations for this.

Helper Class: RanUtilities

```
package coreservlets; // Always use packages!!

/** Simple utility to generate random integers. */

public class RanUtilities {

  /** A random int from 1 to range (inclusive). */

  public static int randomInt(int range) {
    return(1 + ((int)(Math.random() * range)));
  }

  public static void main(String[] args) {
    int range = 10;
    try {
      range = Integer.parseInt(args[0]);
    } catch(Exception e) { // Array index or number format
      // Do nothing: range already has default value.
    }
    for(int i=0; i<100; i++) {
      System.out.println(randomInt(range));
}}}
```

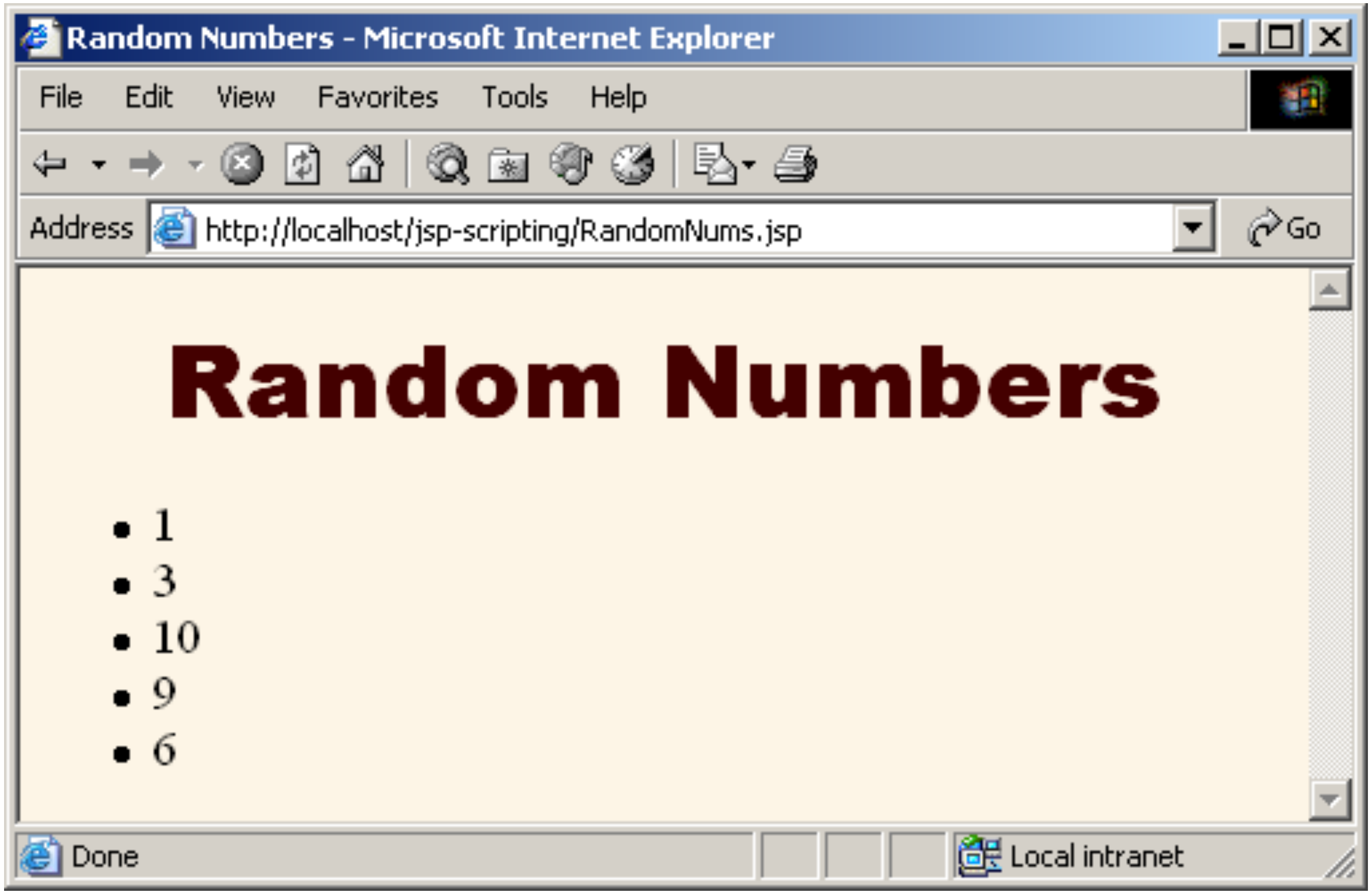
Task 1: JSP Expressions (Code)

```
<!DOCTYPE …>
<HTML>
<HEAD>
<TITLE>Random Numbers</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>Random Numbers</H1>
<UL>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
</UL>
</BODY></HTML>
```

