Lecture 10
Invoking Java Code with JSP Scripting Elements

Lecture Agenda

1. Static vs Dynamic data
2. Dynamic code and good JSP design
3. JSP Expressions
4. Servlet vs JSP pages for similar tasks
5. JSP scriptlets
6. JSP declarations
7. Predefined variables
8. Comparison of expressions, scriptlets and declarations

JSP Scripting Introduction

Uses of JSP

Main uses of JSP

Simple Application

Complex Application

- Scripting elements call servlet code directly
- Scripting elements calling servlet code indirectly (utility classes)
- Beans
- Servlet/JSP combo (MVC)
- MVC with JSP expression language (EL)
- Custom JSPs (Tag Library)
- MVC with beans, custom tags, and a framework like JSF 2.0

Design Strategy

Limit Java code in JSP pages

- You have two options
  1. Put 25 lines of java code directly in the JSP page
  2. Put this 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
  - Debugging: You can debug and set break points in java code (not JSP).
  - Testing: Easier to test Java code in Java class.
  - Reuse: You can reuse the same class for multiple pages.

Basic Syntax

Element	Explanation
HTML Text	• <H1>Blah</H1> • Passed through to client as normal. • HTML in JSP is really translated as … out.println(“<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 <\% … (note \ to escape)

Design Strategy

Limit Java code in JSP pages

- Expressions
  - Format <%= expression %>
  - Evaluated and inserted into servlet's output.
  - ie: 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

JSP Expressions: <%= value %>

JSP Expressions

Expressions

- Format
  - <%= Java Expression %>
- Result
  - Expression evaluated, converted to String and placed into HTML page at the place it occurred in JSP page.
- Examples
  - Current time: <%= new java.util.Date() %>
  - Your hostname: <%= request.getRemoteHost() %>

JSP/Servlet Correspondance

JSP Translation

- Original JSP

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

- Representative resulting **translated** 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());
      ……
}
```

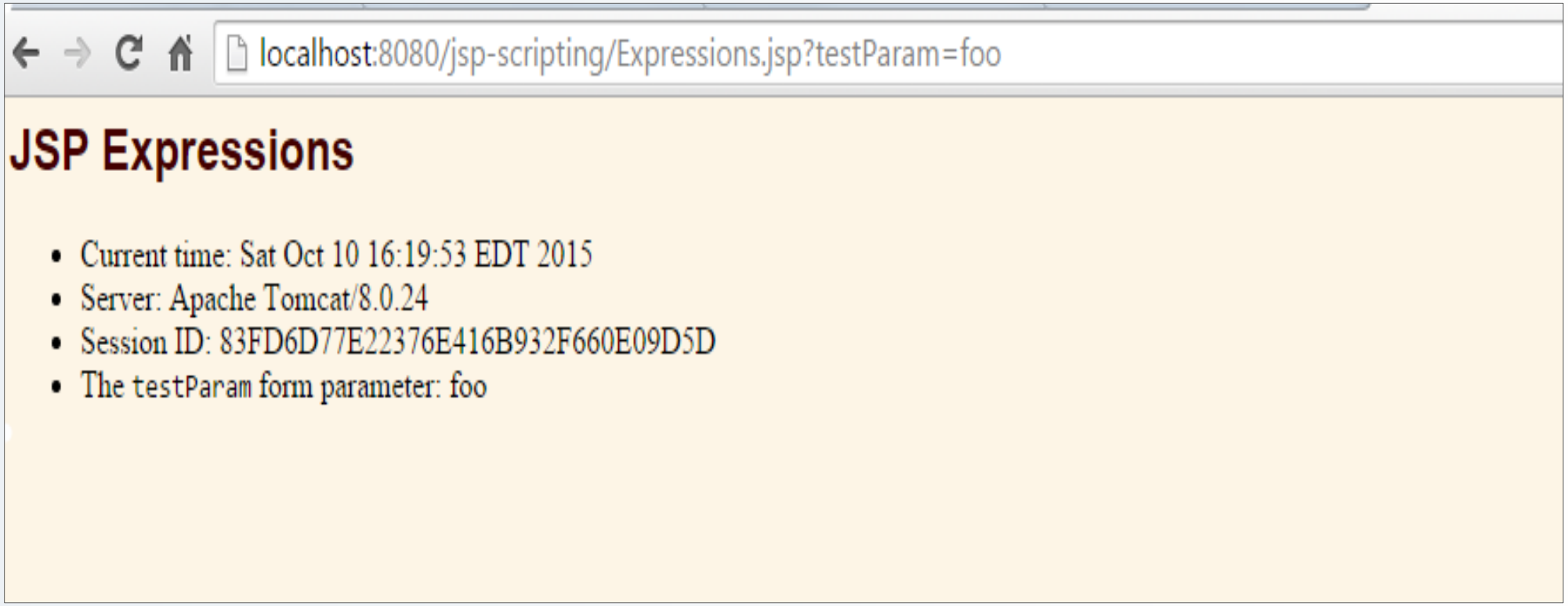
JSP Expression: Example

JSP Example

