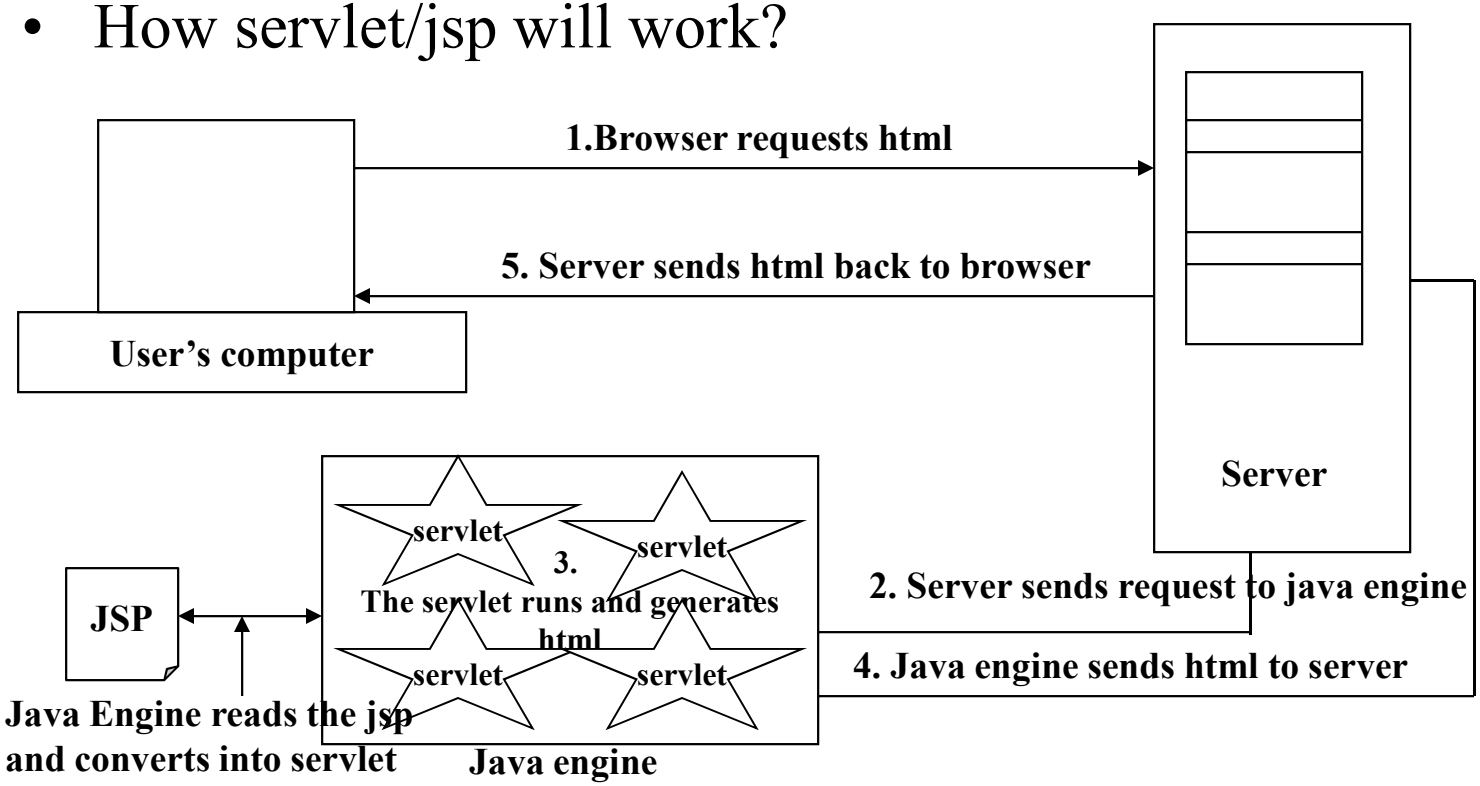
JSP : Java Server Pages

- **We need JSP to take care of the Dynamic Content and Personalization**
- **We need the web designer to focus on the design of the web page and the programmer to focus on the application logic**
- **Whatever you can do with Servlet you can do with JSP**
- **JSP is a way to automate the process of generating responses in HTML format.**
- **If we didn't have JSP, we had to craft the reply ourselves in Servlet.**
- **Servlets are good whenever you have heavy-weight computation that requires "real programming"**

Servlets and Java Server pages

- How servlet/jsp will work?



Simple JSPs

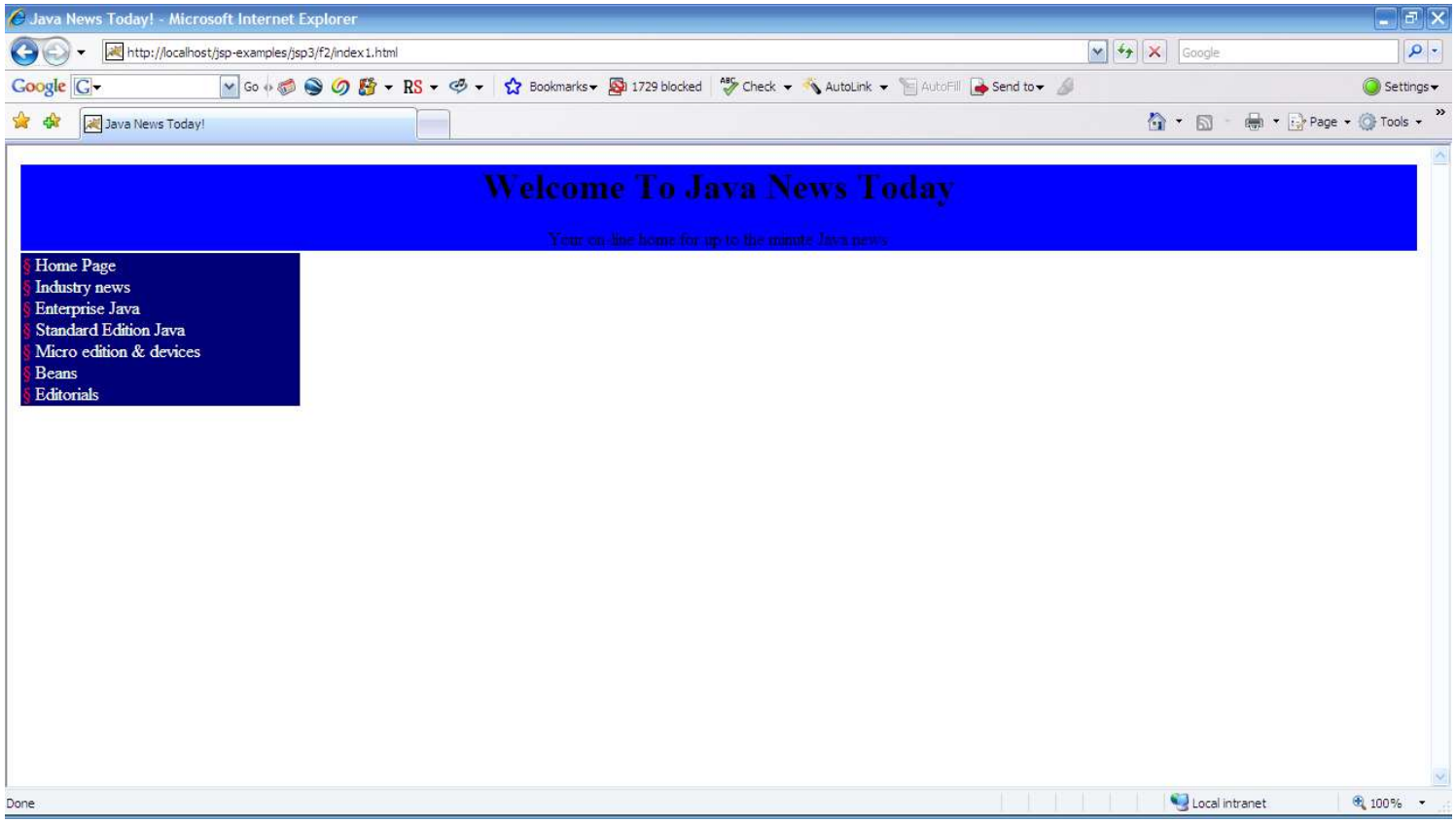
- http://localhost/jsp-examples/jsp3/f2/index1.jsp

Simple JSPs

- In the Browser type:
  - http://localhost/jsp-examples/jsp3/f2/index1.jsp
- Normally you would name it as index1.html. However this time you save it as index1.jsp and place it in your jsp directory of tomcat and point your browser to
  - http://localhost/jsp-examples/jsp3/f2/index1.jsp
- If this were a html page the web server simply reads the contents and sent them to the browser
  - http://localhost/jsp-examples/jsp3/f2/index1.html

