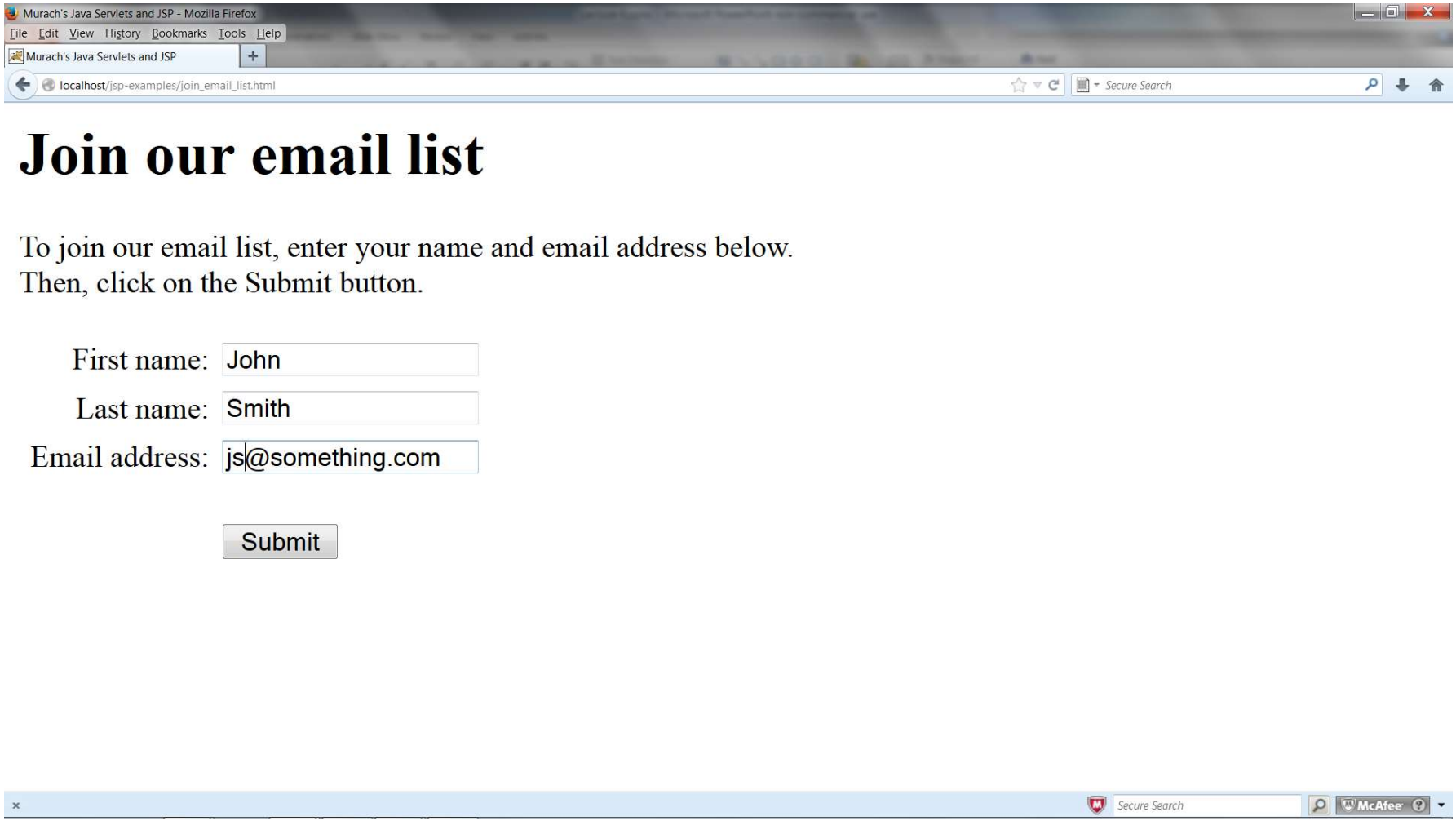