```
<body>
    <h2>JSP Expression</h2>
     <ul>
       <li>Current Time: <%= new java.util.Date() %><li>
       <li>Server: <%= application.getServerInfo() %><li>
       <li>Session ID: <%= session.getId() %><li>
        <li>The <code>testParam</code> from parameter:
                                <%= request.getParameter("testParam") %><li>
    </ul>
<body>
```

JSP Expression: Example

JSP Example



Predefined variables in JSP page

Predefined Variable	Information
request	• The HttpServletRequest (1st argument to doGet/doPost)
response	• The HttpServletResponse (2nd argument to doGet/doPost)
out	• The Writer used to send output to client
session	• The HttpSession associated with the request

application	• The servletContext as obtained from getServletContext()
	• servletContext is created by web container at time of project deployment
	• Can be used to get config info from web.xml
	• There is only **one** servletContext per web application.
	
	//We can get the ServletContext object from ServletConfig object
	ServletContext application=getServletConfig().getServletContext();
	
	//Another convenient way to get the ServletContext object
	ServletContext application=getServletContext();

Comparing Servlet to JSPs

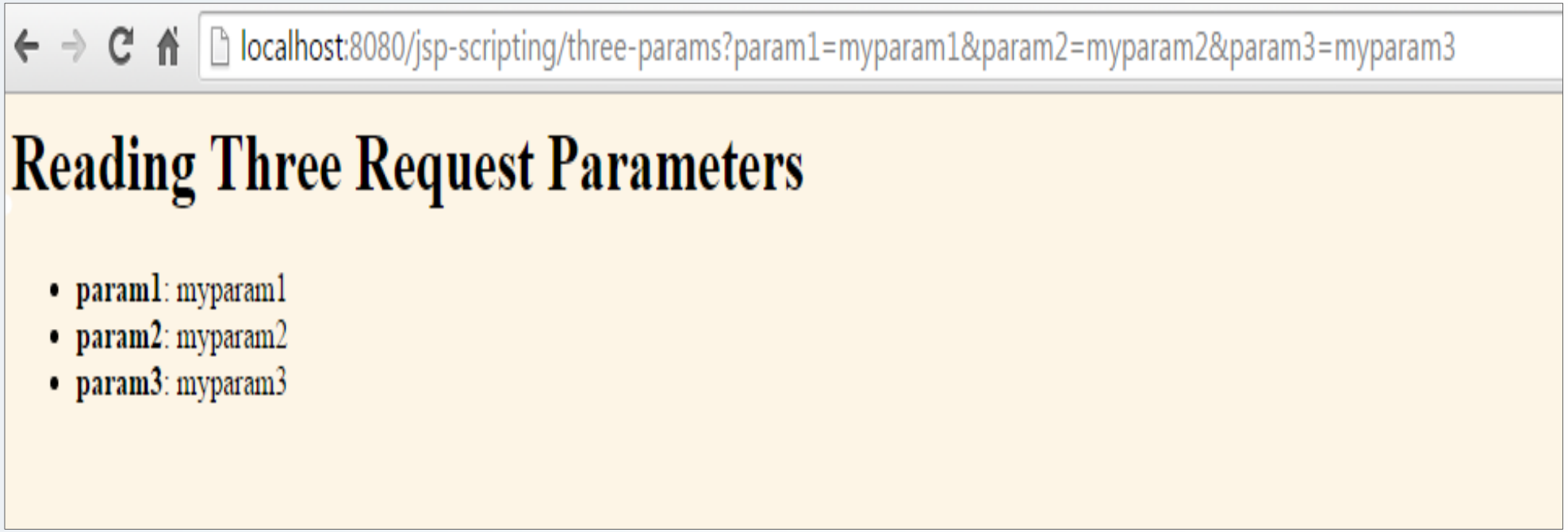
Comparing Servlet to JSPs

Code Snippet: Servlet Version