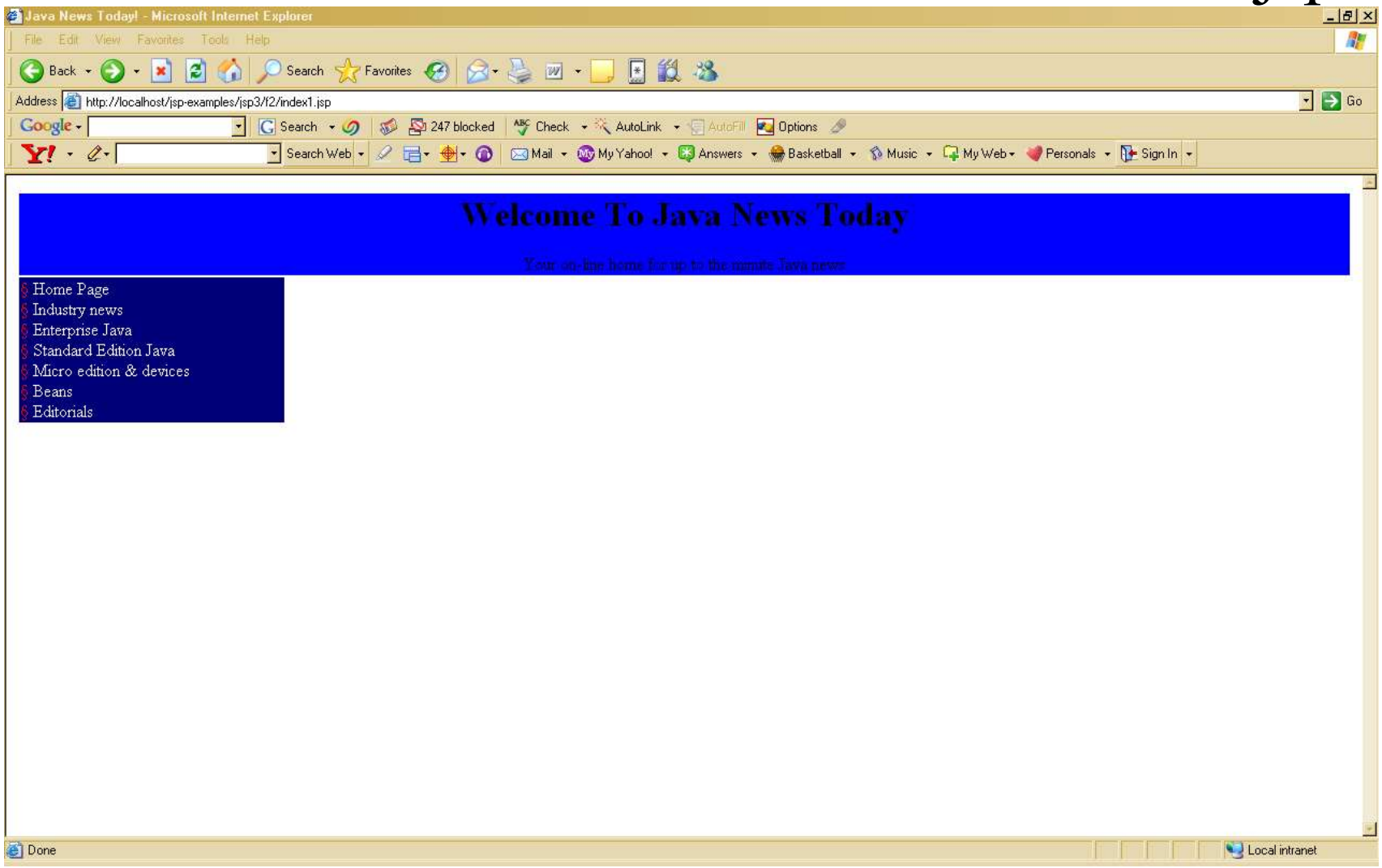
Simple JSPs

- Here is how it looks like in the browser as index1.html



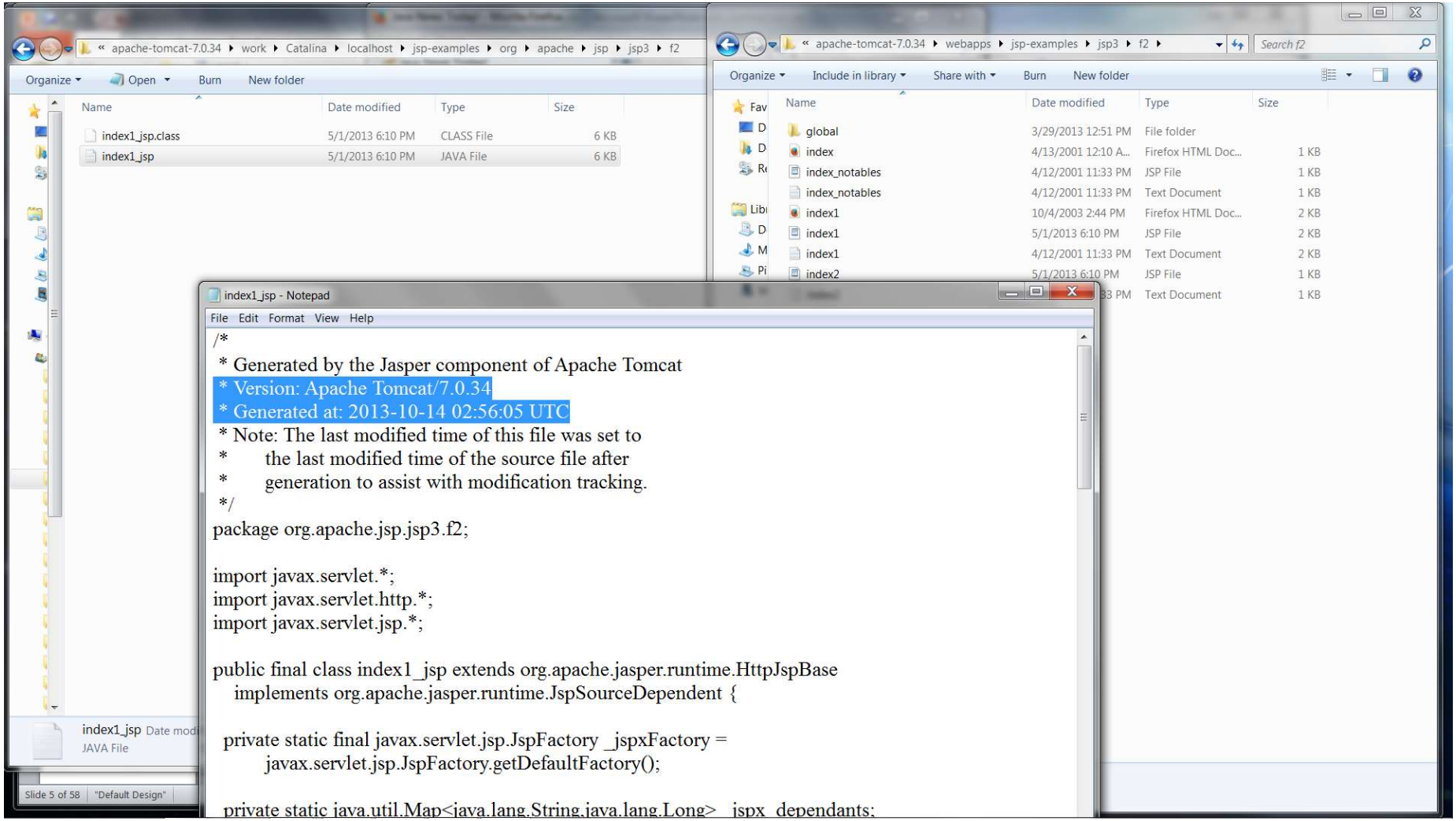
Simple JSPs

- Here is how it looks like in the browser as a jsp



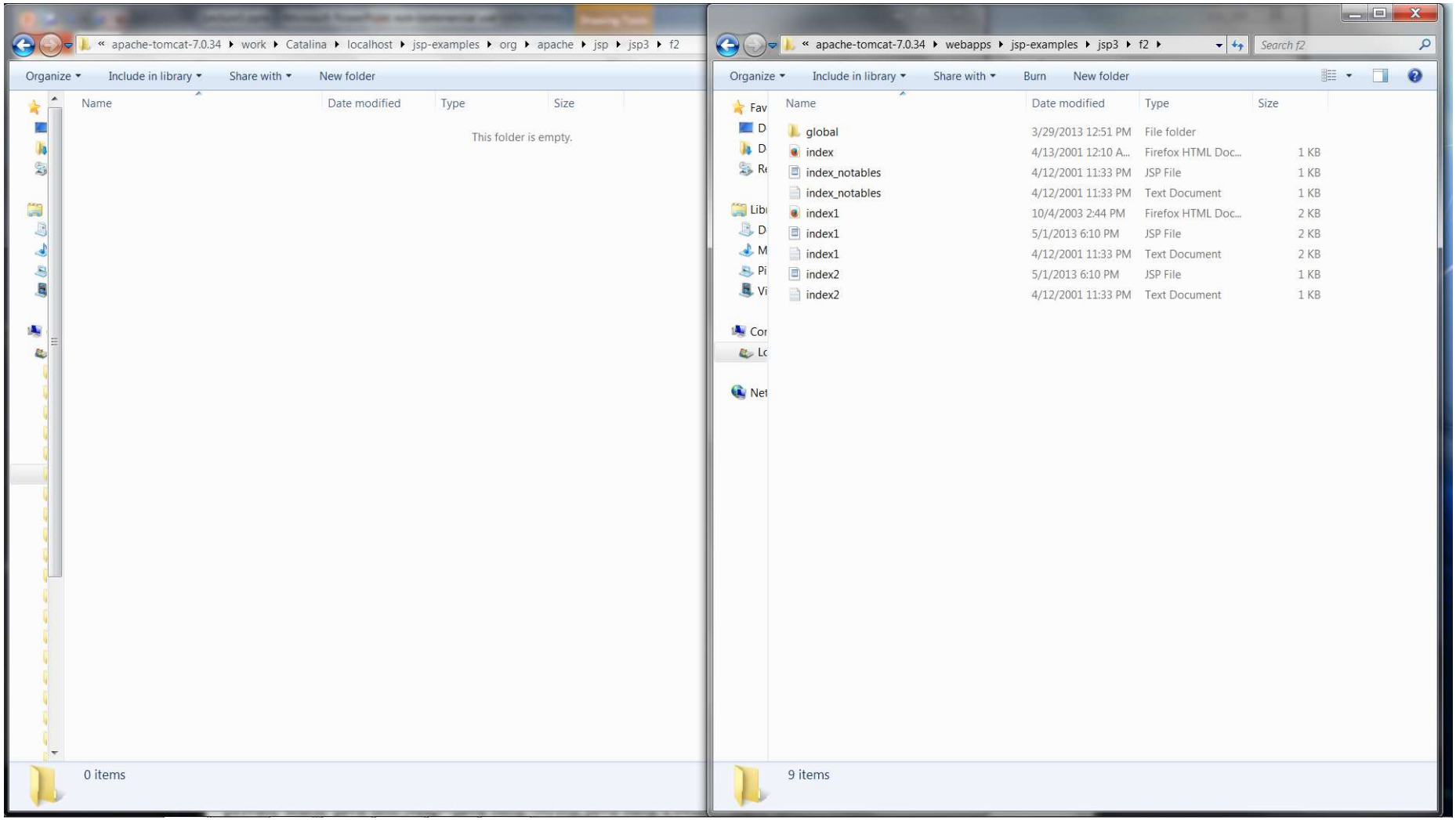
Simple JSPs

- See the Code generated for the JSP



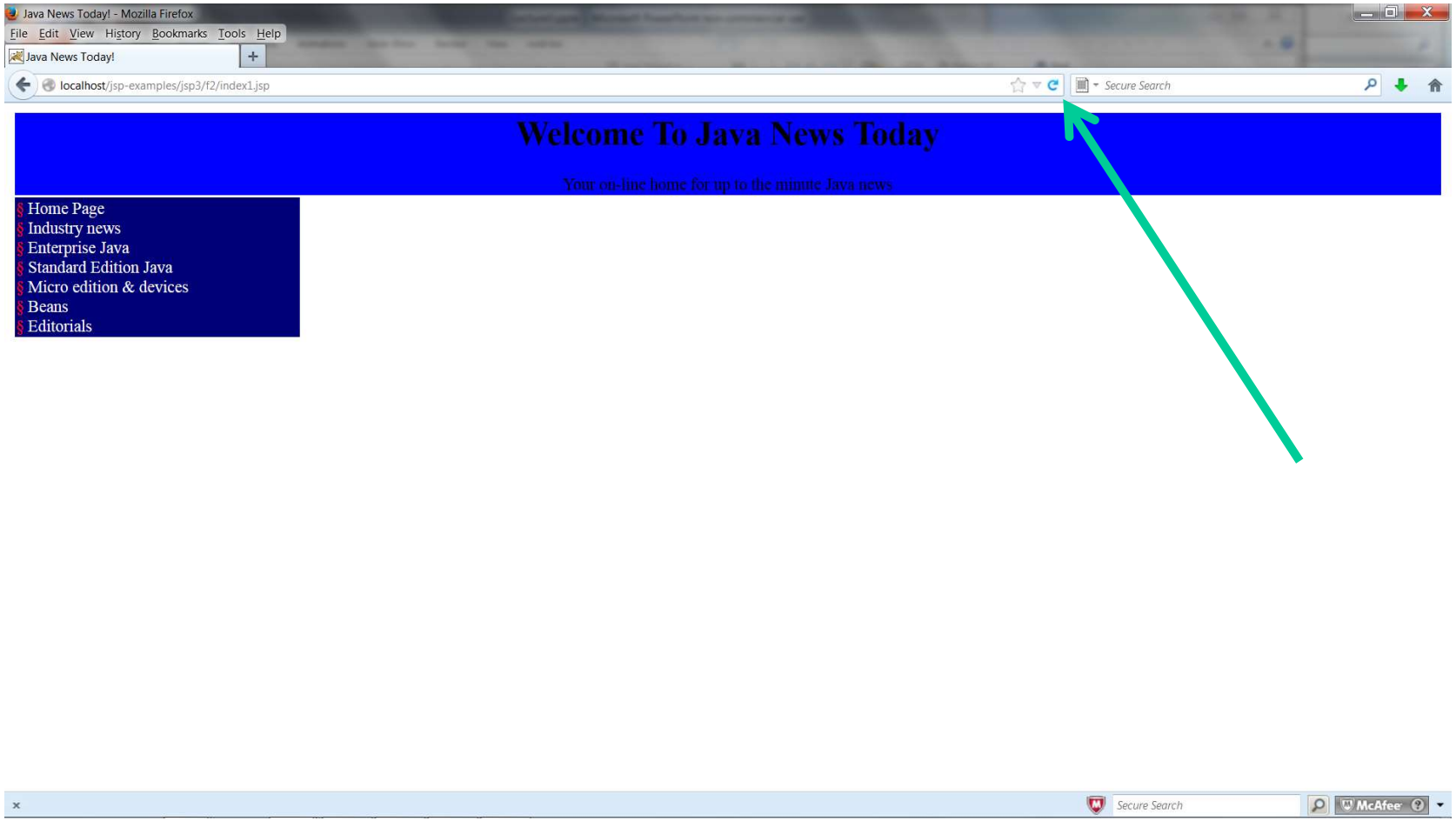
Simple JSPs

- Delete the Code generated for the JSP manually



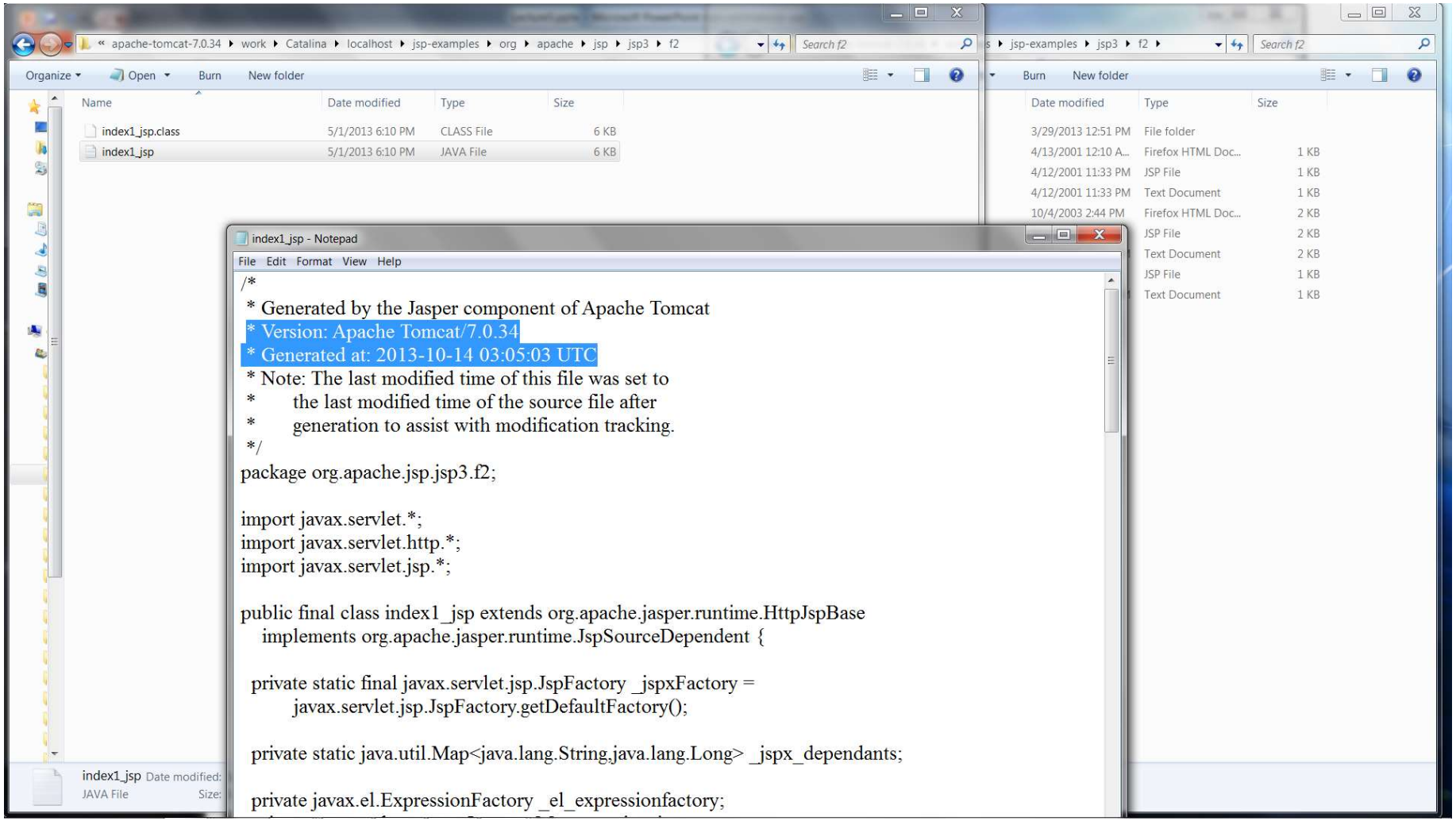
Simple JSPs

- Reload the browser with index1.jsp to RE-Generate the Code



Simple JSPs

- See the Code generated for the JSP AGAIN



Simple JSPs

- See the Code generated for the JSP AGAIN

index1_jsp - Notepad

File Edit Format View Help

```
public void _jspInit() {
  _el_expressionfactory = _jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();
  _jsp_instancemanager = org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServletConfig());
}

public void _jspDestroy() {
}

public void _jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException, javax.servlet.ServletException {

  final javax.servlet.jsp.PageContext pageContext;
  javax.servlet.http.HttpSession session = null;
  final javax.servlet.ServletContext application;
  final javax.servlet.ServletConfig config;
  javax.servlet.jsp.JspWriter out = null;
  final java.lang.Object page = this;
  javax.servlet.jsp.JspWriter _jspx_out = null;
  javax.servlet.jsp.PageContext _jspx_page_context = null;


  try {
    response.setContentType("text/html");
    pageContext = _jspxFactory.getPageContext(this, request, response,
                                  null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
```

Ln 14, Col 1

Official documentation for the JSP life cycle

The Life Cycle of a JSP Page - The Java EE 5 Tutorial - Mozilla Firefox

File Edit View History Bookmarks Tools Help

The Life Cycle of a JSP Page - The Java E...

docs.oracle.com/javaee/5/tutorial/doc/bnahe.html

Deactivating Expression Evaluation
Literal Expressions
Resolving Expressions
Process of Expression Evaluation
EL Resolvers
Implicit Objects
Operators
Reserved Words
Examples of EL Expressions
Functions
Using Functions
Defining Functions
JavaBeans Components
JavaBeans Component Design Conventions
Creating and Using a JavaBeans Component
Setting JavaBeans Component Properties
Retrieving JavaBeans Component Properties
Using Custom Tags
Declaring Tag Libraries
Including the Tag Library Implementation
Reusing Content in JSP Pages
Transferring Control to Another Web Component
`jsp:param` Element
Including an Applet
Setting Properties for Groups of JSP Pages
Deactivating EL Expression Evaluation
Declaring Page Encodings
Defining Implicit Includes
Eliminating Extra White Space

- `jsp:[include|forward]` elements are converted into invocations of the Java Servlet API.
- The `jsp:plugin` element is converted into browser-specific markup for activating an applet.
- Custom tags are converted into calls to the tag handler that implements the custom tag.

In the Application Server, the source for the servlet created from a JSP page named *pageName* is in this file:

domain-dir`/generated/jsp/j2ee-modules/`*WAR-NAME*`/`*pageName*`_jsp.java`

For example, the source for the index page (named `index.jsp`) for the `date` localization example discussed at the beginning of the chapter would be named:

domain-dir`/generated/jsp/j2ee-modules/date/index_jsp.java`

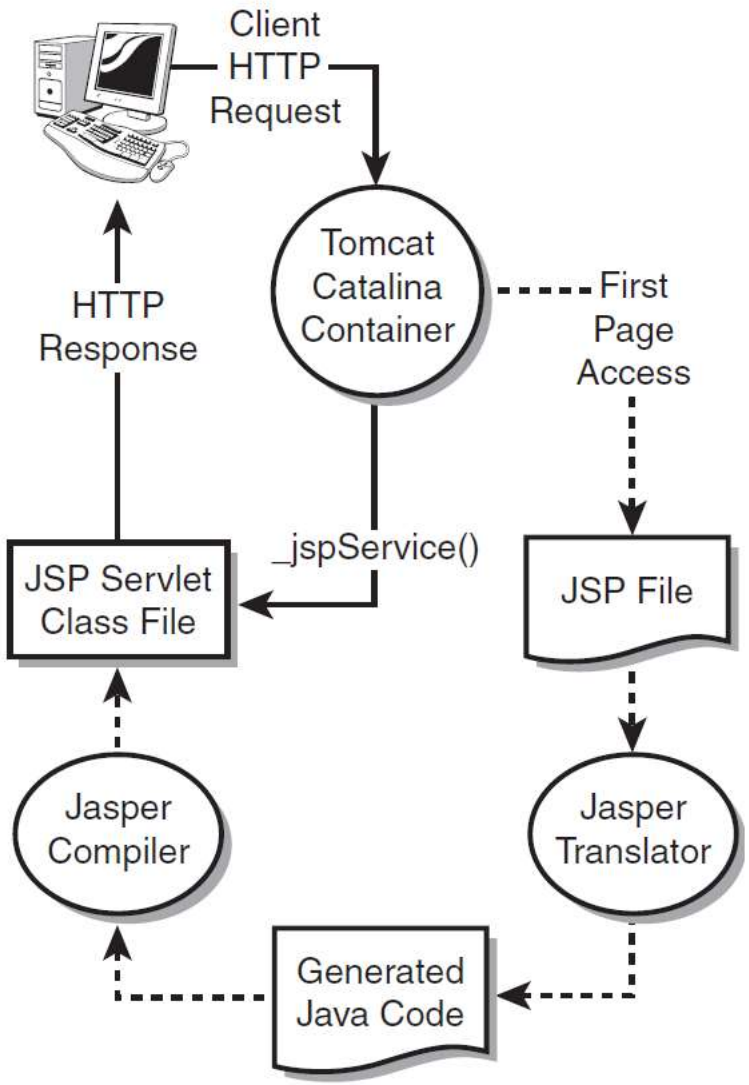
Both the translation and the compilation phases can yield errors that are observed only when the page is requested for the first time. If an error is encountered during either phase, the server will return `JasperException` and a message that includes the name of the JSP page and the line where the error occurred.