Instead of using the package name in each call, you can also import the package first, then call the static methods with no packages:

```
<%@ page import="coreservlets.*" %>
…
<LI><%= RanUtils.randomInt(10) %>
```

Task 1: JSP Expressions (Result)


Random Numbers - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
http://localhost/jsp-scripting/RandomNums.jsp
Go
Done
Local intranet

Random Numbers

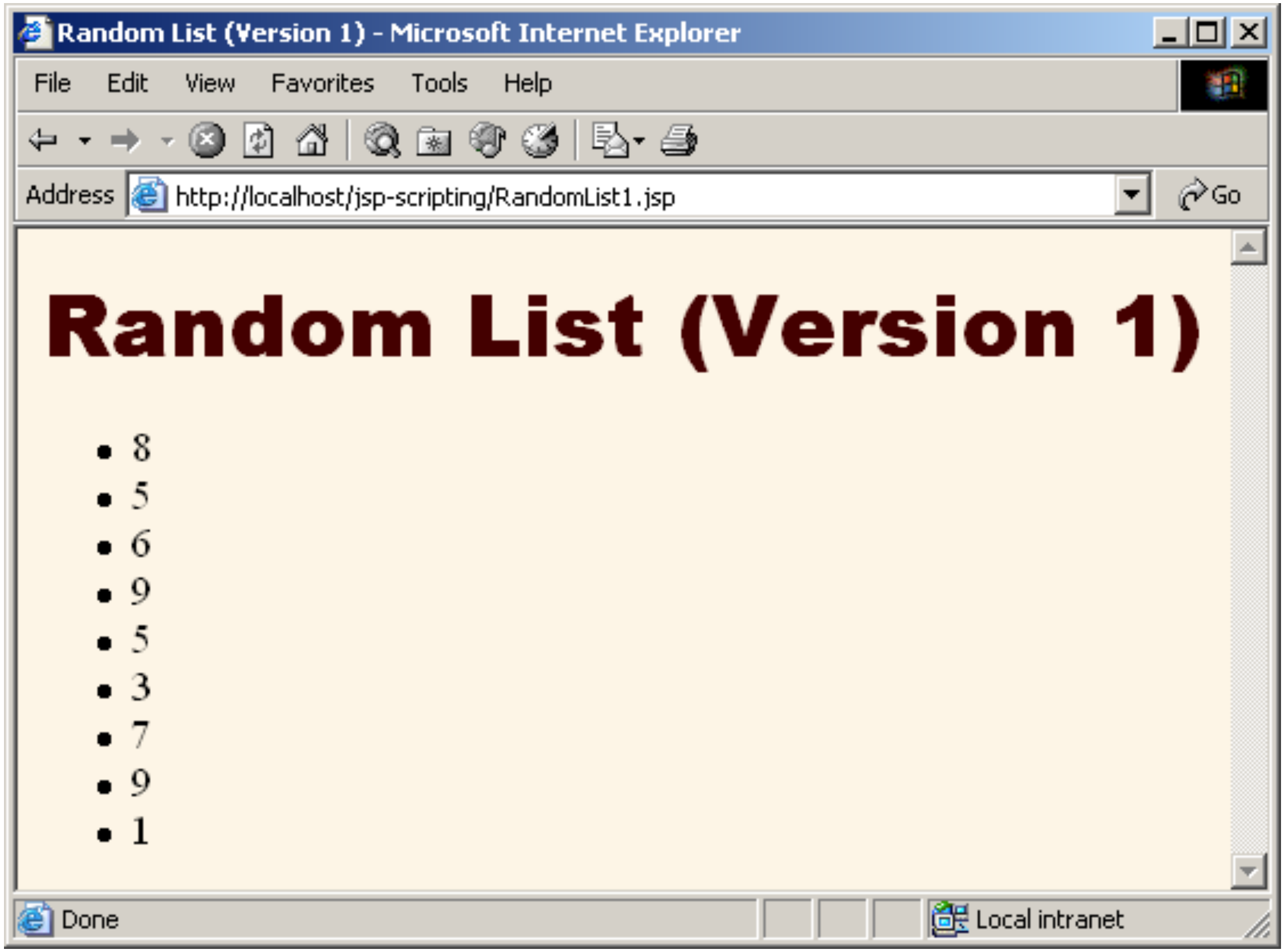
- 1
- 3
- 10
- 9
- 6

Task 2: JSP Scriptlets (Code: Version 1)

```
<!DOCTYPE …>
<HTML>
<HEAD>
<TITLE>Random List (Version 1)</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>Random List (Version 1)</H1>
<UL>
<%
int numEntries = coreservlets.RanUtilities.randomInt(10);
for(int i=0; i<numEntries; i++) {
  out.println("<LI>" +
   coreservlets.RanUtilities.randomInt(10));
}
%>
</UL>
</BODY></HTML>
```

Again, you can import the package with <%@ page import="coreservlets.*" %>, then omit the package name in the calls to the static method.

Task 2: JSP Scriptlets (Result: Version 1)


Random List (Version 1) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/jsp-scripting/RandomList1.jsp
Go
Done
Local intranet