```
@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>");
  }
}
```

Comparing Servlet to JSPs

Servlet Example



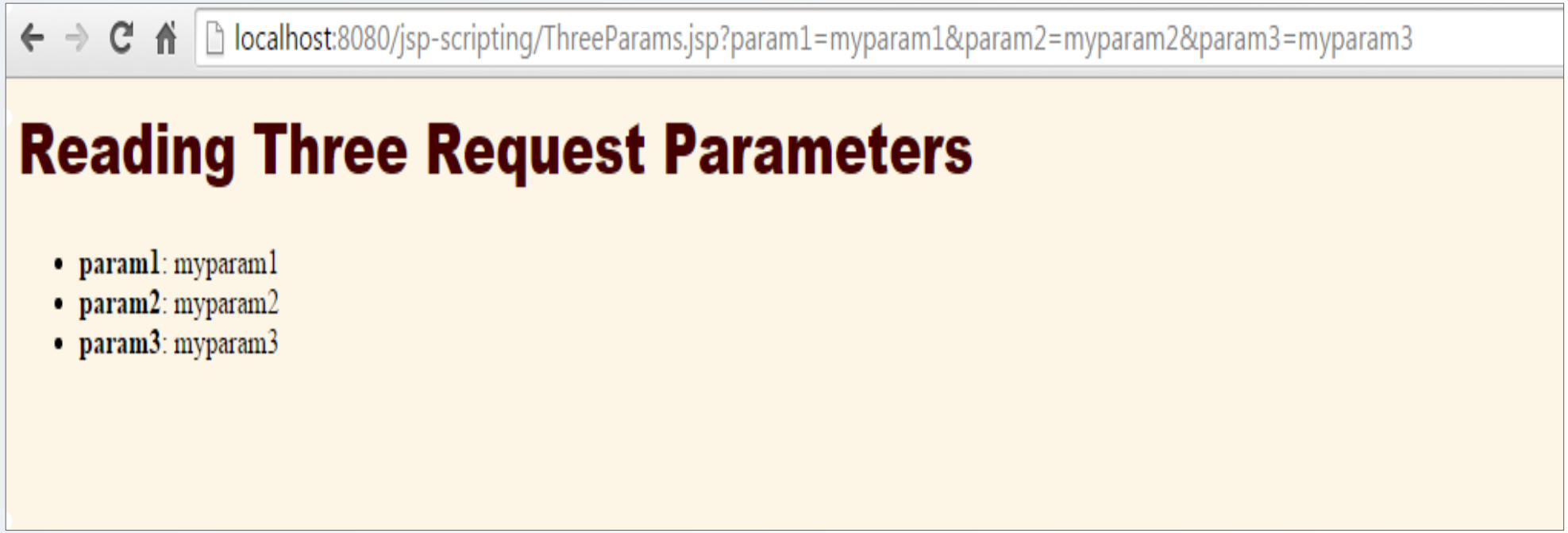
Comparing Servlet to JSPs

Code Snippet: JSP Version

```
<!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>
```

Comparing Servlet to JSPs

Servlet Example



JSP Scriptlets Code: <% Code %>

JSP Expressions

Expressions

- Format

  <%= Java Code %>

- Result
  - Code is inserted verbatim into servlets _jspService.
- Example
  - <% String queryData = request.getQueryString(); %>
  - Attached GET data: <%= queryData %>
  - <% response.setContentType("text/plain"); %>

JSP/Servlet Correspondence

Expressions

- Original JSP
  - <H2>foo</H2>
  - <%= bar() %>
  - <% baz(); %>
- Representative resulting 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();
```

Note this **does not** result in print output to client screen

JSP Scriptlets Example

Question

What’s wrong with the following code?

- Suppose you want to let end-users customize the background color of a page. Determine what may be wrong with the following code?

```
<body bgcolor= “<%= request.getParameter(“bgColor”)%>” >
```