After the page has been translated and compiled, the JSP page's servlet (for the most part) follows the servlet life cycle described in Servlet Life Cycle:

1. If an instance of the JSP page's servlet does not exist, the container:
   a. Loads the JSP page's servlet class
   b. Instantiates an instance of the servlet class
   c. Initializes the servlet instance by calling the `jspInit` method
2. The container invokes the `_jspService` method, passing request and response objects.

If the container needs to remove the JSP page's servlet, it calls the `jspDestroy` method.

Tomcat - JSP life cycle



Simple JSPs

- But since this is a jsp page the web server asks the jsp engine for this page.
- The jsp engine does not directly send the contents, but it converts the jsp into a servlet.
- It then sends a request to the servlet, and the servlet generates a response, which includes the page text, and sends it back to the browser.

Simple JSPs

- Consider the above example.
- We have used <!--…--> for comments. If we use these comments and if the user sees the resulting jsp by view source command then he can see the command.
- The jsp has introduced a special tags for comments <%--….--%>. When the jsp engine encounters this it will not send this back to the user.

Simple JSPs

- In the above example. We have placed code between two comments to separate TWO sections, Header and Navigation:
  - <!-- Begin Header -->
  - <!-- End Header -->
  - <!-- Begin Navigation -->
  - <!-- End Navigation -->

- Instead we can use %@include file="filename.jsp".

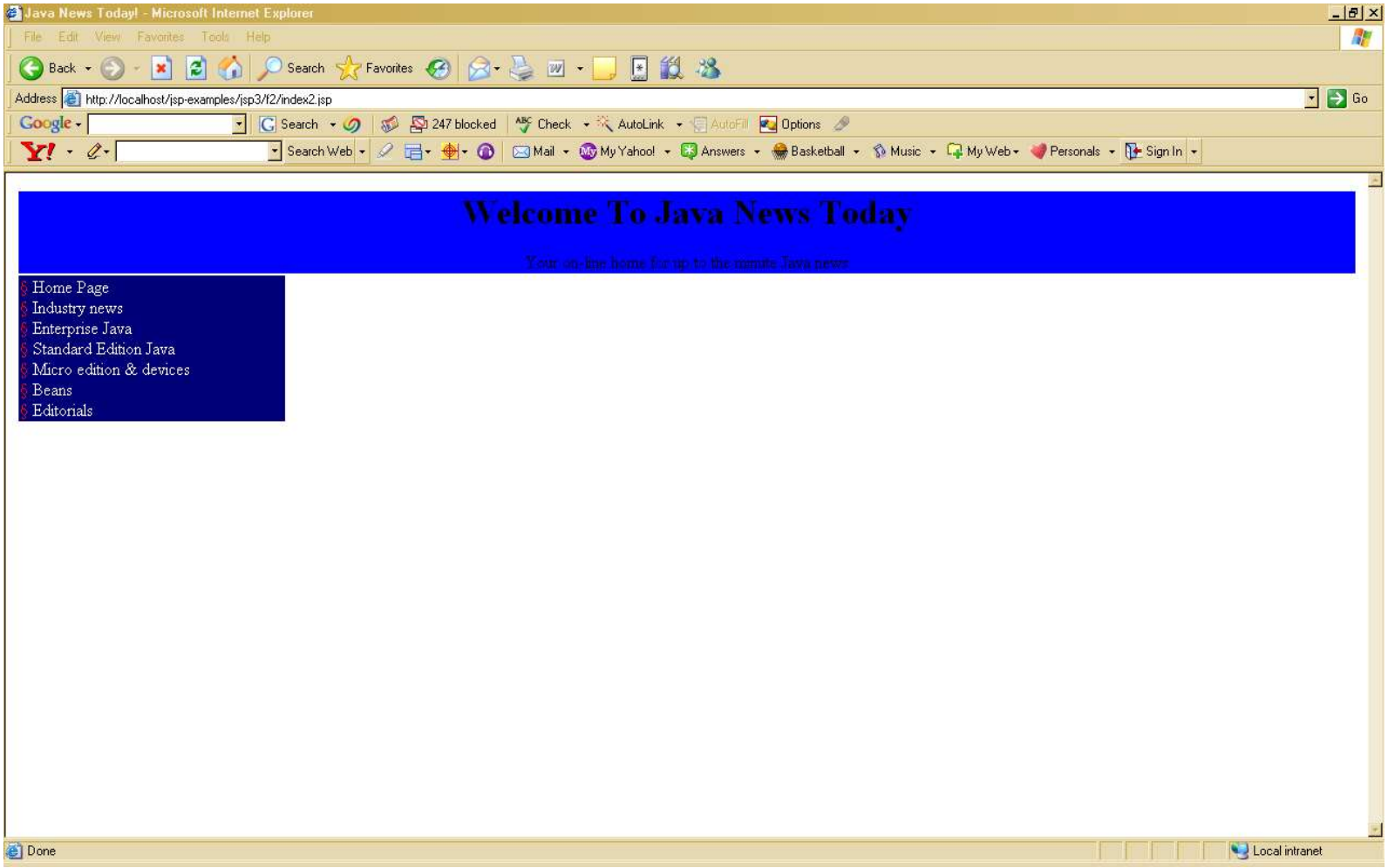
Simple JSPs

- Let us see the following example how to do **Templating** through the use of **include directive**

- Example :
  - http://localhost/jsp-examples/jsp3/f2/index2.jsp

Simple JSPs

- Here is the output …



Simple JSPs

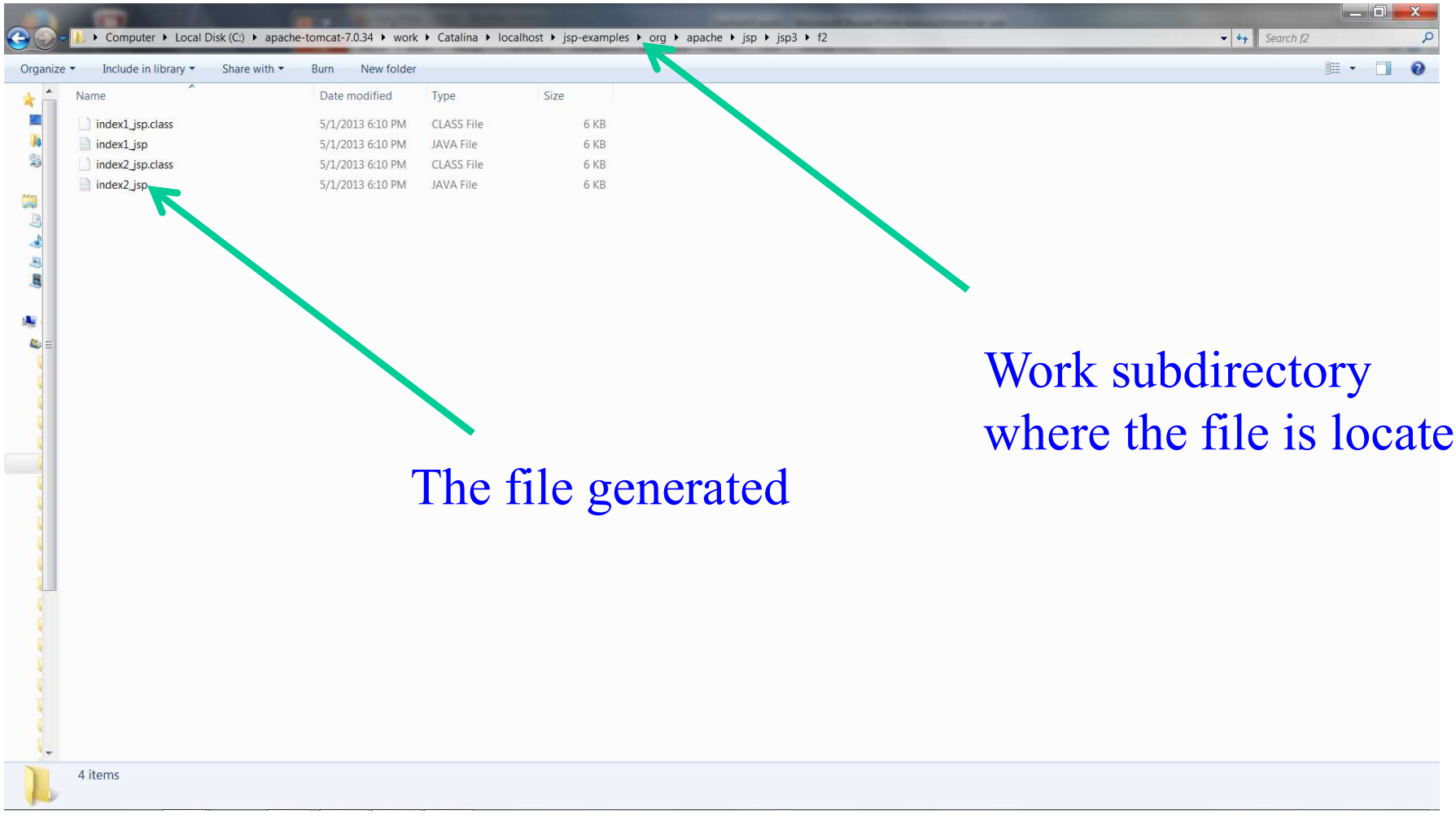
The index2.jsp included 2 files:

1. header.jsp
2. navigation.jsp

Simple JSPs

When the jsp engine encounters the include tag the jsp engine will load the header.jsp and the navigation.jsp into index2.jsp file

Code Generated by Tomcat for index2.jsp



Work subdirectory where the file is located

The file generated

Types of JSP Scripting Elements

- JSP scripting elements let you insert Java code into the servlet that will be generated from the JSP page. There are three forms:
  1. **Expressions** of the form ***<%= Java Expression %>***, which are evaluated and inserted into the servlet's output.
  2. **Scriptlets** of the form ***<% Java Code %>***, which are inserted into the servlet's _jspService method (called by service).
  3. **Declarations** of the form ***<%! Field/Method Declaration %>***, which are inserted into the body of the servlet class, outside any existing methods.

JSP Tags

- Two formats:

<% …%>

or

<jsp: … > </jsp:..>

Simple JSPs

- Now we shall look at some of the JSP tags

- **<u>Comments</u>**

  <%-- …text…--%>

- JSP comments are stripped out by the JSP engine at translation time. Consequently, they never appear in the final servlet and are never sent to the end user. Comments are used to indicate what is the code used for.

Simple JSPs

- **<u>Declarations</u>**

<%! Type varname; %>

<%! Return type methodname(argument1,argument2….){}%>

- Declarations are used to add variables or methods to a jsp. Anything in a declaration is added to the resulting servlet at the class level.

- For variables, the declaration means that there will typically be one instance of the variable shared across all requests.

Simple JSPs

- **<u>Expressions</u>**

<%= expression %>

The expression tag places printable form of a Java expression into the output of a page. Expressions may be anything from a simple variable to a method call to a complex mathematical form involving multiple terms.

Simple JSPs

- **<u>Scriptlets</u>**

<%….java code…%>

Scriptlets allow arbitrary java code, to be placed in a JSP. For the most part such code should reside in beans and other classes, but there are times when placing it in a page is unavoidable. Complex logic can be added to pages by surrounding regions of text with scriptlets, where one scriptlet ends with an open brace and a second one provides the matching close.

Simple JSPs

- **<u>Include Directive</u>**

<%@ include file=“”%>

The include directive adds the text of one jsp or other file to another file at translation time.

The effect is exactly as if the author had used an editor to paste the included file into the primary one.

JSP Expressions

- A JSP expression is used to insert values directly into the output. It has the following form:
  - ***<%= Java Expression %>***
- The expression is evaluated, converted to a string, and inserted in the page.
- This evaluation is performed at runtime (when the page is requested) and thus has full access to information about the request.
- For example, the following shows the date/time that the page was requested.
  - **Current time: <%= new java.util.Date() %>**

JSP Expressions

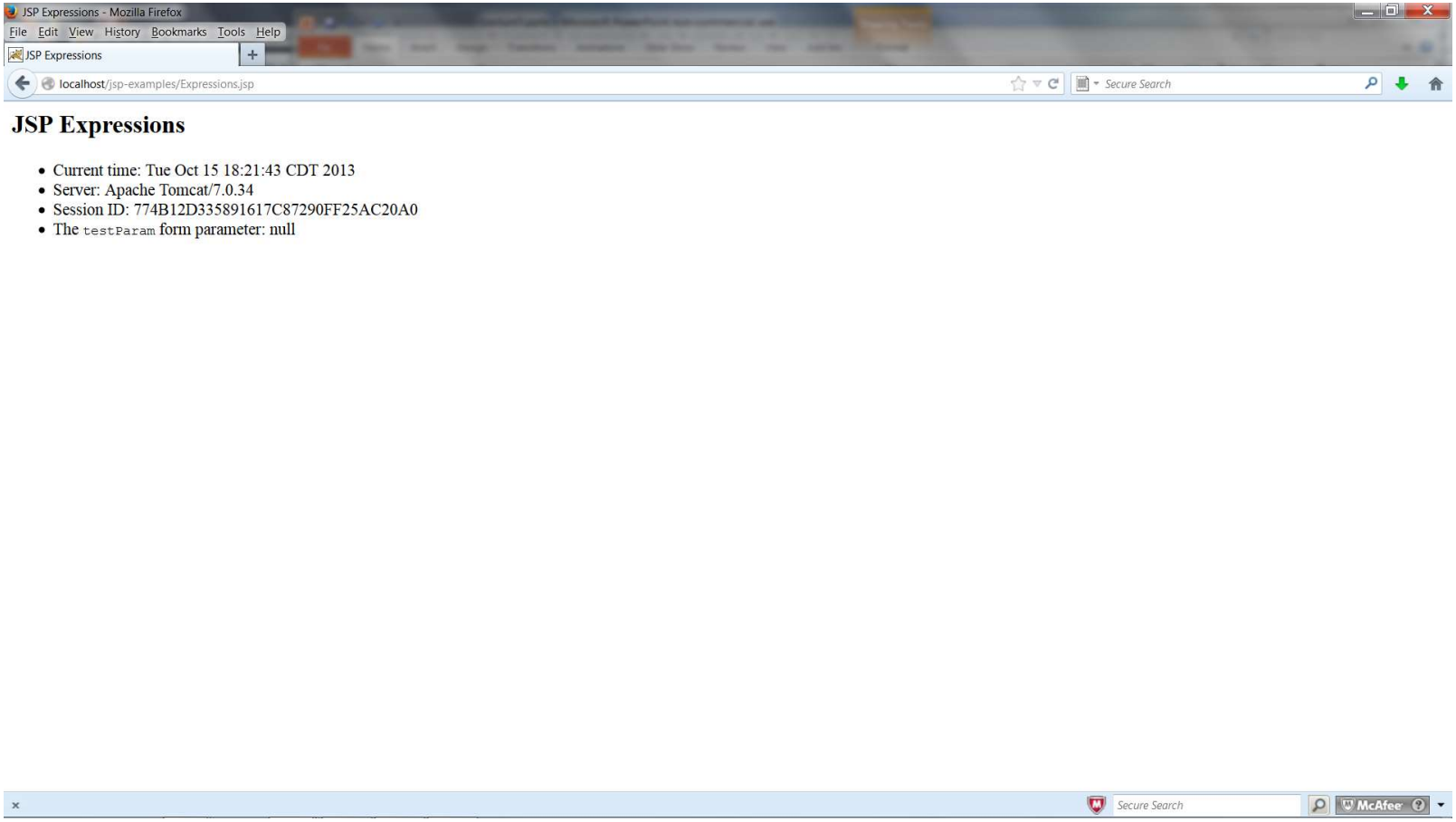
- **Predefined Variables**
  - To simplify these expressions, you can use a number of predefined variables (or “implicit objects”).
  - The system uses these names as local variables in _jspService (the method that replaces doGet in servlets that result from JSP pages).