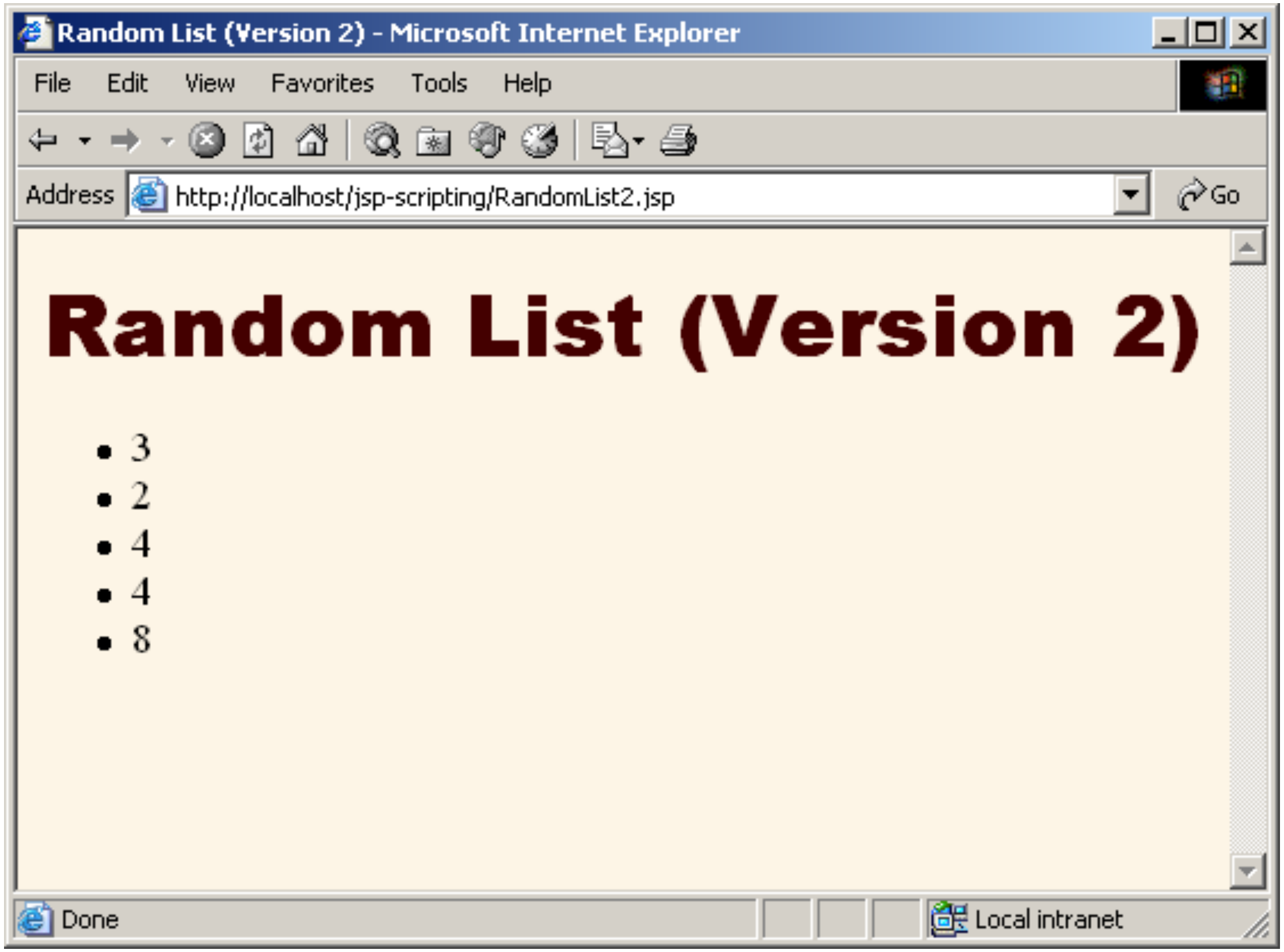
Random List (Version 1)

- 8
- 5
- 6
- 9
- 5
- 3
- 7
- 9
- 1

Task 2: JSP Scriptlets (Code: Version 2)

```
<!DOCTYPE …>
<HTML>
<HEAD>
<TITLE>Random List (Version 2)</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>Random List (Version 2)</H1>
<UL>
<%
int numEntries = coreservlets.RanUtilities.randomInt(10);
for(int i=0; i<numEntries; i++) {
%>
<LI><%= coreservlets.RanUtilities.randomInt(10) %>
<% } %>
</UL>
</BODY></HTML>
```

Task 2: JSP Scriptlets (Result: Version 2)


Random List (Version 2) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Go
Done
Local intranet