Answer

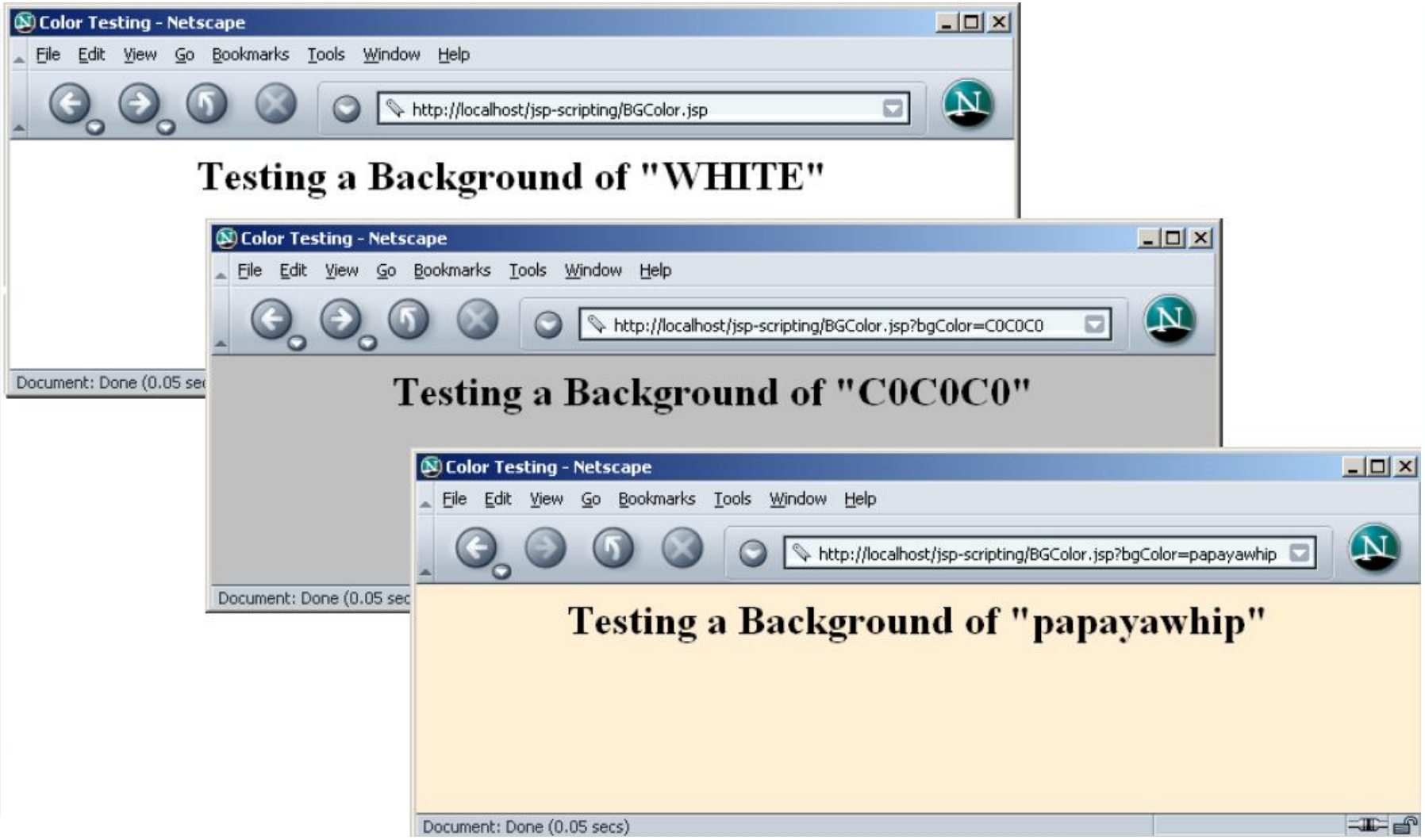
Testing for empty or null value

```
<!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>
```

Code does not test for null or empty value. Logic is required to set default accordingly.

Result

Setting Background Colour



JSP/Scriptlets Conditionals

JSP/Scriptlets Conditionals

JSP and Conditionals

- Background
  - Scriptlets are inserted into servlet **exactly** as written
  - Need not be complete Java expressions
  - Complete expressions are usually clearer and easier to maintain

- JSP Code Example

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

Blend of Java and HTML

JSP/Scriptlets Conditionals Continued …

JSP and Conditionals

- Java Translated 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

JSP Declarations

- Format

  <%! Java Code %>

- Result
  - Code is inserted verbatim into servlets class definition, outside of any existing methods.
- Example
  - <%! 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.

JSP/Servlet Correspondence Continued …

JSP Declarations

- Original JSP

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

- Better Design Alternative
  - Make randomHeading a static method in a separate class rather that declaring method in JSP directly.

JSP/Servlet Correspondence Continued …

JSP Declarations

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

Resulting code, creates generation of separate class and _jspService calling declared method