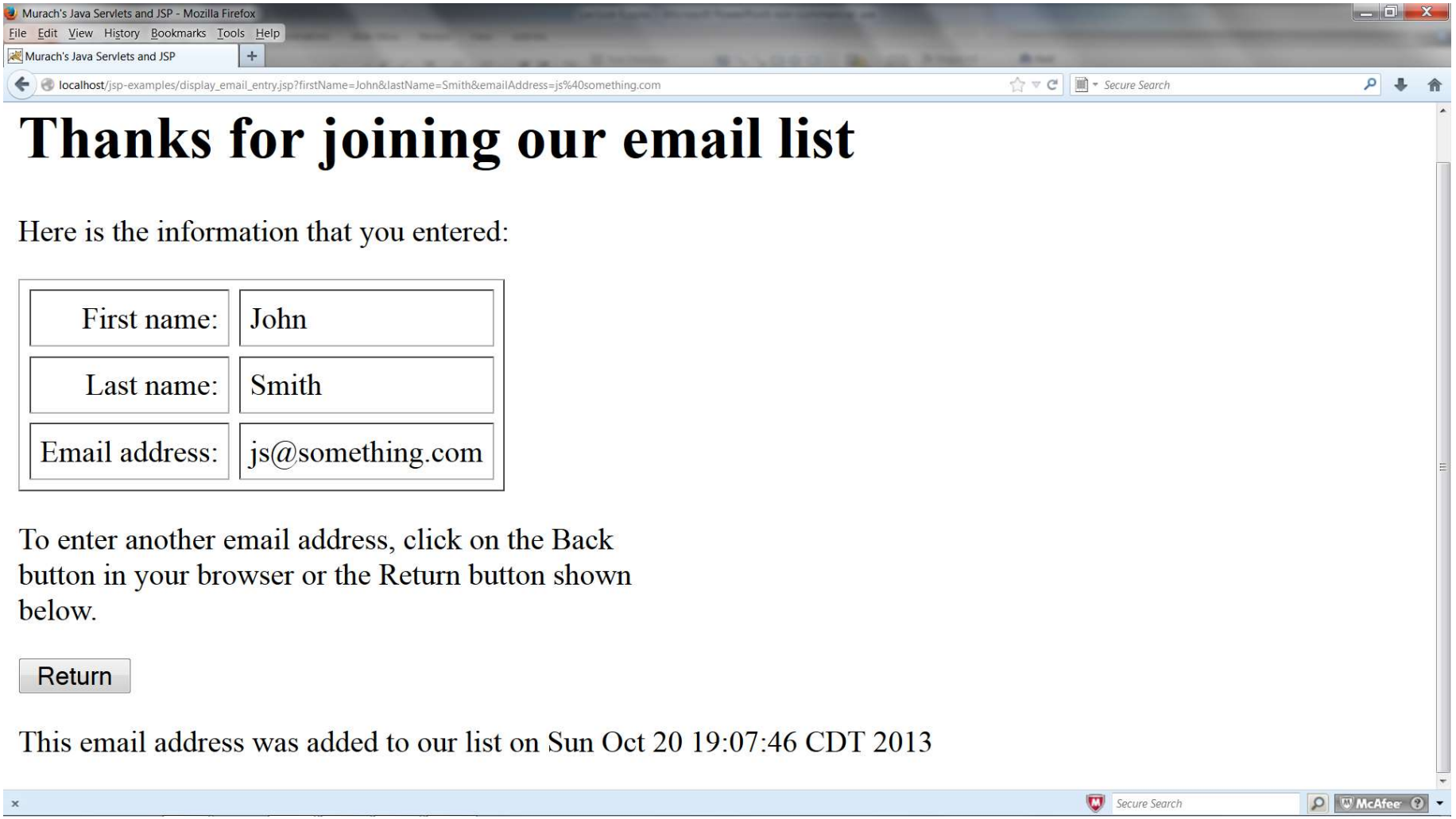
More on