http://localhost/jsp-scripting/RandomList2.jsp

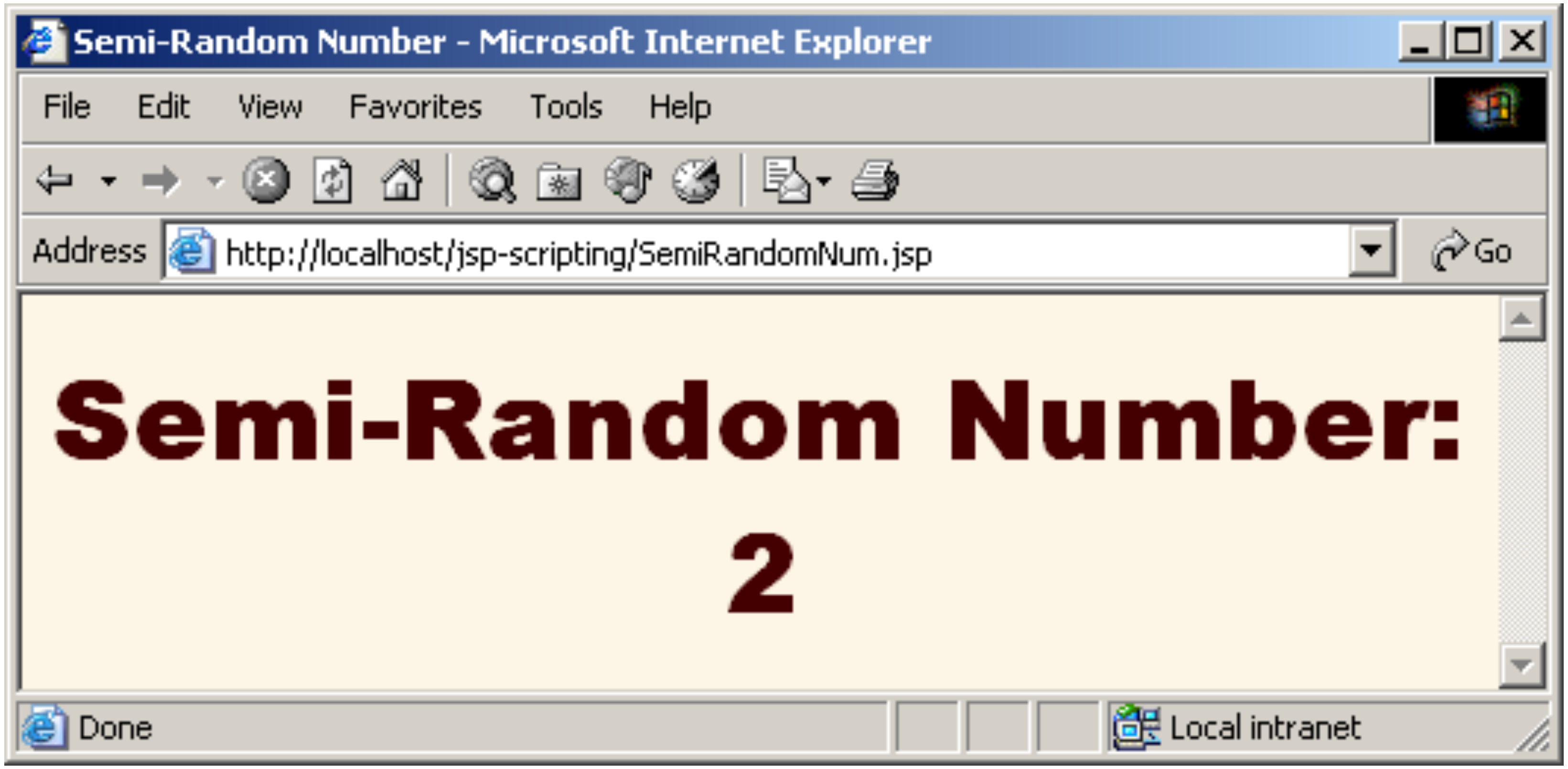
Random List (Version 2)

- 3
- 2
- 4
- 4
- 8

Task 3: JSP Declarations (Code)

```
<!DOCTYPE …>
<HTML>
<HEAD>
<TITLE>Semi-Random Number</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<%!
private int randomNum =
  coreservlets.RanUtilities.randomInt(10);
%>
<H1>Semi-Random Number:<BR><%= randomNum %></H1>
</BODY>
</HTML>
```

Task 3: JSP Declarations (Result)


Semi-Random Number - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Go
Done
Local intranet

http://localhost/jsp-scripting/SemiRandomNum.jsp

Semi-Random Number:
2

JSP Pages with XML Syntax

Why Two Versions?