JSP Declarations: Example

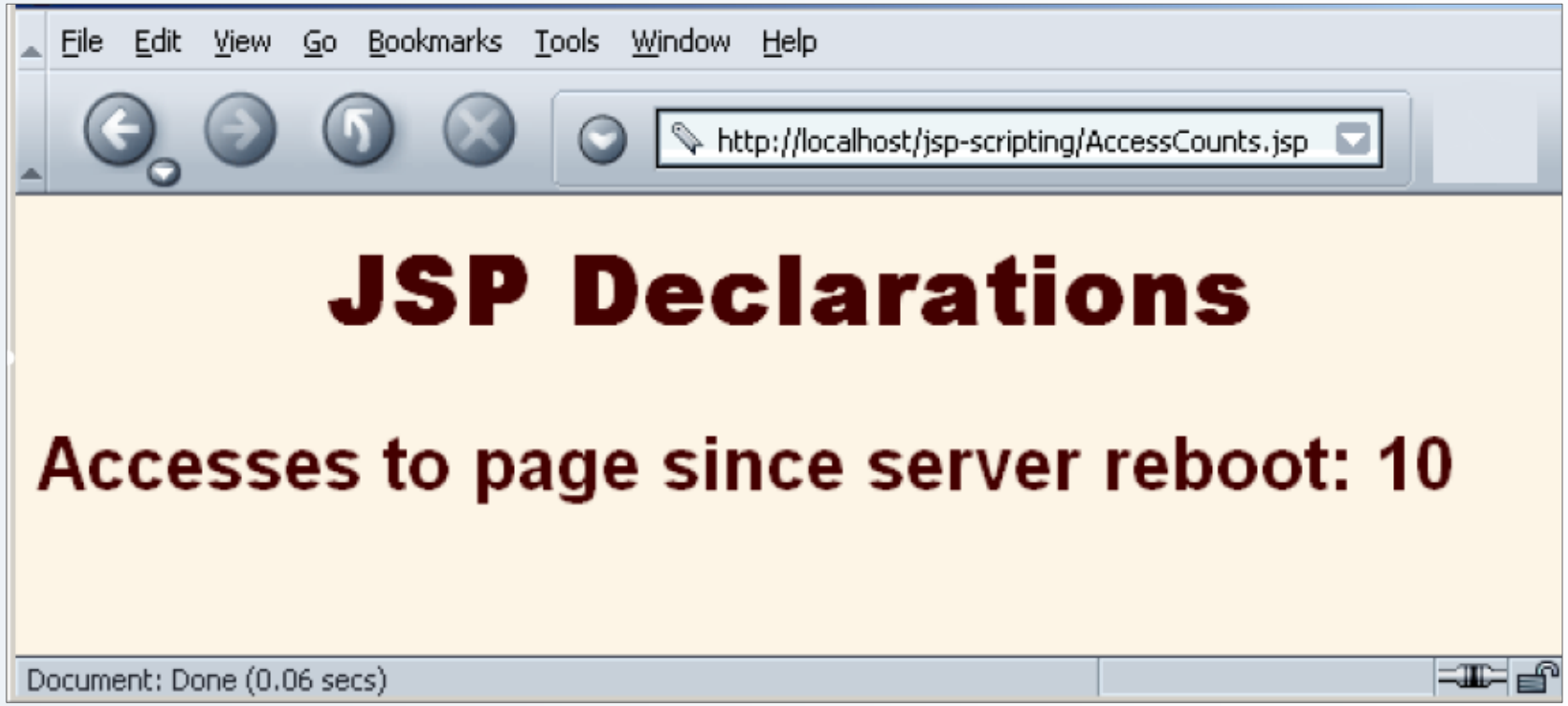
JSP Declarations

```
<!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 Declaration and JSP Value Tags

JSP Declarations: Result

JSP Declarations Result



Comparing JSP Scripting Elements

Using JSP Expressions, JSP Scriptlets and JSP Declarations

Expressions, Scriptlets and Declarations

- Task 1
  - Output a bulleted list of five random ints from 1 to 10.
    - Since the structure of the page is **fixed** and we should use a separate helper class (ex: for the a 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. JSP declarations work like static attributes between different client requests. Variable saves its state between differing requests.
    - Use **JSP declarations** for this.

Helper Class

Create Helper Utility Class

```
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));
}}}
```

Helper class, offering static **randomInt** helper method

Task 1: JSP Expressions – Fixed List

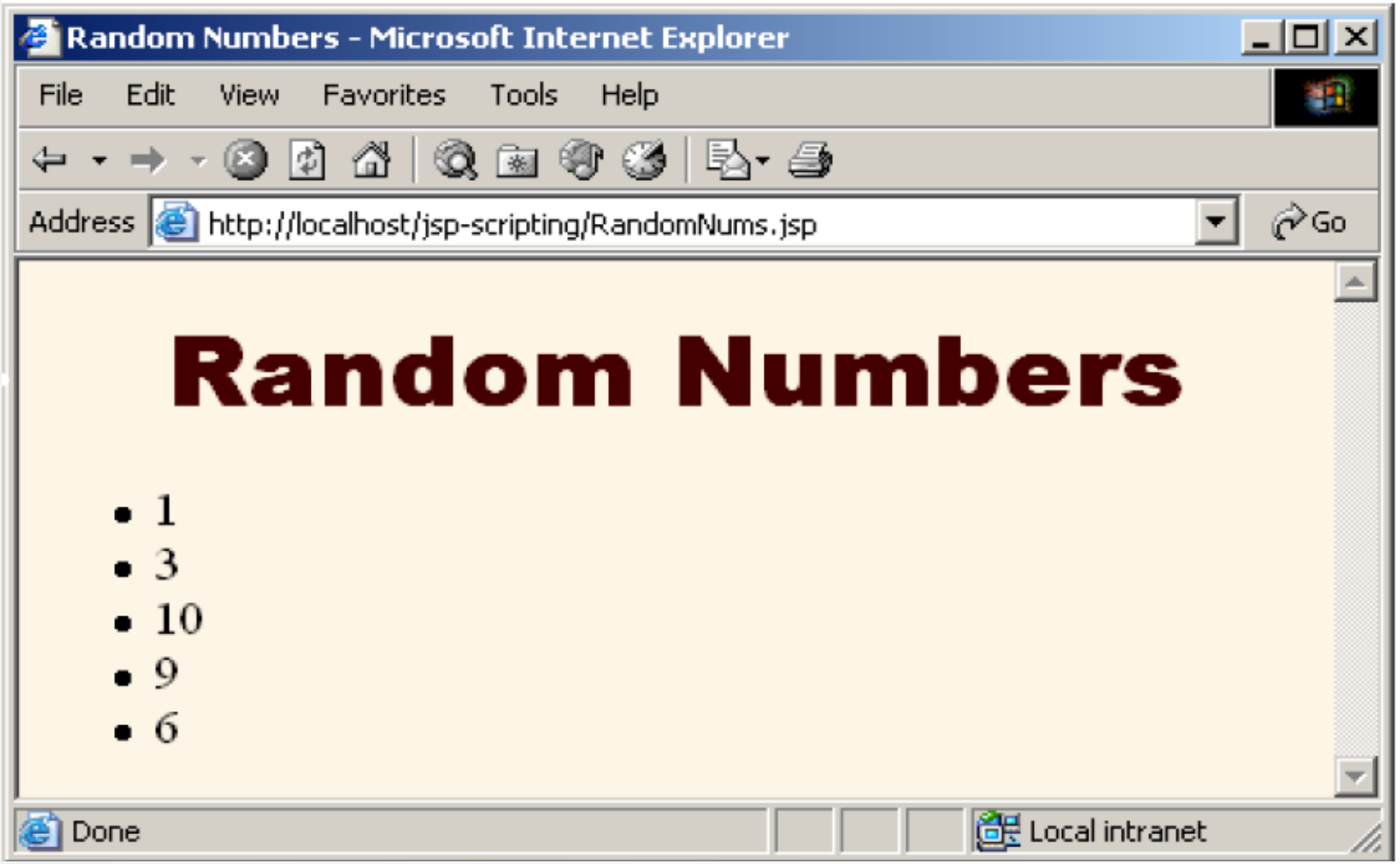
Task 1: Code