JavaServer Pages

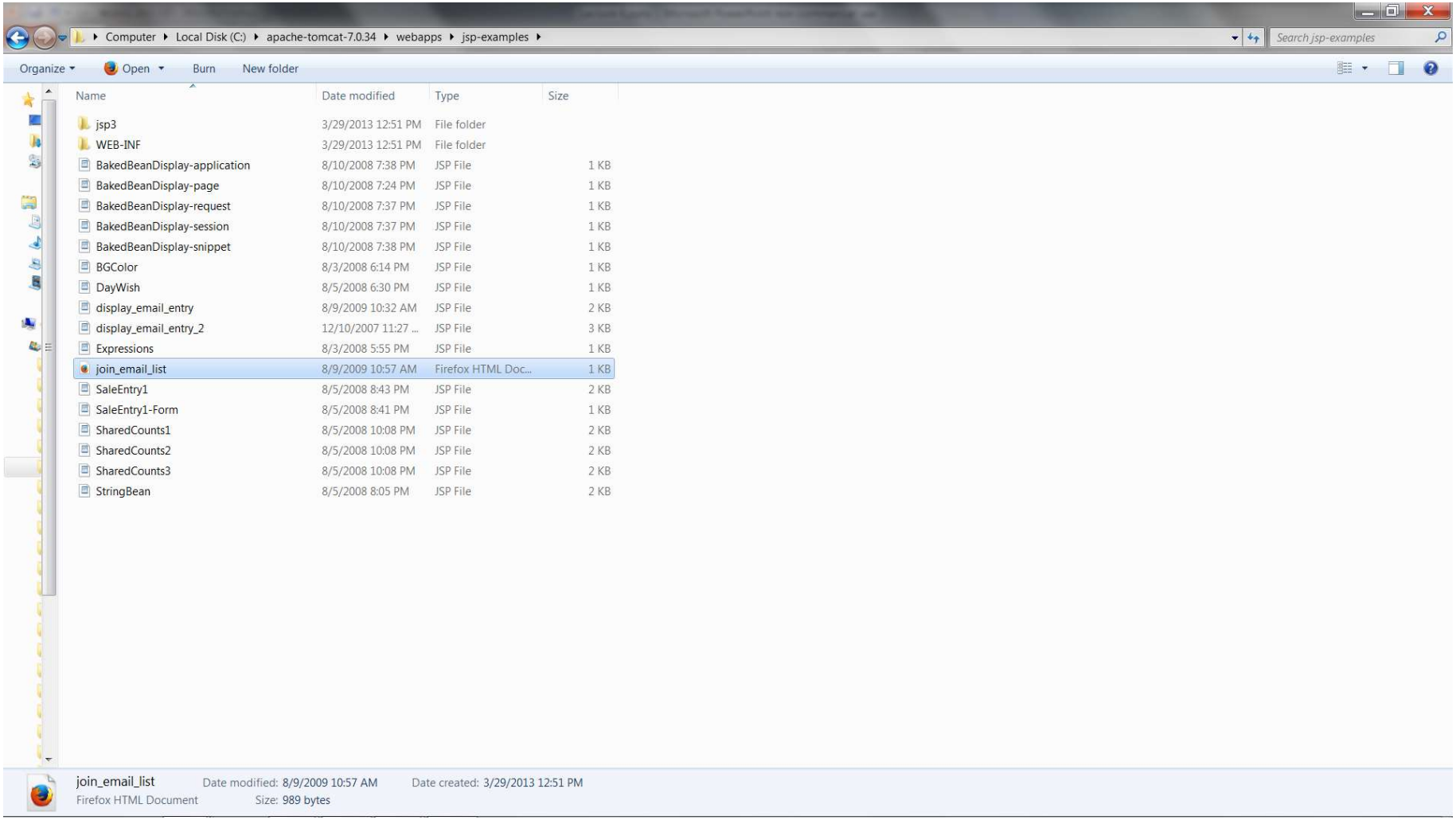
The HTML page for an Email List application