- **Classic syntax is not XML-compatible**
  - <%= ... %>, <% ... %>, <%! ... %> are illegal in XML
  - HTML 4 is not XML compatible either
  - So, you cannot use XML editors like XML Spy
- **You might use JSP in XML environments**
  - To build xhtml pages
  - To build regular XML documents
    - You can use classic syntax to build XML documents, but it is sometimes easier if you are working in XML to start with
      - For Web services
      - For Ajax applications
- **So, there is a second syntax**
  - Following XML rules

XML Syntax for Generating XHTML Files (somefile.jspx)

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns:jsp="http://java.sun.com/JSP/Page">
<jsp:output
    omit-xml-declaration="true"
    doctype-root-element="html"
    doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" />
<jsp:directive.page contentType="text/html"/>
<head><title>Some Title</title></head>
<body bgcolor="#fdf5e6">
Body
</body></html>
```

The jsp namespace is required if you use jsp:blah commands. You can use other namespaces for other custom tag libraries.

Needed because of Internet Explorer bug where xhtml pages that have the XML declaration at the top run in quirks mode.

Builds DOCTYPE line.

For JSP pages in XML syntax, default content type is text/xml.

Normal xhtml content, plus JSP commands that use jsp:blah syntax, plus JSP custom tag libraries.

XML Syntax for Generating Regular XML Files (somefile.jspx)

```
<?xml version="1.0" encoding="UTF-8" ?>
<your-root-element xmlns:jsp="http://java.sun.com/JSP/Page">
  <your-tag1>foo</your-tag1>
  <your-tag2>bar</your-tag2>
<your-root-element>
```

- **Uses**
  - When you are sending to client that expects real XML
    - Ajax
    - Web services
    - Custom clients