```
<!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>
```

JSP Expression
Tags calling static
randomInt
method

Task 1: JSP Expressions

Result



Task 2: JSP Scriptlets – Dynamic List

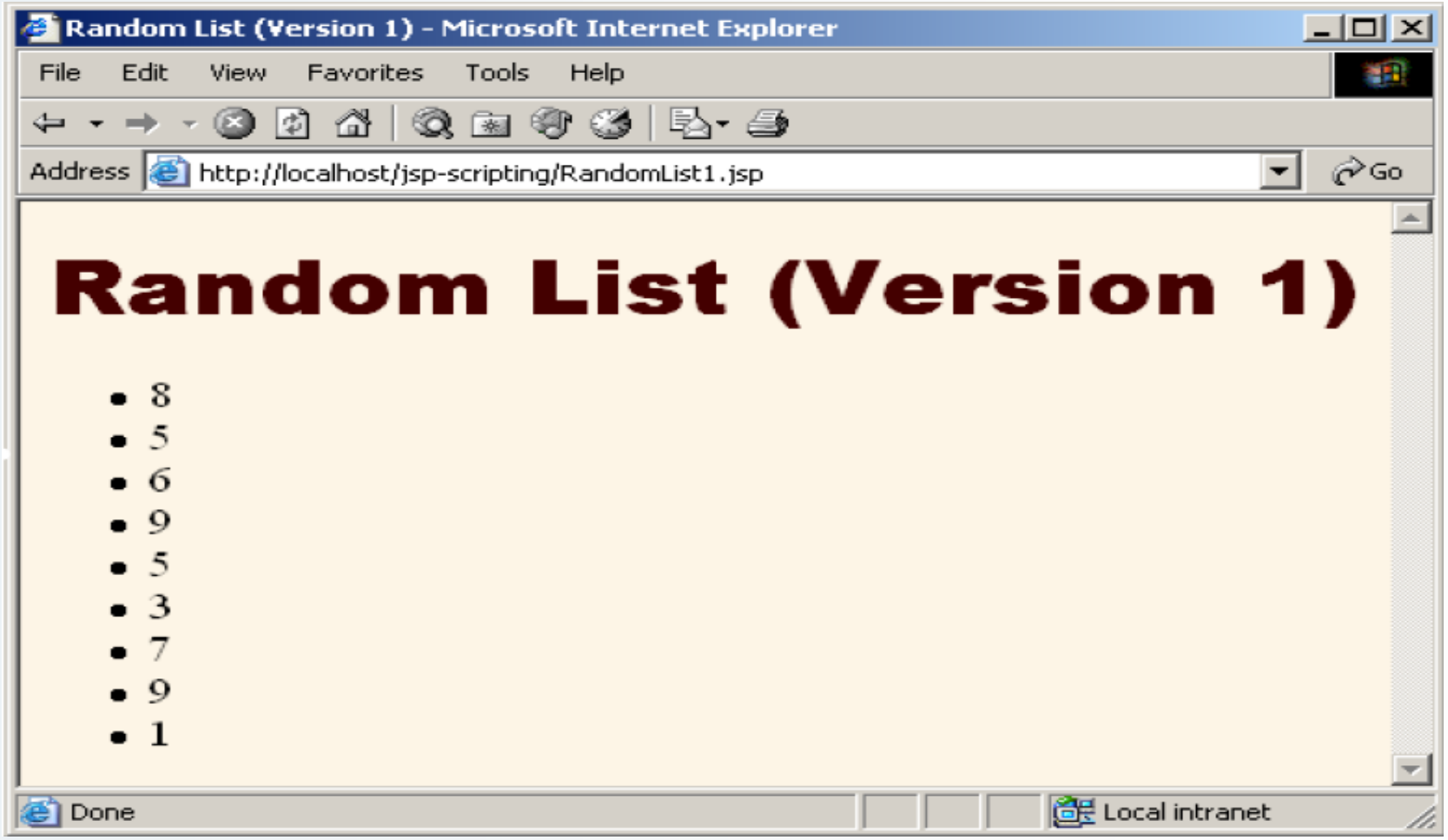
Code: Version 1 or 2