JSP Expressions

- The most important **Predefined Variables** are :
  - **request**, the HttpServletRequest.
  - **response**, the HttpServletResponse.
  - **session**, the HttpSession associated with the request
  - **out**, the Writer (a buffered version of type JspWriter) used to send output to the client.
  - **application**, the ServletContext. This is a data structure shared by all servlets and JSP pages in the Web application and is good for storing shared data.

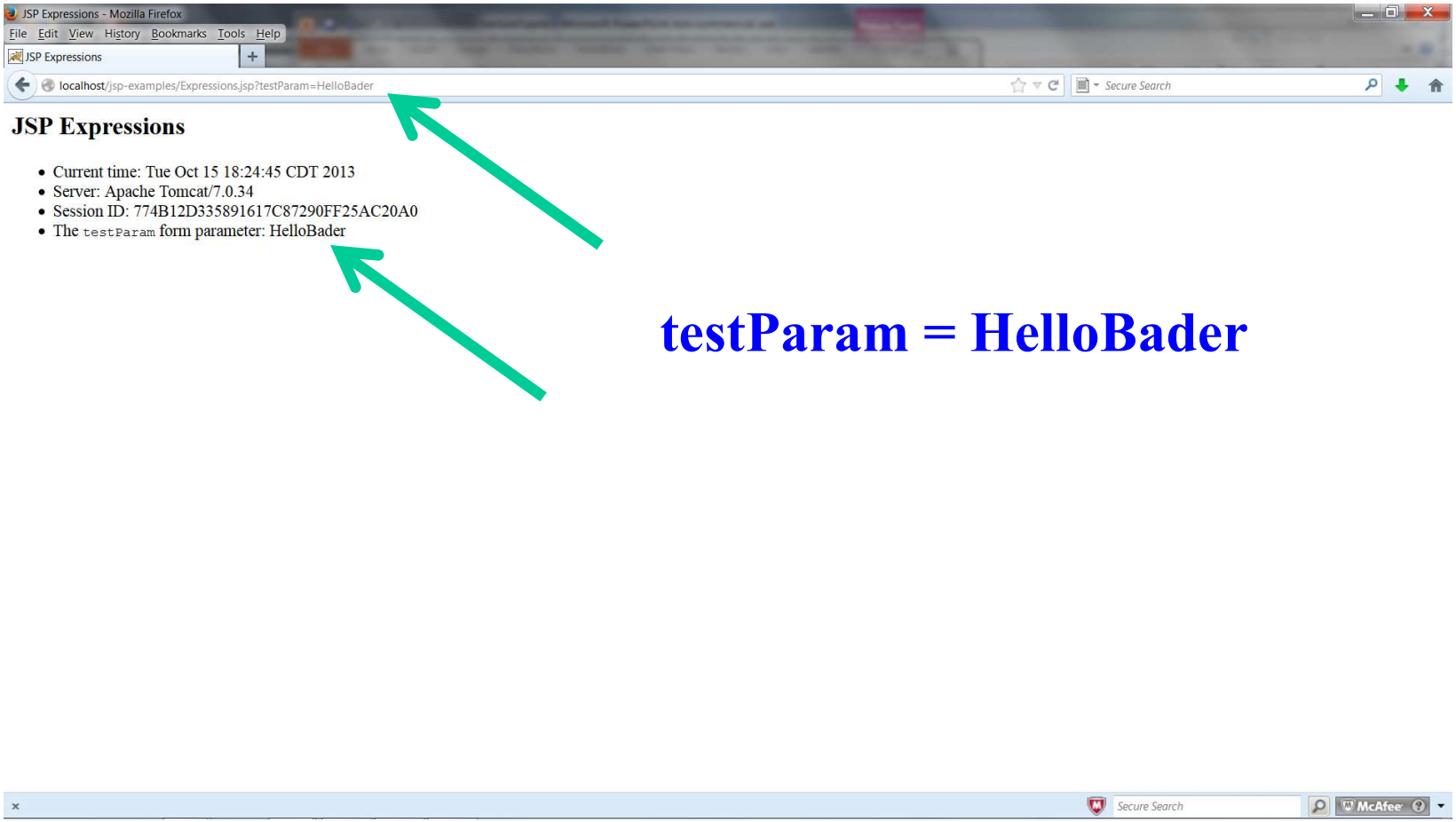
JSP Expression

- http://localhost/jsp-examples/Expressions.jsp



JSP Expression

- http://localhost/jsp-examples/Expressions.jsp



Code Generated by Tocamt for http://localhost/jsp-examples/Expressions.jsp

Expression_jsp File

```
/*
 * Generated by the Jasper component of Apache Tomcat
 * Version: Apache Tomcat/7.0.34
 * Generated at: 2013-05-01 15:38:09 UTC
 * Note: The last modified time of this file was set to
 *       the last modified time of the source file after
 *       generation to assist with modification tracking.
 */
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class Expressions_jsp extends org.apache.jasper.runtime.HttpJspBase
   implements org.apache.jasper.runtime.JspSourceDependent {

  private static final javax.servlet.jsp.JspFactory _jspxFactory =
        javax.servlet.jsp.JspFactory.getDefaultFactory();

  private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;

  private javax.el.ExpressionFactory _el_expressionfactory;
  private org.apache.tomcat.InstanceManager _jsp_instancemanager;

  public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
    return _jspx_dependants;
  }
```

Code Generated by Tocamt for http://localhost/jsp-examples/Expressions.jsp

Computer ▸ Local Disk (C:) ▸ apache-tomcat-7.0.34 ▸ work ▸ Catalina ▸ localhost ▸ jsp-examples ▸ org ▸ apache ▸ jsp ▸

Organize ▾ Include in library ▾ Share with ▾

Name

- jsp3
- BakedBeanDisplay_002dpage_jsp.class
- BakedBeanDisplay_002dpage_jsp
- BakedBeanDisplay_002drequest_jsp.class
- BakedBeanDisplay_002drequest_jsp
- BakedBeanDisplay_002dsession_jsp.class
- BakedBeanDisplay_002dsession_jsp
- BakedBeanDisplay_002dsnippet_jsp.class
- BakedBeanDisplay_002dsnippet_jsp
- BGColor_jsp.class
- BGColor_jsp
- DayWish_jsp.class
- DayWish_jsp
- display_005femail_005fentry_jsp.class
- display_005femail_005fentry_jsp
- Expressions_jsp.class
- Expressions_jsp
- SaleEntry1_002dForm_jsp.class
- SaleEntry1_002dForm_jsp
- SaleEntry1_jsp.class
- SaleEntry1_jsp
- StringBean_jsp.class
- StringBean_jsp

23 items

Expressions_jsp - Notepad

File Edit Format View Help

```
public void _jspDestroy() {
}

public void _jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletResponse
    throws java.io.IOException, javax.servlet.ServletException {

  final javax.servlet.jsp.PageContext pageContext;
  javax.servlet.http.HttpSession session = null;
  final javax.servlet.ServletContext application;
  final javax.servlet.ServletConfig config;
  javax.servlet.jsp.JspWriter out = null;
  final java.lang.Object page = this;
  javax.servlet.jsp.JspWriter _jspx_out = null;
  javax.servlet.jsp.PageContext _jspx_page_context = null;

  try {
    response.setContentType("text/html");
    pageContext = _jspxFactory.getPageContext(this, request, response,
                                null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;
```

Predefined Variable

Ln 1, Col 1

Code Generated by Tocamt for http://localhost/jsp-examples/Expressions.jsp

Computer ▸ Local Disk (C:) ▸ apache-tomcat-7.0.34 ▸ work ▸ Catalina ▸ localhost ▸ jsp-examples ▸ org ▸ apache ▸ jsp ▸

Organize ▾ Include in library ▾ Share with ▾

Name

- jsp3
- BakedBeanDisplay_002dpage_jsp.class
- BakedBeanDisplay_002dpage_jsp
- BakedBeanDisplay_002drequest_jsp.class
- BakedBeanDisplay_002drequest_jsp
- BakedBeanDisplay_002dsession_jsp.class
- BakedBeanDisplay_002dsession_jsp
- BakedBeanDisplay_002dsnippet_jsp.class
- BakedBeanDisplay_002dsnippet_jsp
- BGColor_jsp.class
- BGColor_jsp
- DayWish_jsp.class
- DayWish_jsp
- display_005femail_005fentry_jsp.class
- display_005femail_005fentry_jsp
- Expressions_jsp.class
- Expressions_jsp
- SaleEntry1_002dForm_jsp.class
- SaleEntry1_002dForm_jsp
- SaleEntry1_jsp.class
- SaleEntry1_jsp
- StringBean_jsp.class
- StringBean_jsp

23 items

Expressions_jsp - Notepad

File Edit Format View Help

```
    out.write("<LINK REL=STYLESHEET\r\n");
    out.write("      HREF=\"JSP-Styles.css\"\r\n");
    out.write("      TYPE=\"text/css\">\r\n");
    out.write("</HEAD>\r\n");
    out.write("<BODY>\r\n");
    out.write("<H2>JSP Expressions</H2>\r\n");
    out.write("<UL>\r\n");
    out.write("  <LI>Current time: ");
    out.print( new java.util.Date() );
    out.write("\r\n");
    out.write("  <LI>Server: ");
    out.print( application.getServerInfo() );
    out.write("\r\n");
    out.write("  <LI>Session ID: ");
    out.print( session.getId() );
    out.write("\r\n");
    out.write("  <LI>The <CODE>testParam</CODE> form parameter:\r\n");
    out.write("      ");
    out.print( request.getParameter("testParam") );
    out.write("\r\n");
    out.write("</UL>\r\n");
    out.write("</BODY></HTML>");
  } catch (java.lang.Throwable t) {
    if (!(t instanceof javax.servlet.jsp.SkipPageException)){
      out = _jspx_out;
      if (out != null && out.getBufferSize() != 0)
        try { out.clearBuffer(); } catch (java.io.IOException e) {}
      if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
```

Ln 1, Col 1

Output

JSP Expression

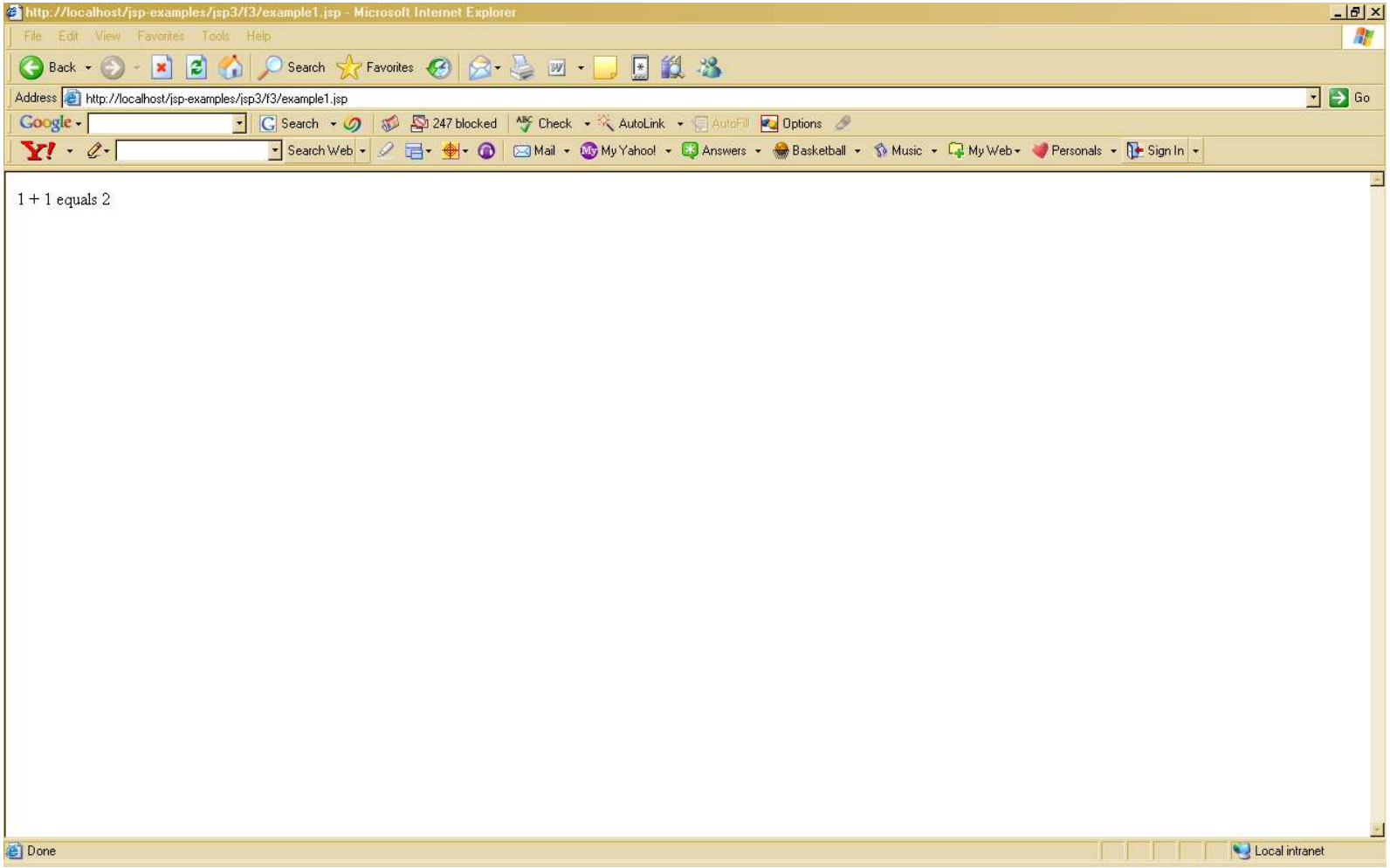
We can use the expression tag to do some mathematical calculations

```
<HTML>
<BODY>
1 + 1 equals <%= 1 + 1 %>
</BODY>
</HTML>
```

- http://localhost/jsp-examples/jsp3/f3/example1.jsp

JSP Expression

Here is the output …



JSP Implicit Object

- The implicit object **request** is an important one for passing information from the browser to the server

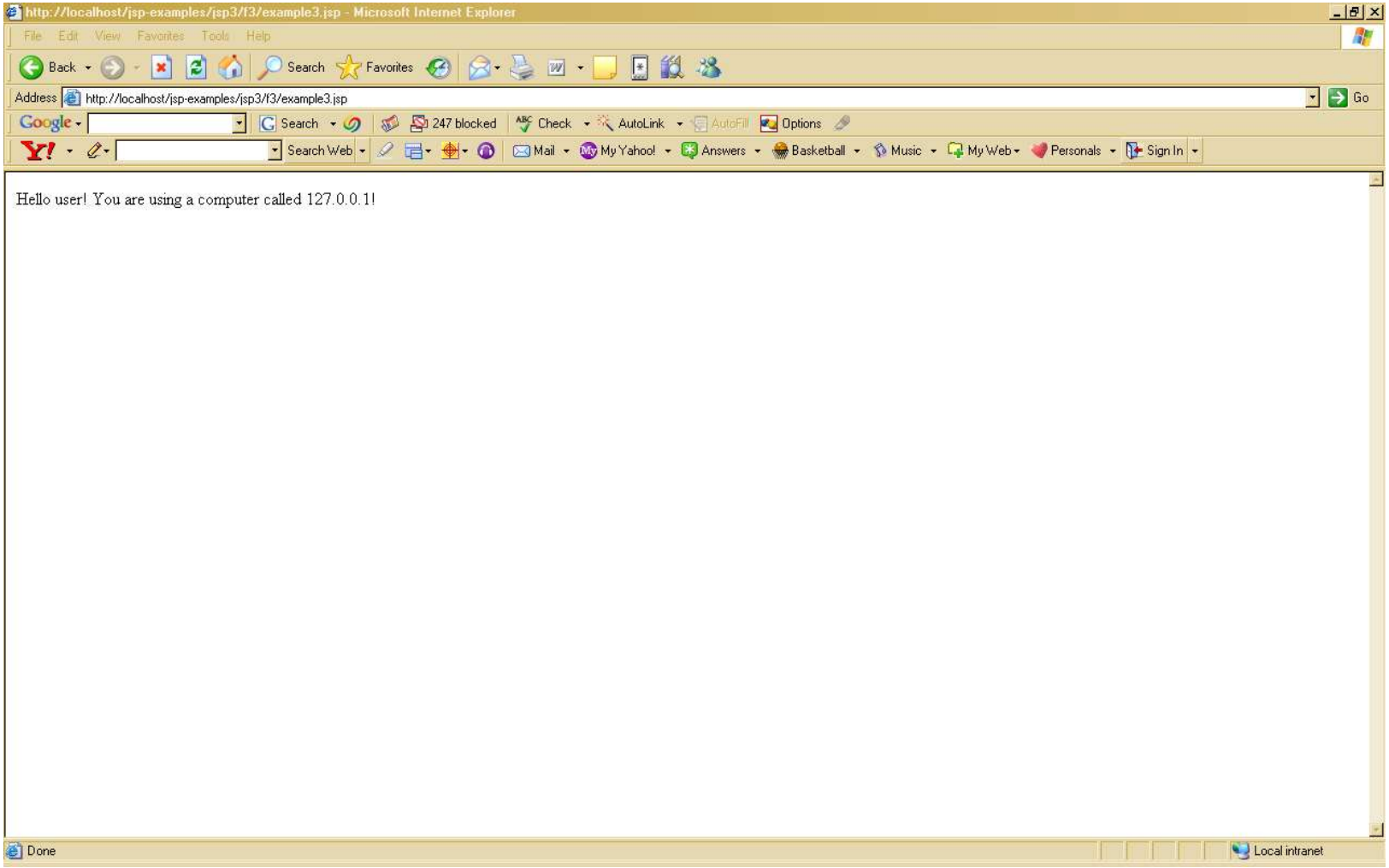
```
<HTML>
<BODY>
Hello user!  You are using a computer
called <%= request.getRemoteHost() %>!
</BODY>
</HTML>
```