  - Note
    - You can omit the xmlns declaration if you are not using any JSP tags. But then you could just use .xml extension.

XML Syntax for Generating HTML 4 Files (somefile.jspx)

- **Many extra steps required**
  - Enclose the entire page in jsp:root
  - Enclose the HTML in CDATA sections
    - Between <![CDATA[and]]>
    - Because HTML 4 does not obey XML rules
  - Usually not worth the bother

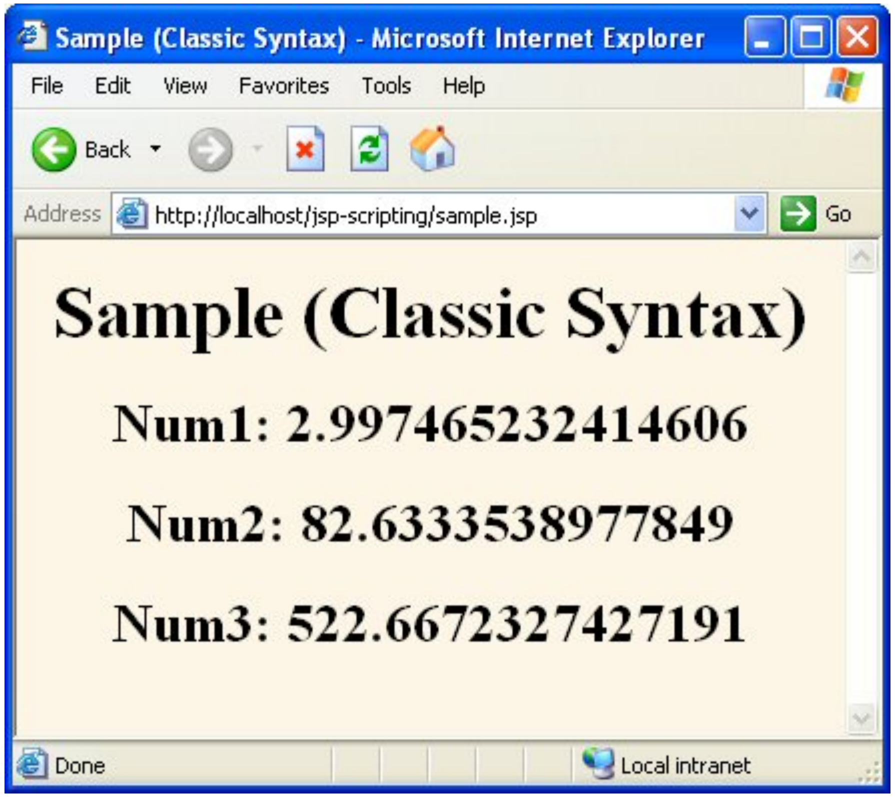
Sample HTML 4 Page: Classic Syntax (sample.jsp)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD ...">
<HTML>
<HEAD><TITLE>Sample (Classic Syntax)</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>Sample (Classic Syntax)</H1>

<H2>Num1: <%= Math.random()*10 %></H2>
<% double num2 = Math.random()*100; %>
<H2>Num2: <%= num2 %></H2>
<%! private double num3 = Math.random()*1000; %>
<H2>Num3: <%= num3 %></H2>

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

Sample XHTML Page: XML Syntax (sample.jspx)

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns:jsp="http://java.sun.com/JSP/Page">
<jsp:output
     omit-xml-declaration="true"
     doctype-root-element="html"
     doctype-public="-//W3C//DTD ..."
     doctype-system="http://www.w3.org...dtd" />
<jsp:directive.page contentType="text/html"/>
<head><title>Sample (XML Syntax)</title></head>
<body bgcolor="#fdf5e6">
<div align="center">
<h1>Sample (XML Syntax)</h1>
<h2>Num1: <jsp:expression>Math.random()*10</jsp:expression></h2>
<jsp:scriptlet>
double num2 = Math.random()*100;
</jsp:scriptlet>
<h2>Num2: <jsp:expression>num2</jsp:expression></h2>
<jsp:declaration>
private double num3 = Math.random()*1000;
</jsp:declaration>
<h2>Num3: <jsp:expression>num3</jsp:expression></h2>
</div></body></html>
```