```
<!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>
```

JSP Scriptlets called. Dynamic list printed to end-user

Task 2: JSP Scriptlets

Result: Version 1



Task 2: JSP Scriptlets – Dynamic List

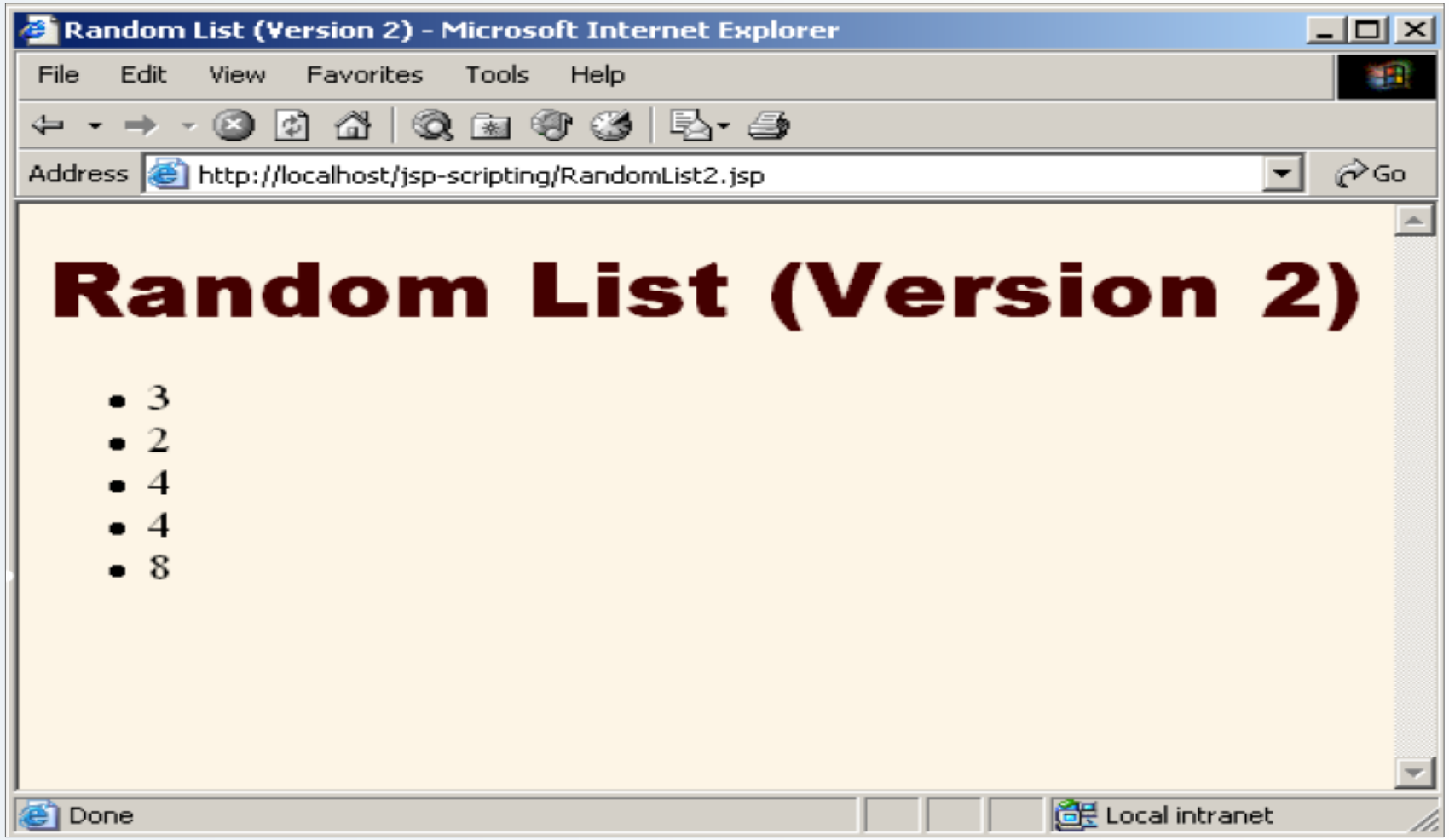
Code: Version 2 of 2