- http://localhost/jsp-examples/jsp3/f3/example3.jsp

- http://localhost/jsp-examples/jsp3/f3/example4.jsp

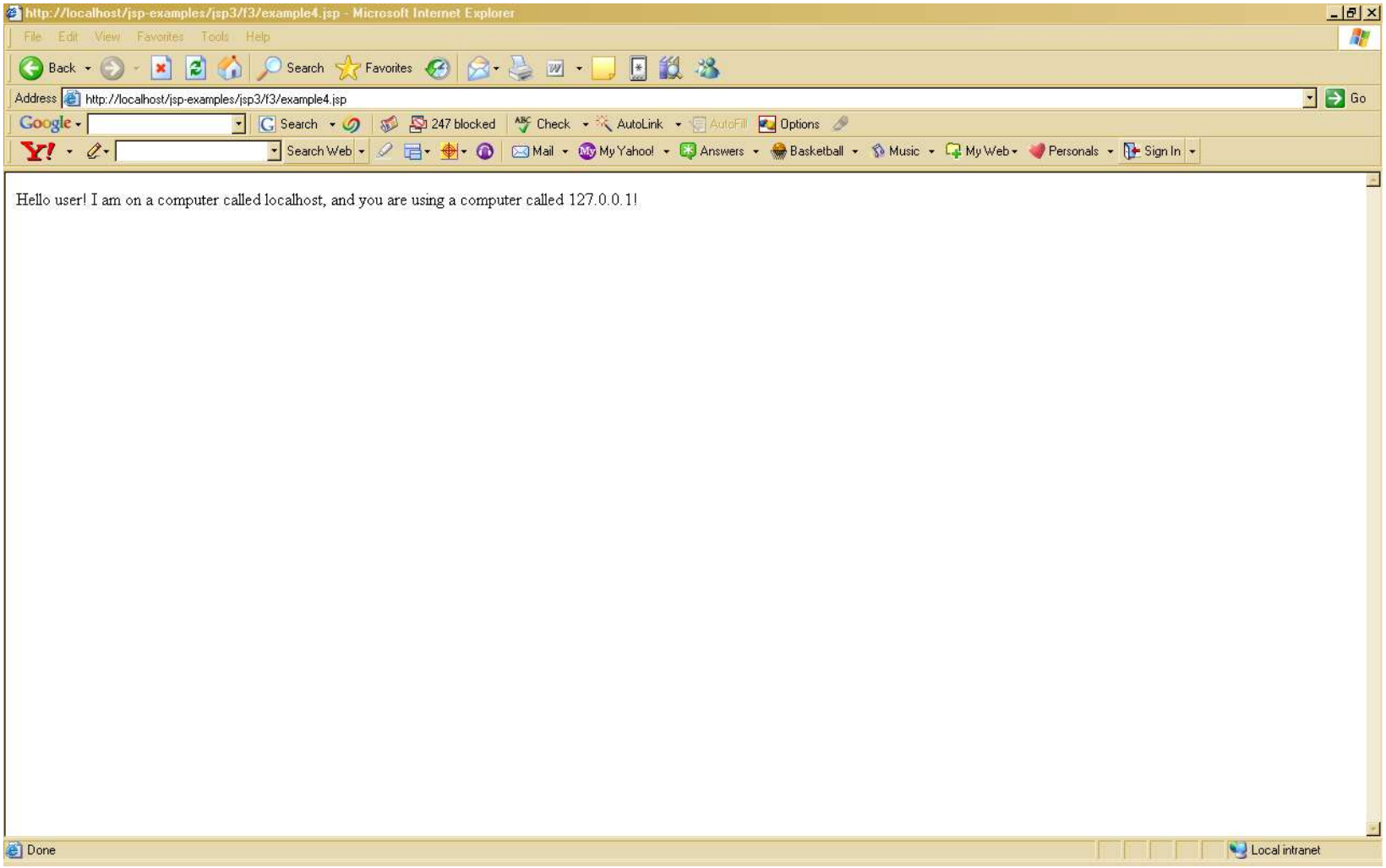
JSP Implicit Object

- The output for example3.jsp



JSP Implicit Object

- The output for example4.jsp



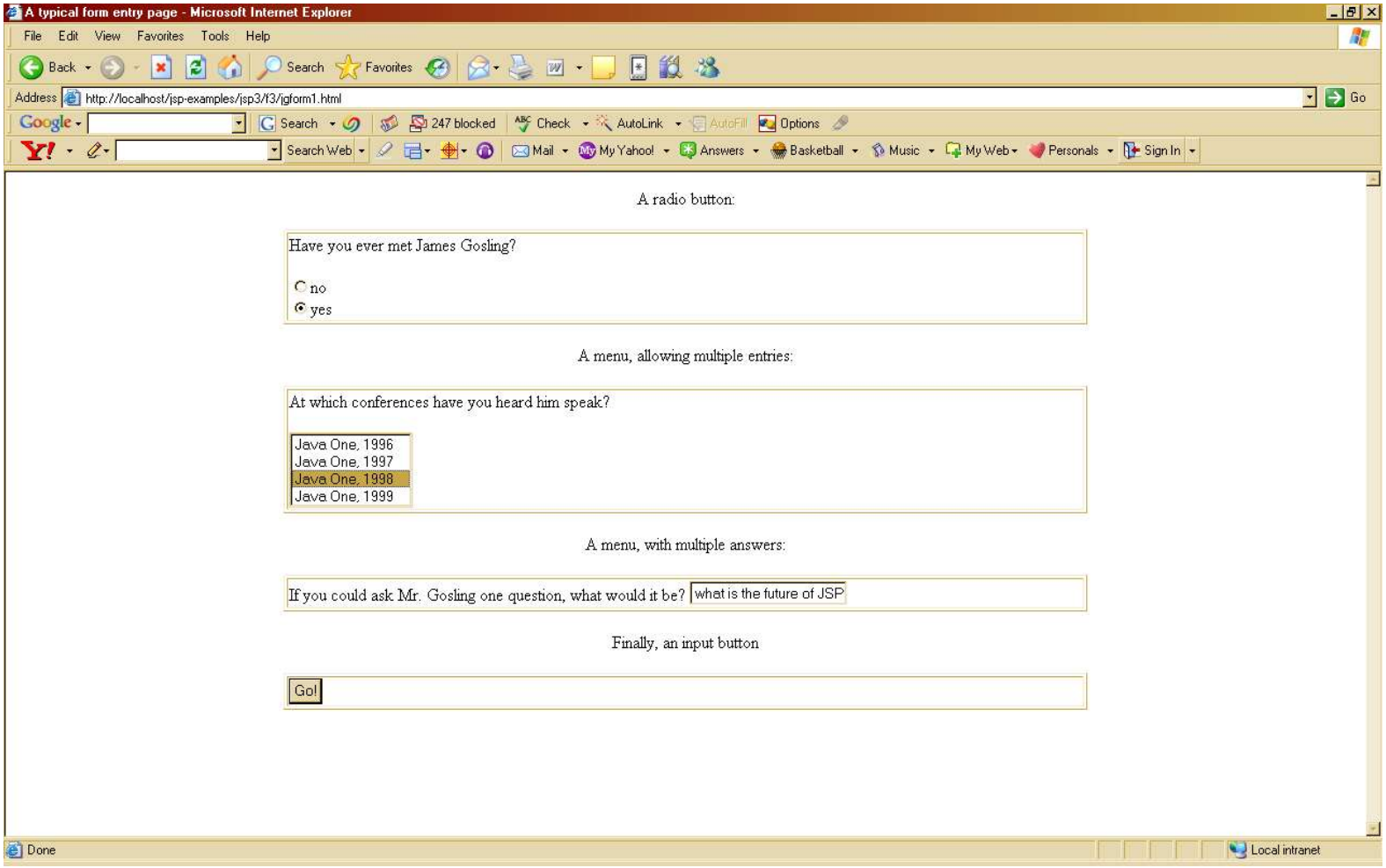
Request.getParameter()

- Some times we need to get back from the form more than one variable and JSP provides us with the method request.getParameter("parameter name") to get the value of each variable in the form

- http://localhost/jsp-examples/jsp3/f3/jgform1.html

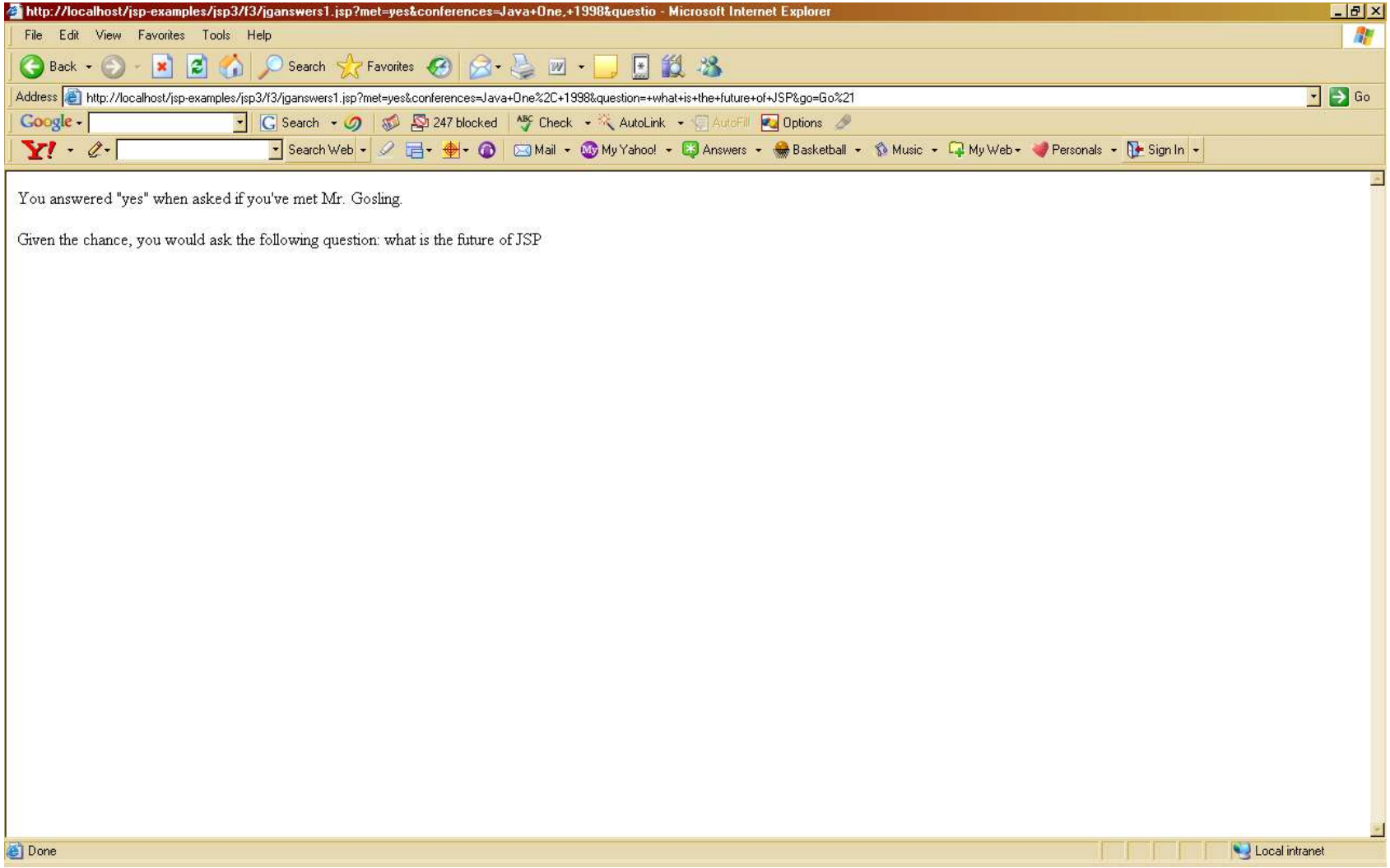
Request.getParameter()

- Here is the page …



Request.getParameter()

- Here is the page after pressing the GO button …



Scriptlet

- If you want to do something more complex than output the value of a simple expression, JSP scriptlets let you insert arbitrary code into the servlet's _jspService method (which is called by service).

- Scriptlets have the following form:
  - ***<% Java Code %>***

- Scriptlets have access to the same automatically defined variables as do expressions (request, response, session, out, etc.).

- So, for example, if you want to explicitly send output to the resultant page, you could use the out variable, as in the following example.

```
<%
String queryData = request.getQueryString();
out.println("Attached GET data: " + queryData);
%>
```

Scriptlet Example

- As an example of code that is too complex for a JSP expression alone, following example presents a JSP page that uses the bgColor request parameter to set the background color of the page.