Sample Pages: Results

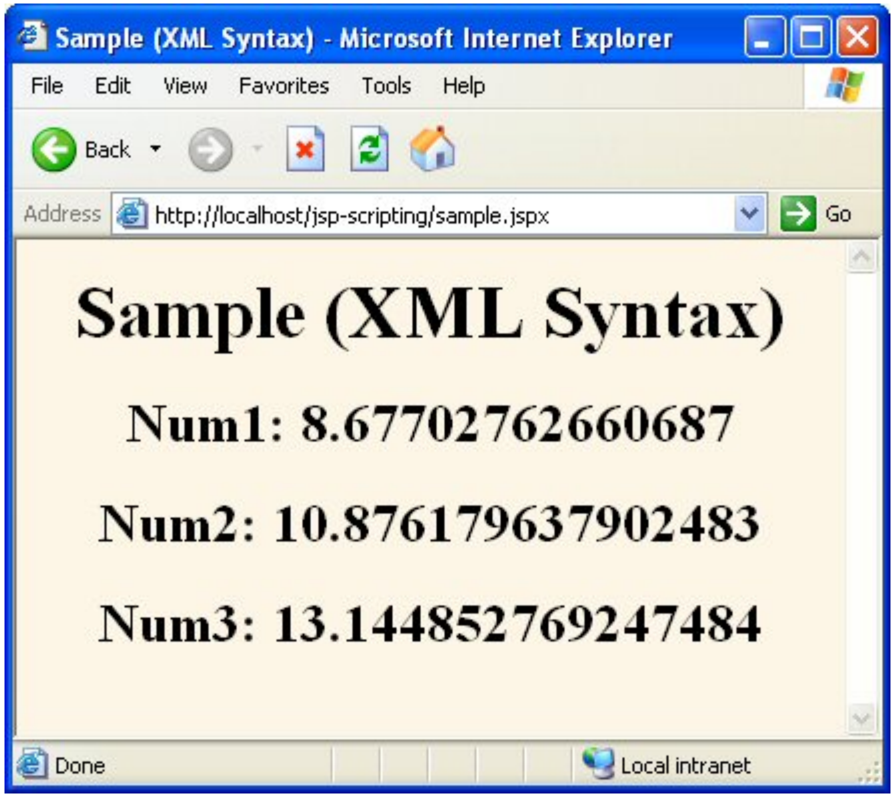

Sample (Classic Syntax) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address http://localhost/jsp-scripting/sample.jsp
Go
Done
Local intranet