```
<!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>
```

Notice where the client output is generated (within html li element)

Task 2: JSP Scriptlets

Result: Version 2



Task 3: JSP Declarations

Code

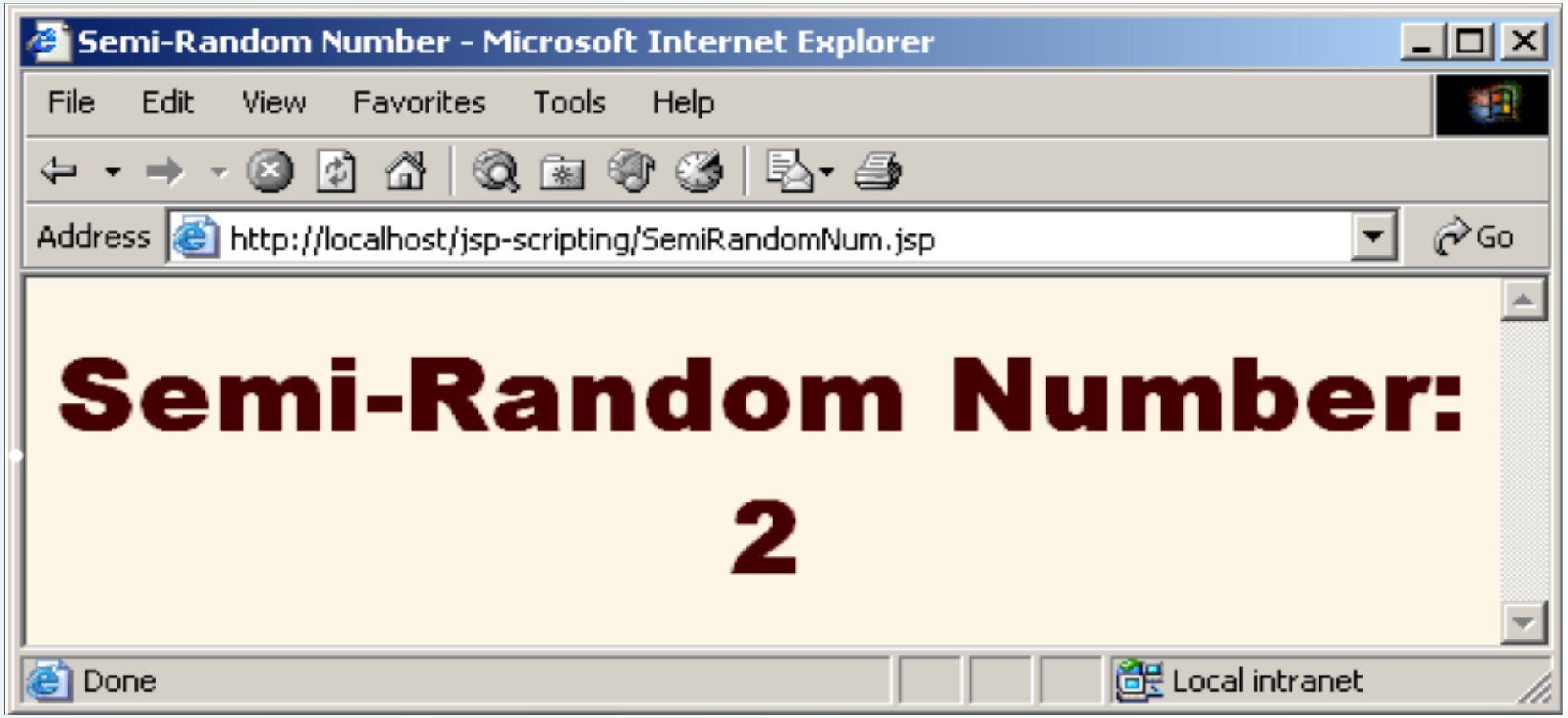
```
<!DOCTYPE …>
<HTML>
<HEAD>
<TITLE>Semi-Random Number</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<%!
private int randomNum =
  coreservlets.RanUtilities.randomInt(10);
%>
<H1>Semi-Random Number:<BR><%= randomNum %></H1>
</BODY>
</HTML>
```

<%! JSP declarations work like static attribute.

Declare **randomNum** and utilize in JSP declaration and expression tag

Task 3: JSP Declarations

Result



Summary

JSP Expressions, JSP Scripltets and JSP Declarations Summary

- JSP Expressions
  - Format: <%= expression %>
  - Wrapped in out.println and inserted into _jspService
- JSP Scriptlets
  - Format: <% code %>
  - Inserted verbatim into the servlets _jspService method
- JSP Declarations
  - Format: <%! code %>
  - Inserted verbatim into the body of the servlet class
- Predefined variables
  - request, response, out, session, application

Summary

Best Practice

- **Remember!!**
  - Limit the Java code that is directly in the jsp page.
  - Use helper classes, beans etc …

Questions?