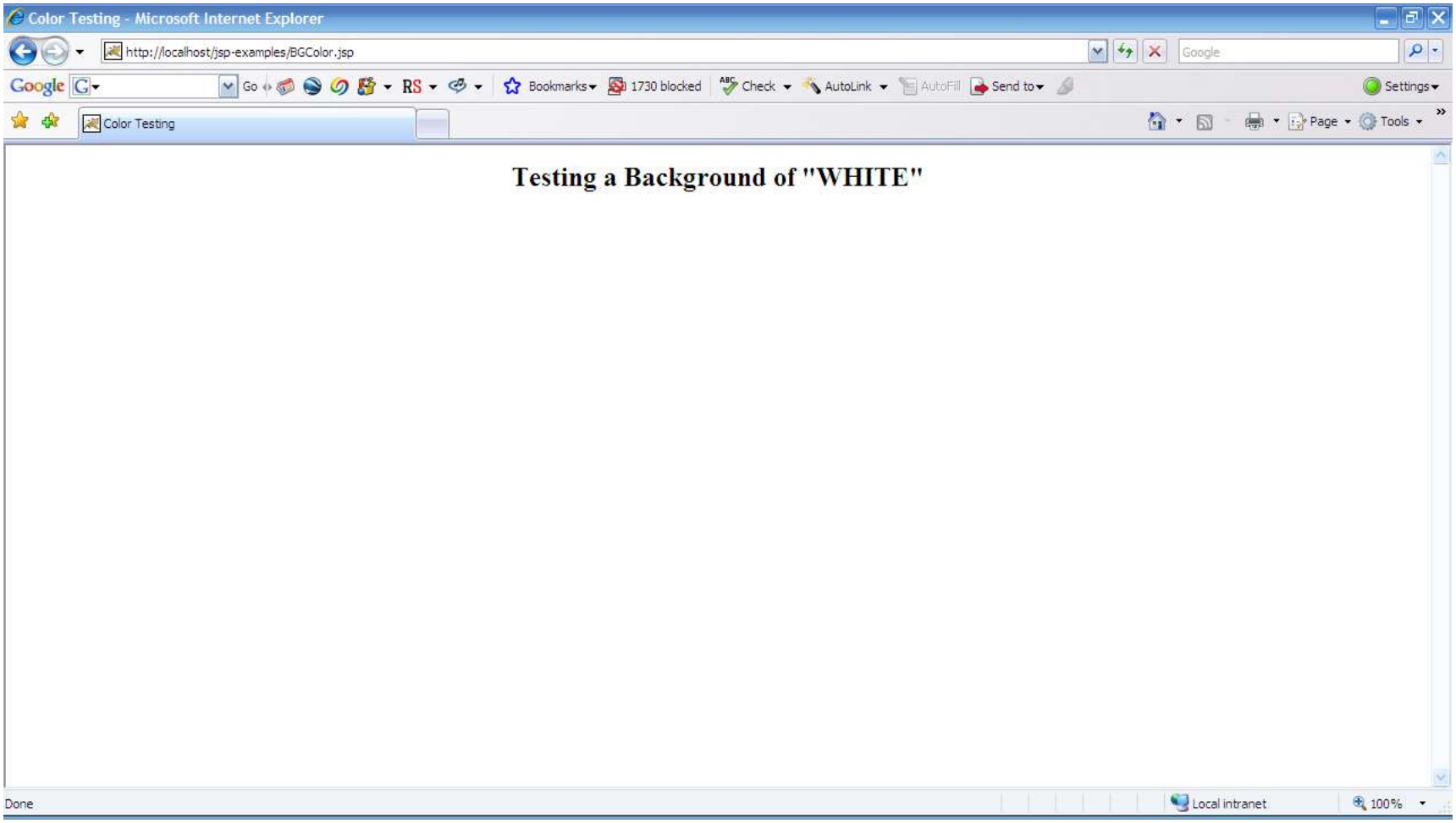
- Simply using

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

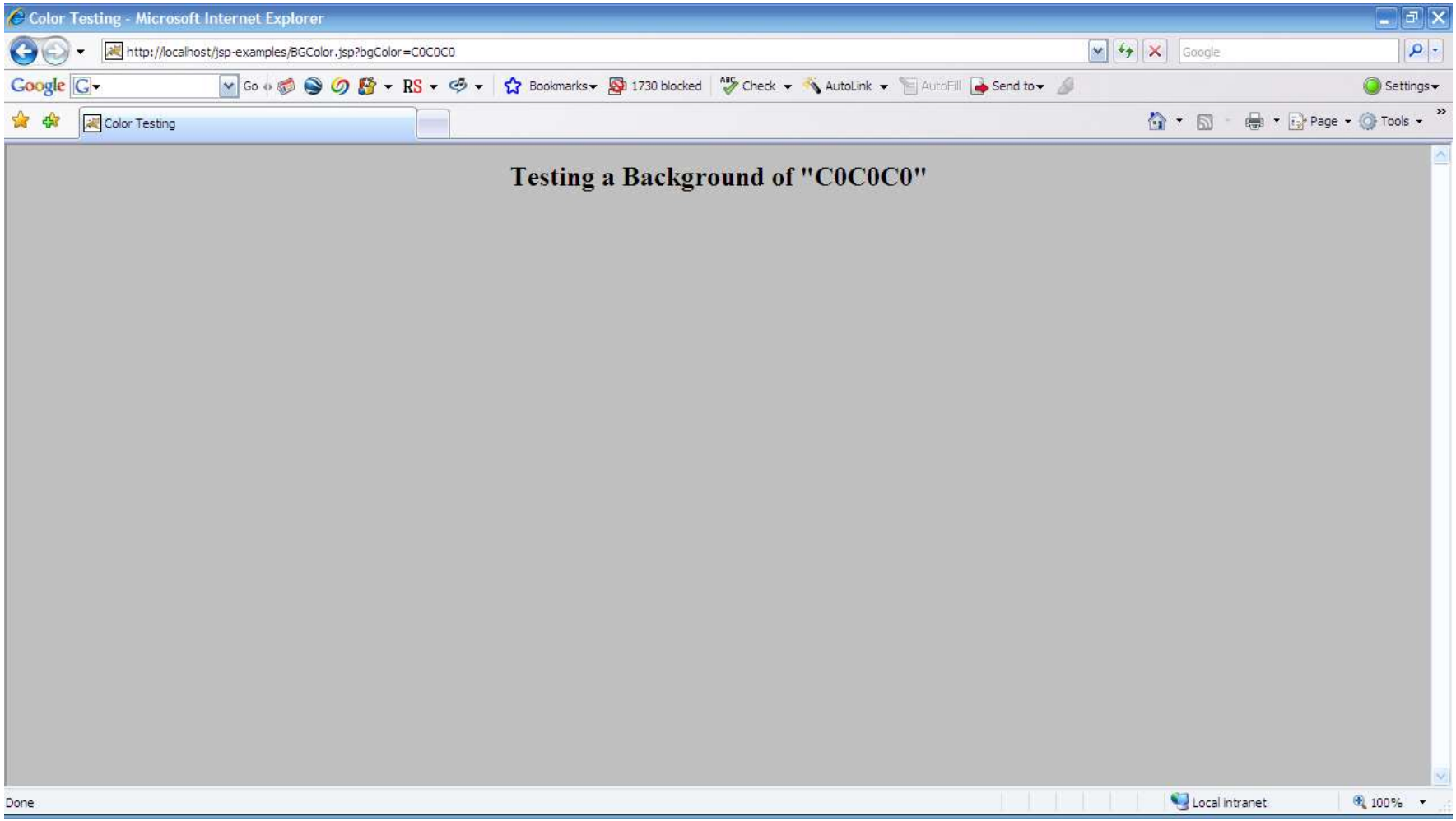
- would violate the cardinal rule of reading form data: always check for missing or malformed data. So, we use a scriptlet instead.

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

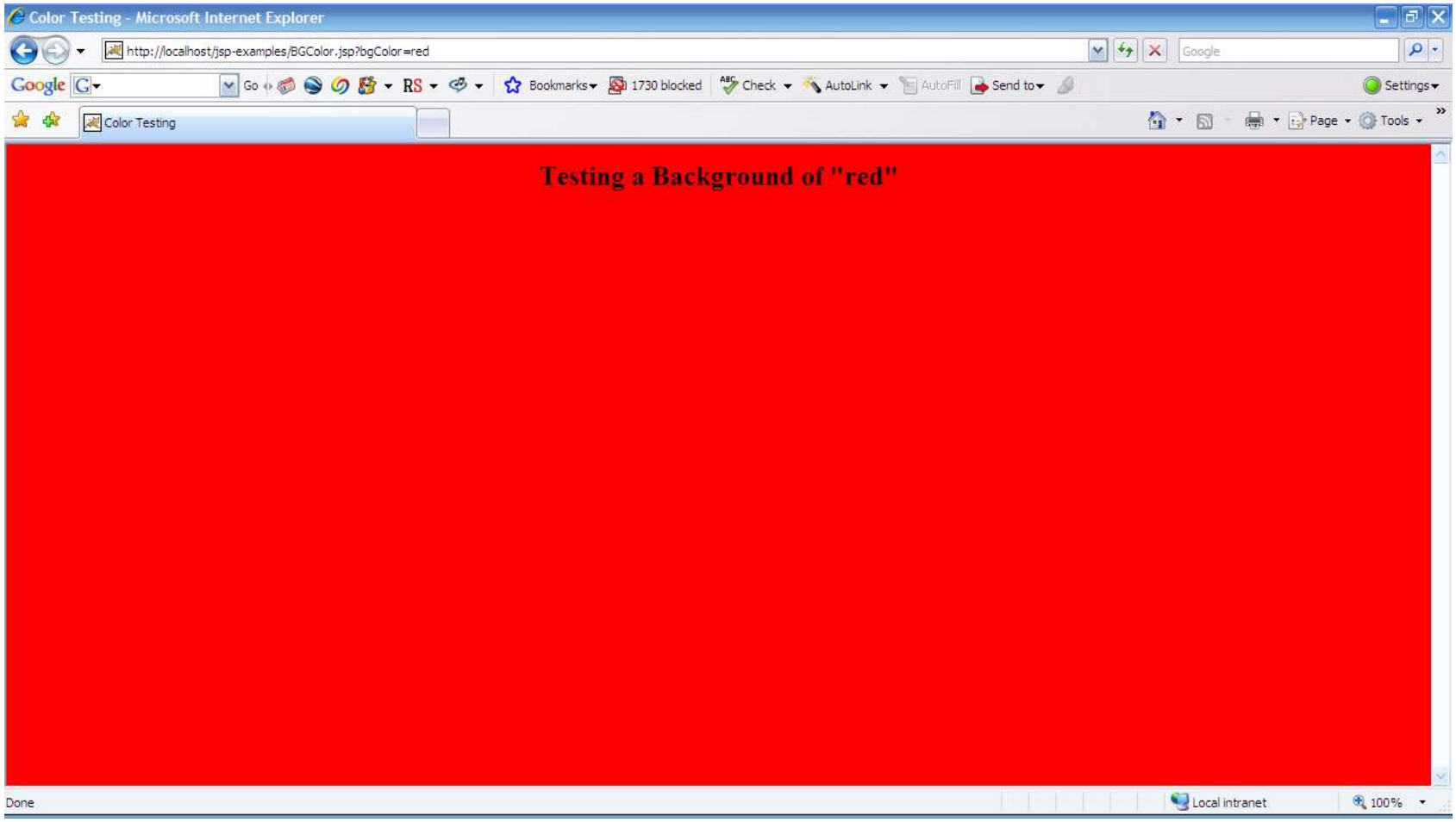
Scriptlet Example



Scriptlet Example



Scriptlet Example



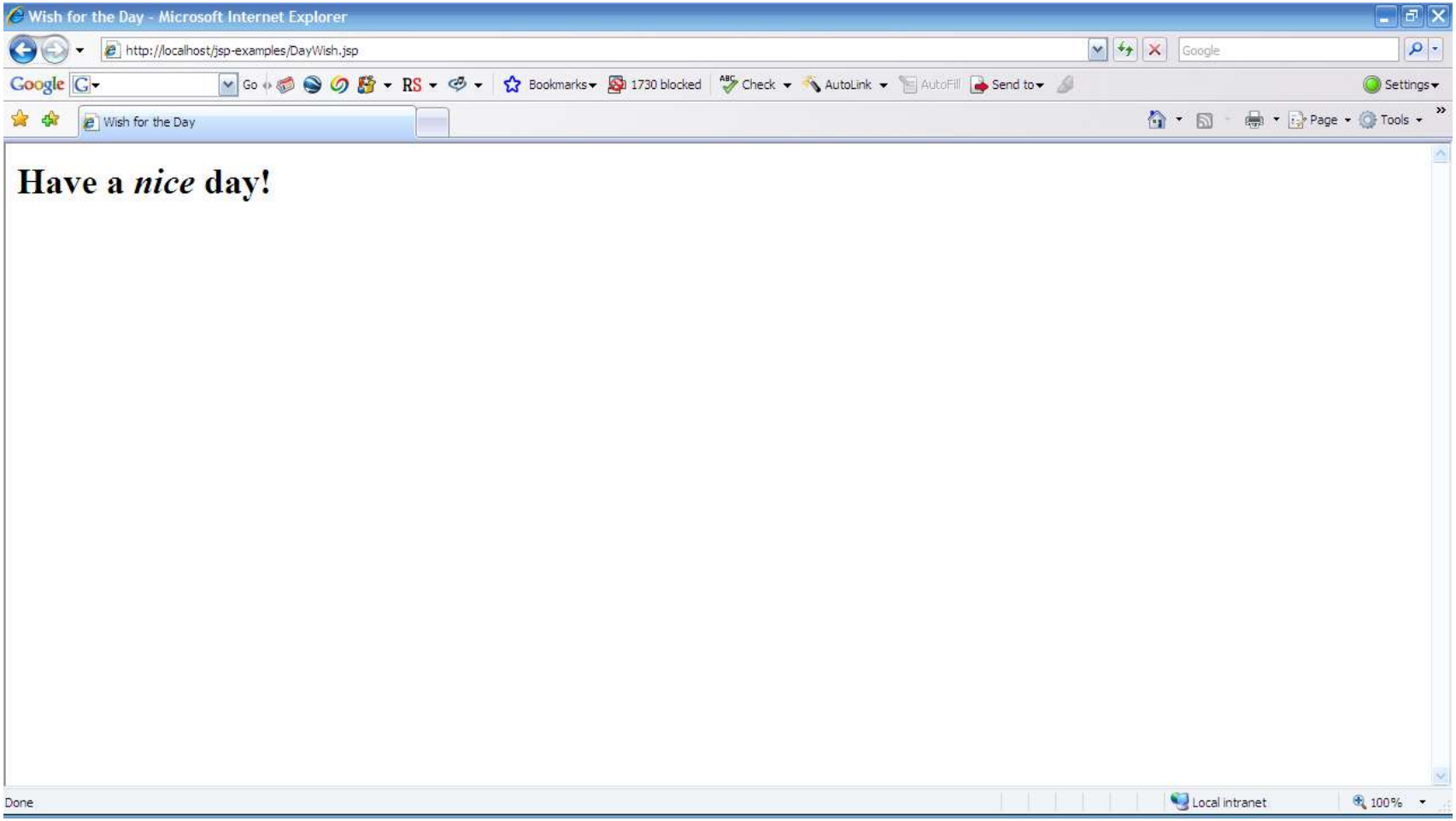
Using Scriptlets to Make Parts of the JSP Page Conditional

- Scriptlets are used to conditionally output HTML or other content that is *not* within any JSP tag.

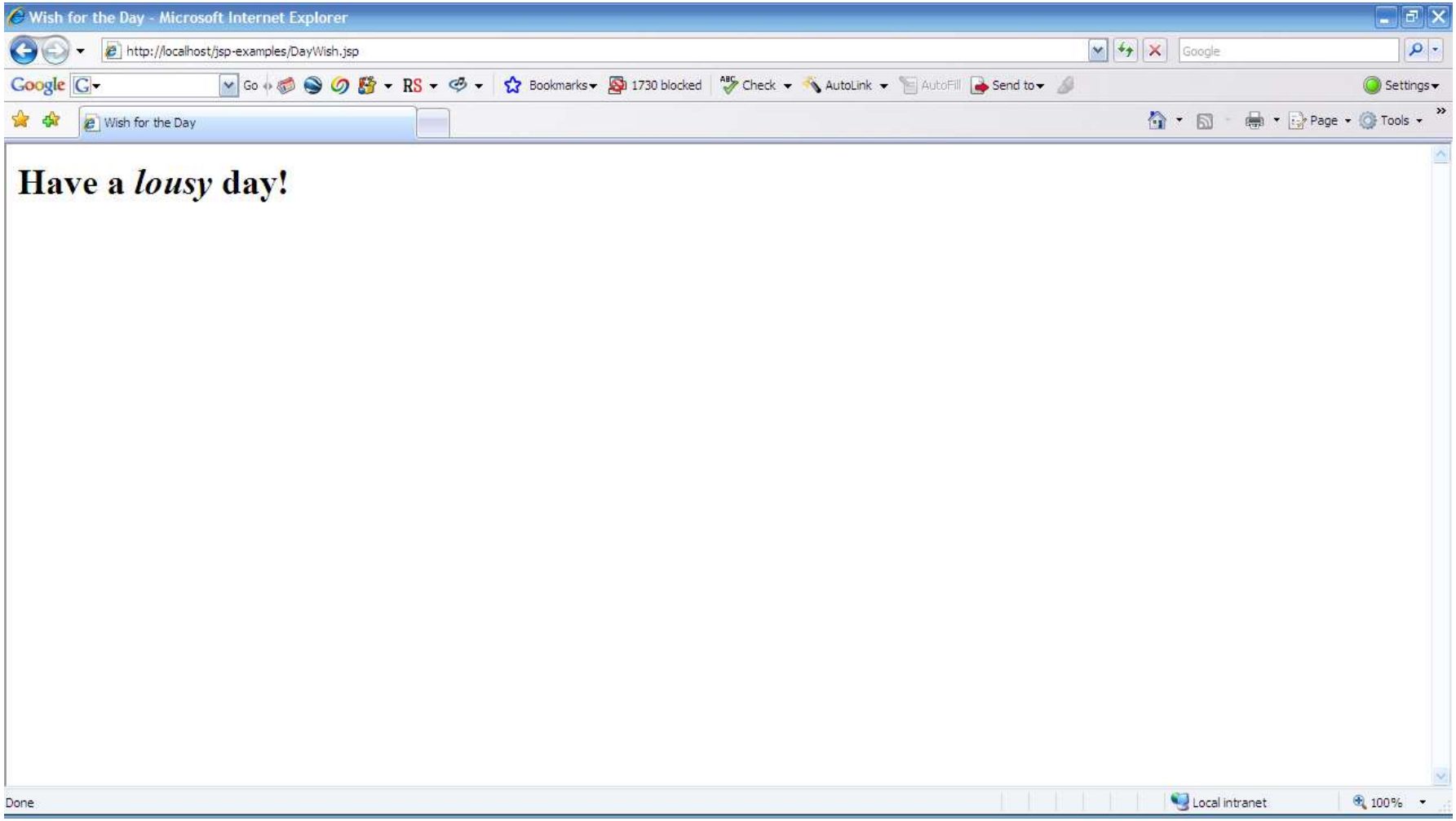
- The key to this approach are the facts that
  - (a) code inside a scriptlet gets inserted into the resultant servlet's _jspService method (called by service) *exactly* as written and
  - (b) that any static HTML (template text) before or after a scriptlet gets converted to print statements.