Sample (Classic Syntax)

Num1: 2.997465232414606

Num2: 82.6333538977849

Num3: 522.6672327427191


Sample (XML Syntax) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address http://localhost/jsp-scripting/sample.jspx
Go
Done
Local intranet

Sample (XML Syntax)

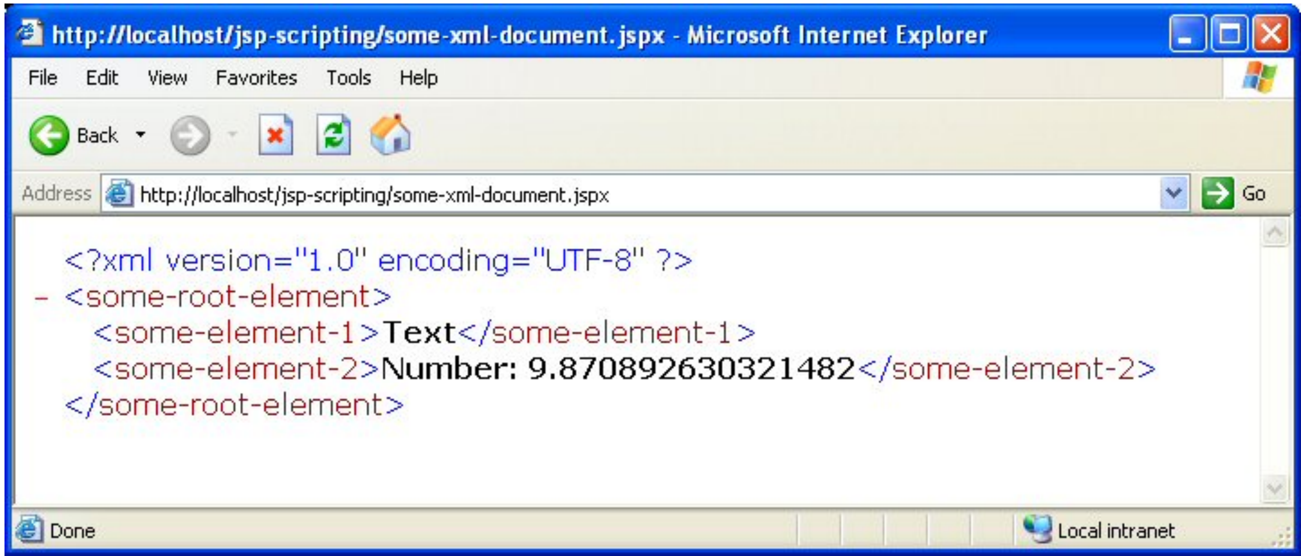
Num1: 8.67702762660687

Num2: 10.876179637902483

Num3: 13.144852769247484

XML Document Generated with XML Syntax

```
<?xml version="1.0" encoding="UTF-8" ?>
<some-root-element
    xmlns:jsp="http://java.sun.com/JSP/Page">
  <some-element-1>Text</some-element-1>
  <some-element-2>
    Number:
    <jsp:expression>Math.random()*10</jsp:expression>
  </some-element-2>
</some-root-element>
```


http://localhost/jsp-scripting/some-xml-document.jspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address http://localhost/jsp-scripting/some-xml-document.jspx
Go
Done
Local intranet

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <some-root-element>
    <some-element-1>Text</some-element-1>
    <some-element-2>Number: 9.870892630321482</some-element-2>
  </some-root-element>
```

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, servlet/JSP combo (MVC), JSP expression language, custom tags
- **XML Syntax**
  - There is alternative JSP syntax that is sometimes useful when generating XML-compliant documents, probably for Ajax apps.
    - But is more trouble than it is worth for most HTML applications

Questions?