- http://localhost/jsp-examples/DayWish.jsp

Using Scriptlets to Make Parts of the JSP Page Conditional



Using Scriptlets to Make Parts of the JSP Page Conditional



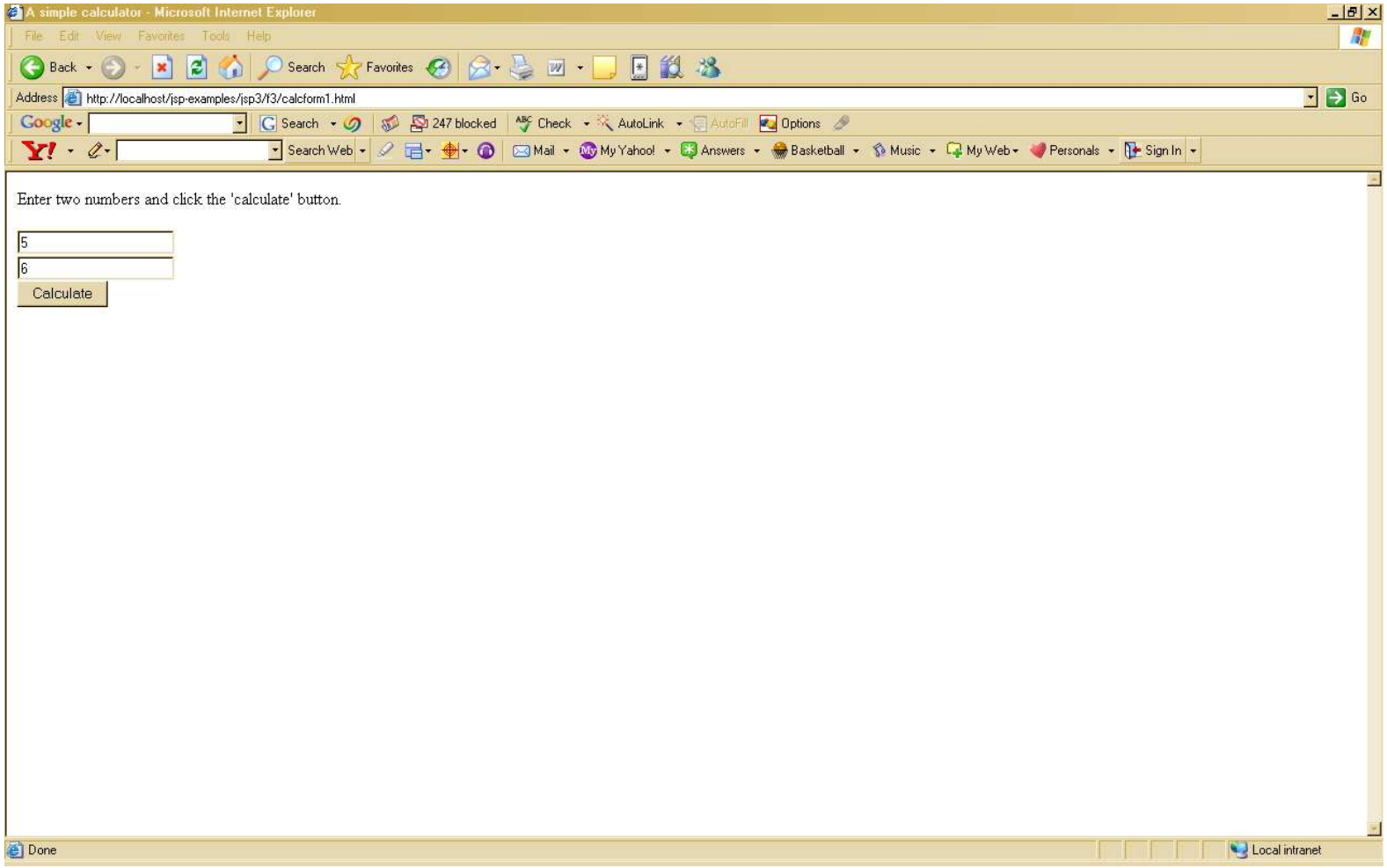
Simple Calculator

- We want the user to enter 2 values and then call request.getParameter("parameter name") and then find the sum of the 2 values. Pay careful attention to "+" it is concatenation and we want to add 2 numbers, so we need to use Integer.parseInt(…)

- http://localhost/jsp-examples/jsp3/f3/calcform1.html

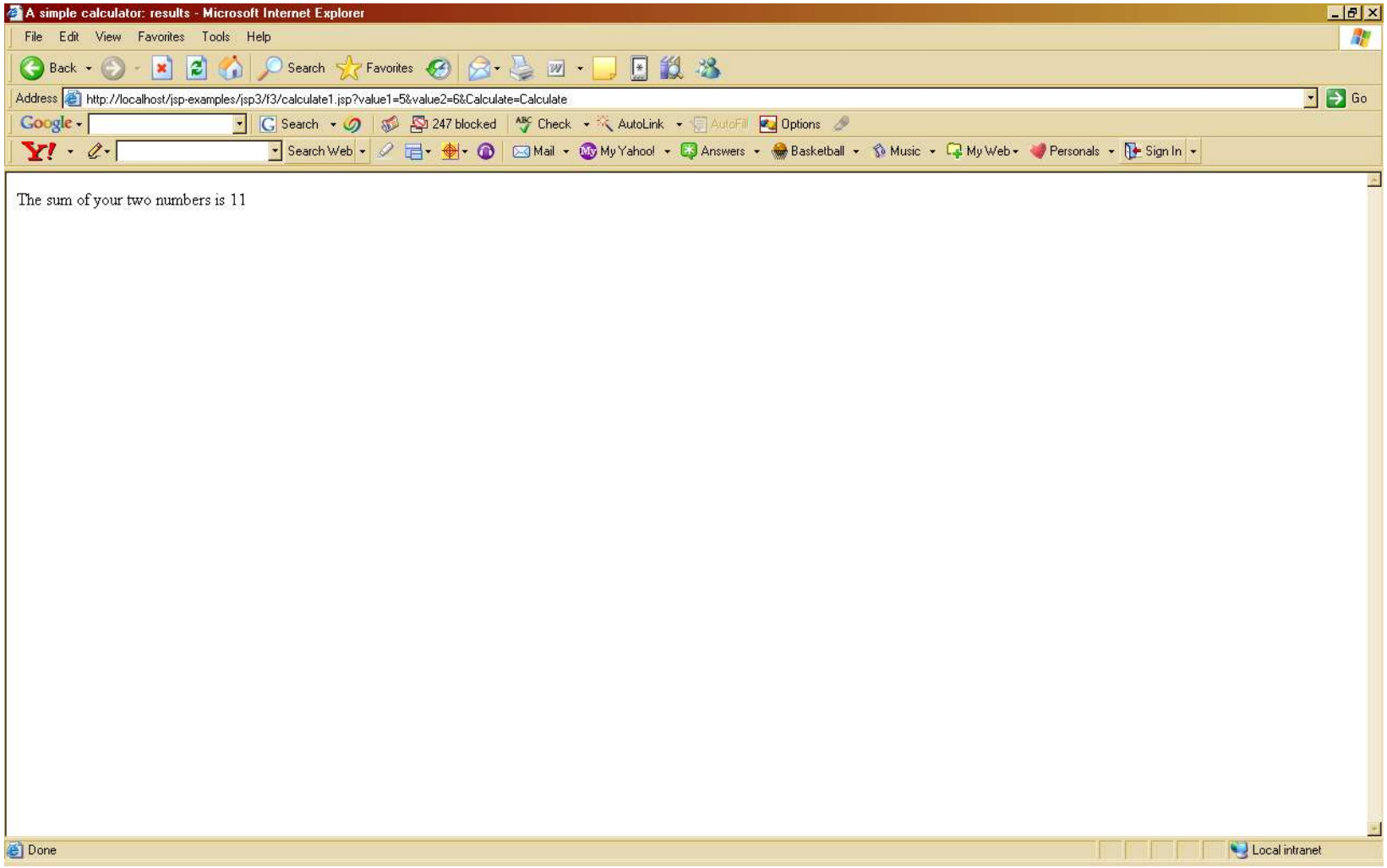
Simple Calculator

- Here is the page …



Simple Calculator

- Here is the page after pressing the Calculate button…



Beans

- **Why Use Beans?**
- The use of JavaBeans components in JSP provides three advantages over scriptlets and JSP expressions that refer to normal Java classes:
  1. **No Java syntax.** By using beans, page authors can manipulate Java objects using only XML-compatible syntax: no parentheses, semicolons, or curly braces.
  2. **Simpler object sharing.** When you use the JSP bean constructs, you can much more easily share objects among multiple pages or between requests than if you use the equivalent explicit Java code.
  3. **Convenient correspondence between request parameters and object properties.** The JSP bean constructs greatly simplify the process of reading request parameters, converting from strings, and putting the results inside objects.

Beans

- **Beans Coding Styles and Constraints**
  - **A bean class must have a zero-argument (default) constructor.** The default constructor will be called when JSP elements create beans.
  - **A bean class should have no public instance variables (fields).**
  - **Persistent values should be accessed through methods called get*Xxx* and set*Xxx*.**
    - The one exception to this naming convention is with boolean properties: they are permitted to use a method called is*Xxx* to look up their values; So, for example, your Car class might have methods called isLeased and setLeased

Beans

- **<u>Use three main constructs</u>** to build and manipulate JavaBeans components in JSP pages:
  - **jsp:useBean.** In the simplest case, this element builds a new bean. It is normally used as follows:
    - <jsp:useBean id="beanName“ class="package.Class" />

Beans

- **jsp:getProperty.** reads and outputs the value of a bean property. Reading a property is a shorthand notation for calling a method of the form get*Xxx*. This element is used as follows:
  - <jsp:getProperty name="beanName“ property="propertyName" />
- **jsp:setProperty.** Modifies a bean property (i.e., calls a method of the form set*Xxx*). It is normally used as follows:
  - <jsp:setProperty name="beanName“ property="propertyName“ value="propertyValue" />

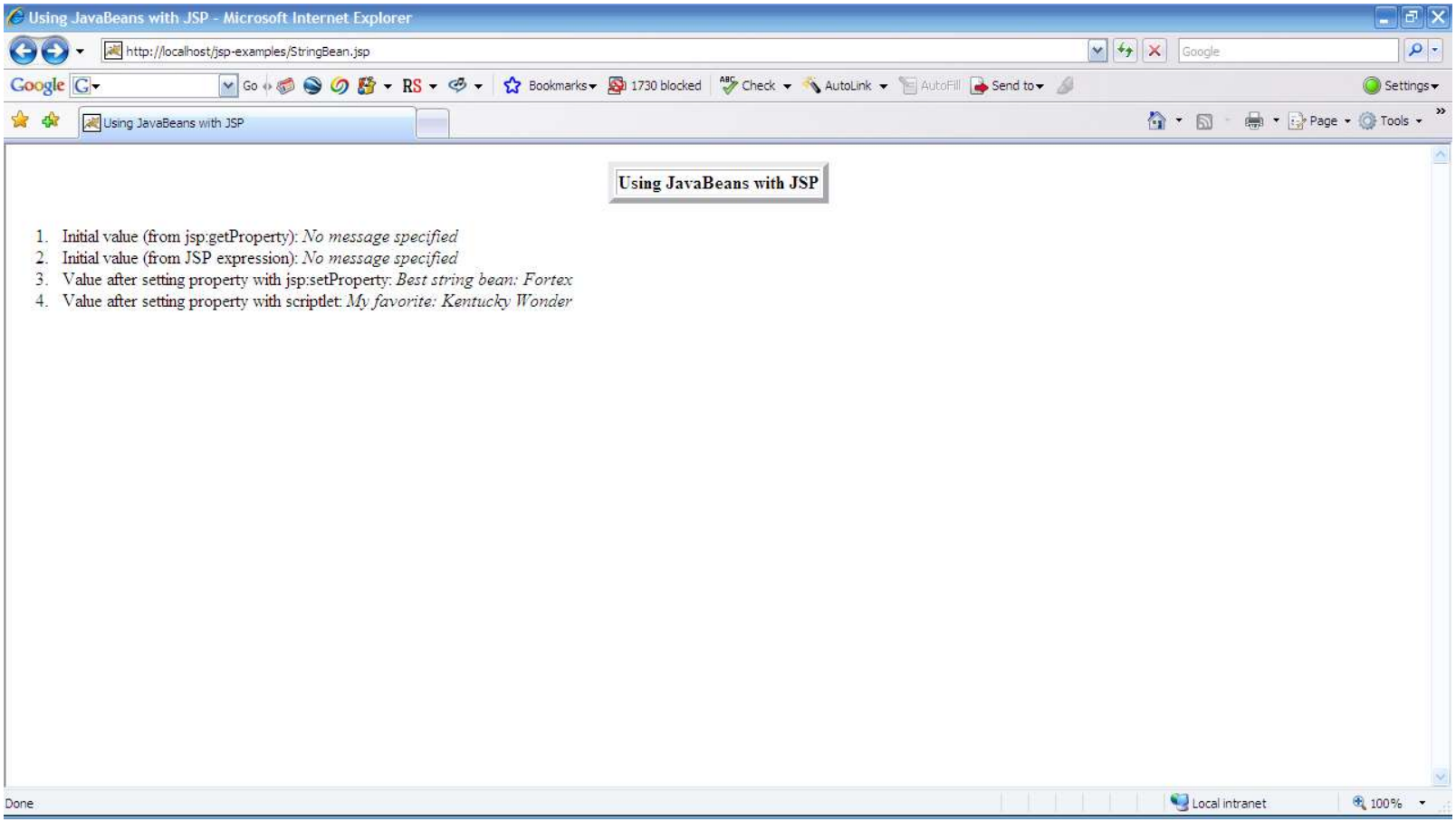
Example: StringBean

- Always put your Java Beans in Packages

- Following example presents a simple class called StringBean that is in the mybean package.

- http://localhost/jsp-examples/StringBean.jsp
  - Calls StringBean from
  - C:\apache-tomcat-7.0.34\webapps\jsp-examples\WEB-INF\classes\mybean

Example: StringBean



Beans

- A Bean is a black-box component
- Just follows a naming convention, for each variable x, there are 2 methods defined on the bean getX(), setX(..)
- JSP uses the following tags to declare and access a bean:

```
<jsp:useBean id="bean1" class="mybean.Bean1"/>

<jsp:getProperty name="bean1" property="currentTime"/>
```

Beans

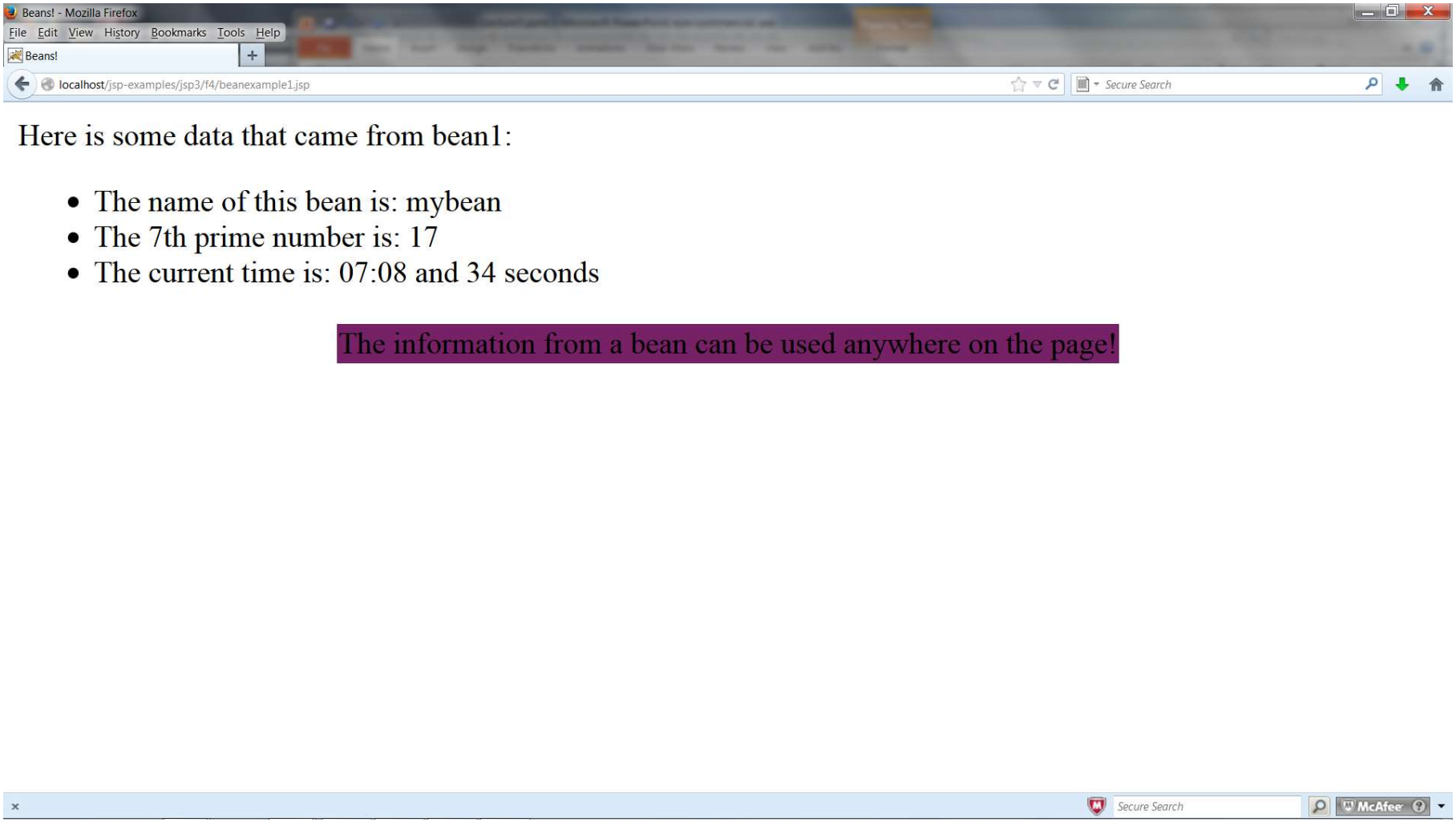
- Build your bean, Bean1.java, and save it in the following directory:

  - **C:\apache-tomcat-7.0.34\webapps\jsp-examples\WEB-INF\classes\mybean**

- Let us see how we can call a bean from jsp
  - http://localhost/jsp-examples/jsp3/f4/beanexample1.jsp

Beans

- Here is the output of beanexample1.jsp



Beans - Forms

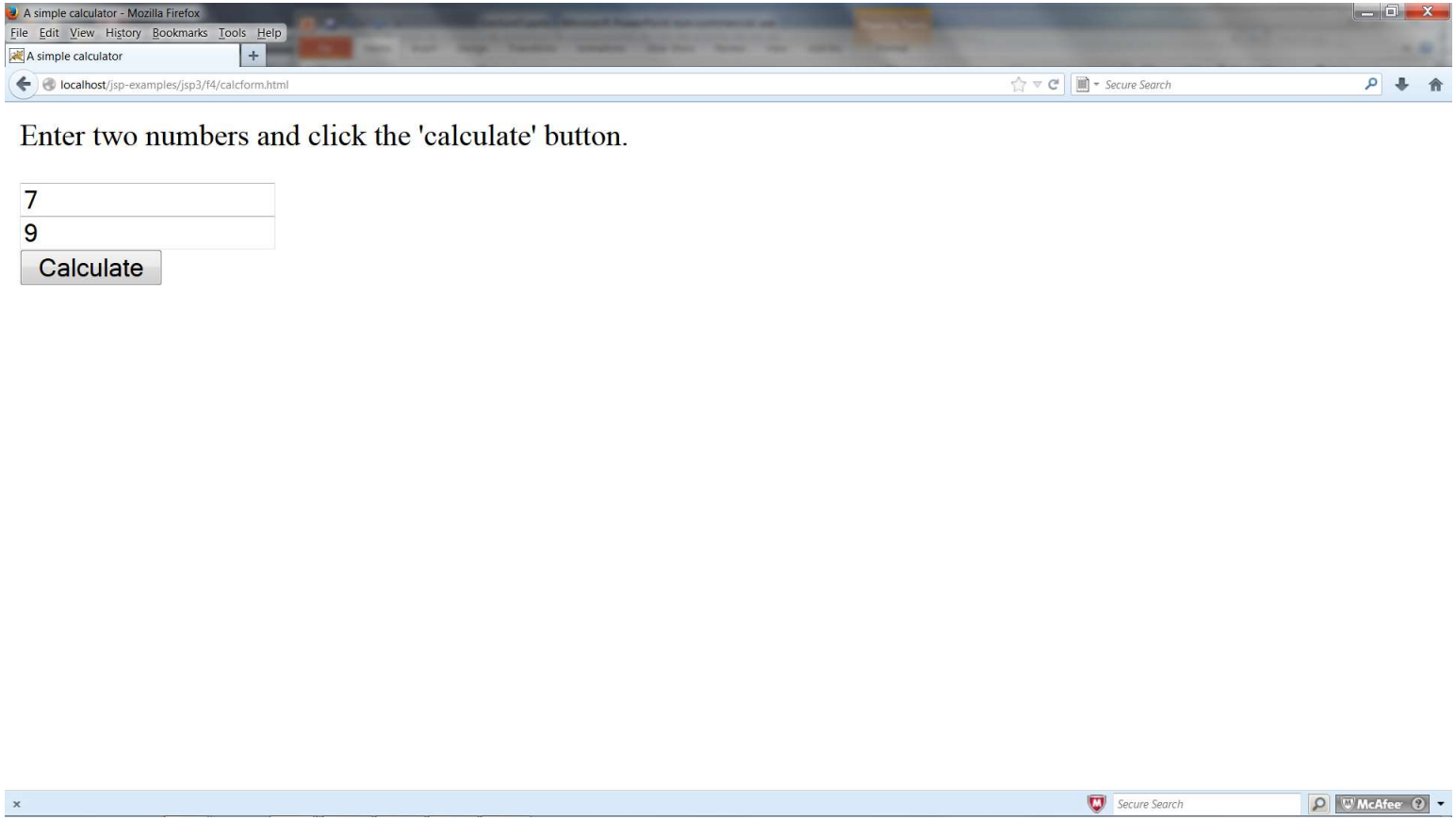
- Let us see how we can build a bean that can access variables from a form
- Notice how we can pass the whole form parameters through the use of the wild character.
- Compile and save your bean at:
  - **C:\apache-tomcat-7.0.34\webapps\jsp-examples\WEB-INF\classes\calculatorBean\CalcBean**
- Let us see how we can call a bean from jsp

- http://localhost/jsp-examples/jsp3/f4/calcform.html

Beans - Forms

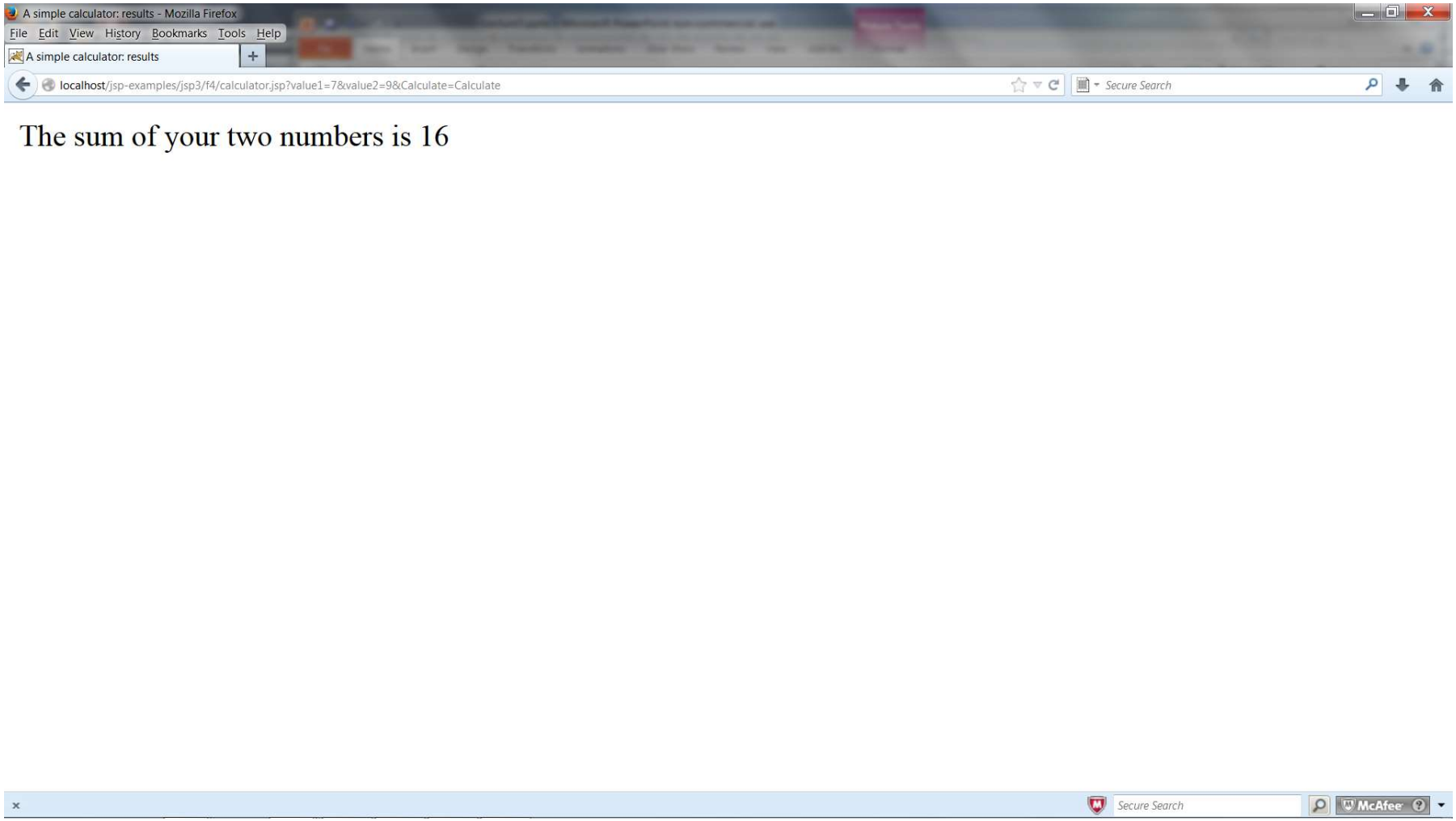
- Here is the page for:

  http://localhost/jsp-examples/jsp3/f4/calcform.html



Beans - Forms

- Here is the page after pressing calculate button:



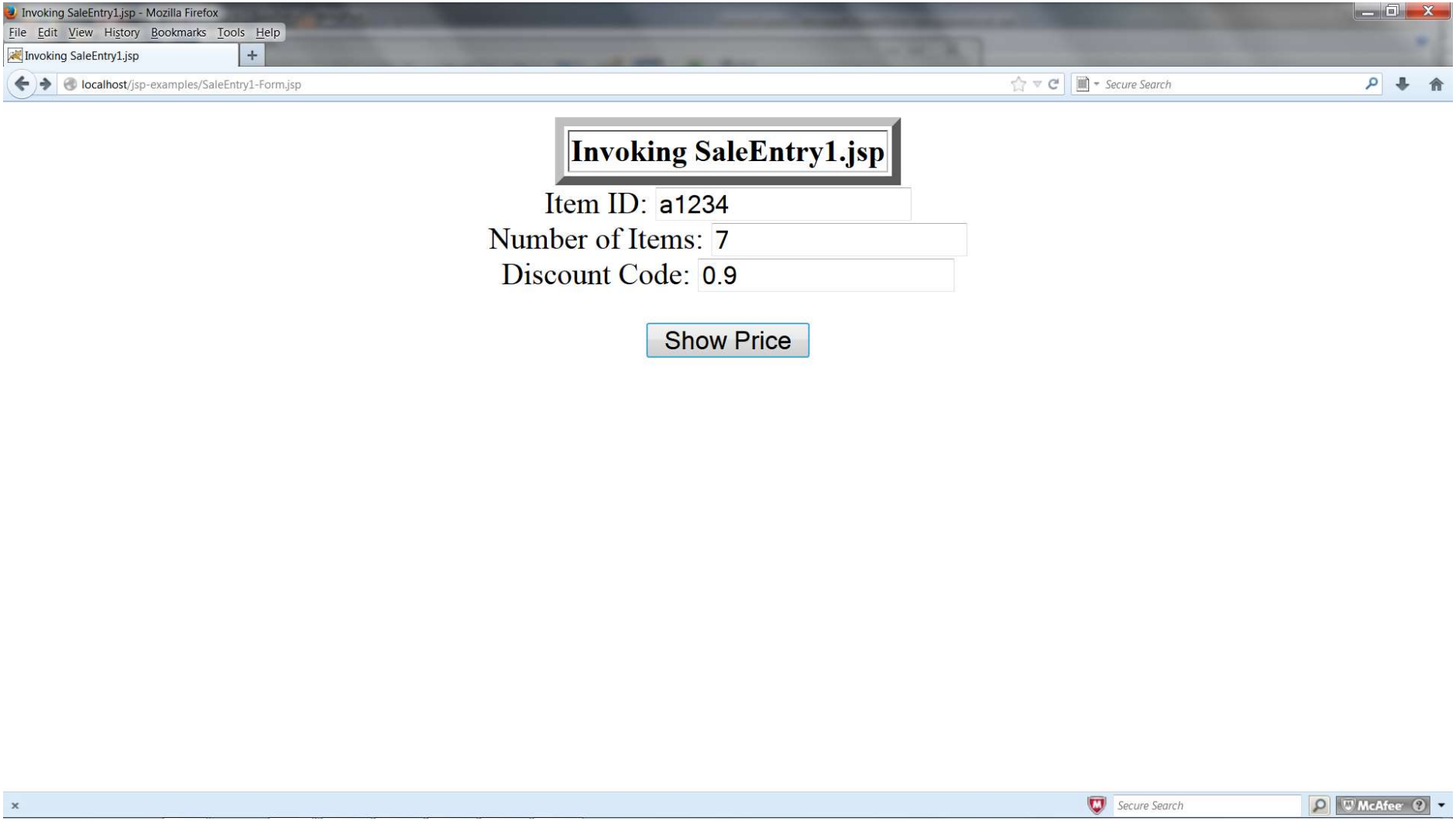
Setting Bean Properties:

- We use jsp:setProperty to set bean properties.

- jsp:setProperty takes three attributes:
  - **name** (which should match the id given by jsp:useBean),
  - **property** (the name of the property to change), and
  - **value** (the new value).

- Following example has:
  - http://localhost/jsp-examples/SaleEntry1-Form.jsp which calls
    - http://localhost/jsp-examples/SaleEntry1.jsp which calls
      - C:\apache-tomcat-7.0.34\webapps\jsp-examples\WEB-INF\classes\mybean\SaleEntry

Setting Bean Properties:



Setting Bean Properties:

Using jsp:setProperty - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Using jsp:setProperty

localhost/jsp-examples/SaleEntry1.jsp?itemID=a1234&numItems=7&discountCode=0.9

Secure Search

Using jsp:setProperty

Item ID	Unit Price	Number Ordered	Total Price
a1234	$11.69	7	$81.83

Secure Search

McAfee