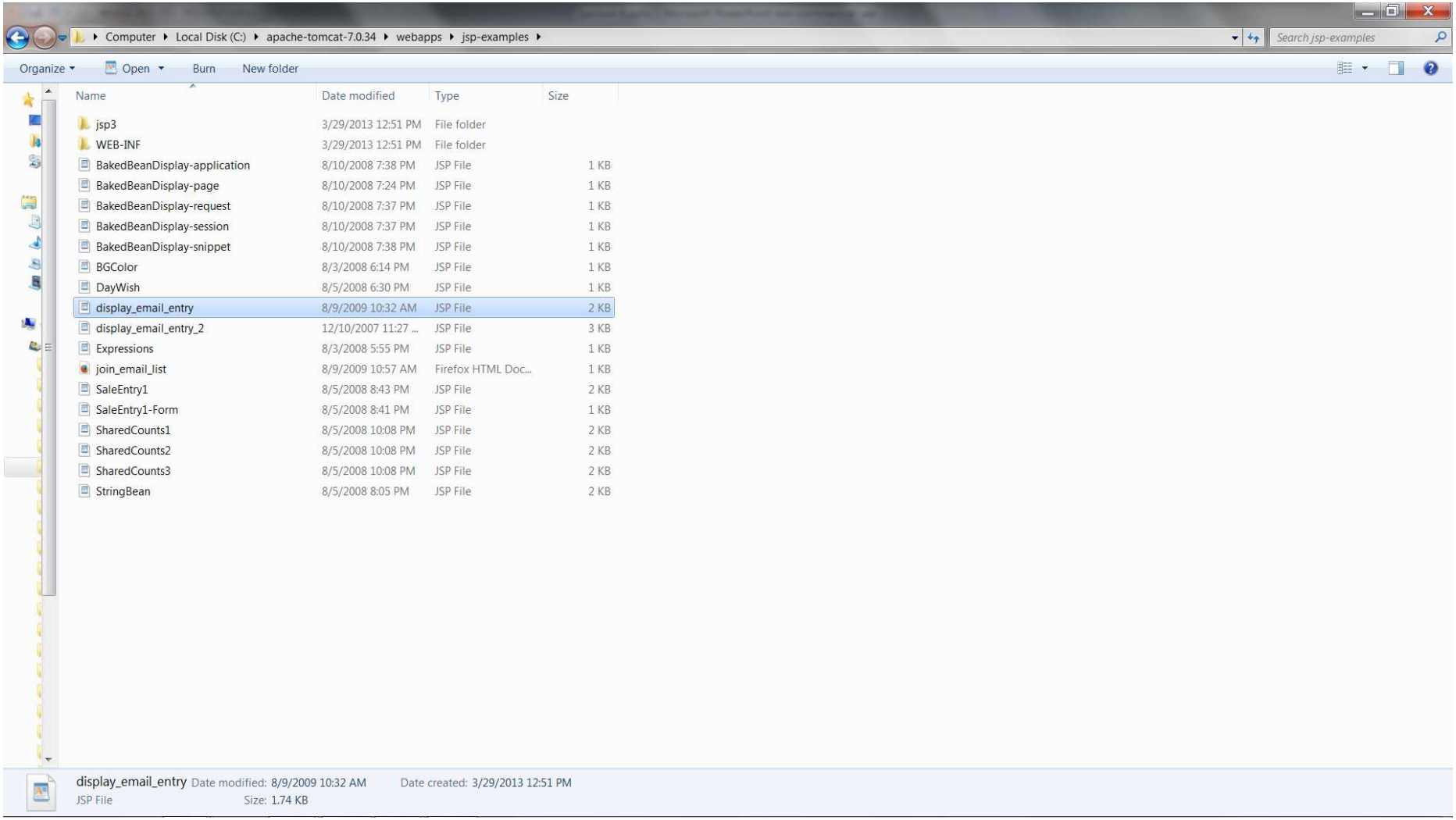
The JSP for an Email List application



The JSP for an Email List application



The JSP for an Email List application



The code for the HTML page

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>
    <title>Murach's Java Servlets and JSP</title>
</head>

<body>
    <h1>Join our email list</h1>
    <p>To join our email list, enter your name and
     email address below. <br>
     Then, click on the Submit button.</p>

    <form action="display_email_entry.jsp" method="get">
    <table cellspacing="5" border="0">
        <tr>
            <td align="right">First name:</td>
            <td><input type="text" name="firstName"></td>
        </tr>
```

The code for the HTML page (continued)

```
        <tr>
            <td align="right">Last name:</td>
            <td><input type="text" name="lastName"></td>
        </tr>
        <tr>
            <td align="right">Email address:</td>
            <td><input type="text" name="emailAddress"></td>
        </tr>
        <tr>
            <td></td>
            <td><br>
                <input type="submit" value="Submit"></td>
        </tr>
    </table>
    </form>
</body>

</html>
```

The code for the HTML page that calls the JSP

- The Action and Method attributes for the Form tag set up a request for a JSP that will be executed when the user clicks on the Submit button.
- The three text boxes represent *parameters* that will be passed to the JSP when the user clicks the Submit button.

The code for the JSP

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
    <title>Murach's Java Servlets and JSP</title>
</head>
<body>
    <%
        // get parameters from the request
        String firstName =
            request.getParameter("firstName");
        String lastName = request.getParameter("lastName");
        String emailAddress =
            request.getParameter("emailAddress");
    %>

    <h1>Thanks for joining our email list</h1>

    <p>Here is the information that you entered:</p>
```

The code for the JSP (continued)

```
<table cellspacing="5" cellpadding="5" border="1">
    <tr>
        <td align="right">First name:</td>
        <td><%= firstName %></td>
    </tr>
    <tr>
        <td align="right">Last name:</td>
        <td><%= lastName %></td>
    </tr>
    <tr>
        <td align="right">Email address:</td>
        <td><%= emailAddress %></td>
    </tr>
</table>

<p>To enter another email address, click on the Back <br
button in your browser or the Return button shown <br>
below.</p>
```

The code for the JSP (continued)

```
    <form action="join_email_list.html" method="post">
        <input type="submit" value="Return">
    </form>

</body>
</html>
```

The code for a JSP

- A JSP contains HTML tags and embedded Java code.
- To code a *scriptlet* that contains one or more Java statements, you use the <% and %> tags.
- To code an *expression* that can be converted to a string, you use the <%= and %> tags.
- To get the values of the parameters that are passed to the JSP, you can use the getParameter method of *implicit request object* named request.

The syntax for a JSP scriptlet

```
<% Java statements %>
```

The syntax for a JSP expression

```
<%= any Java expression that can be converted to a string %>
```

The syntax for getting a parameter from the implicit request object

```
request.getParameter(parameterName);
```

A scriptlet and expression that display the value of the firstName parameter

```
<%
    String firstName = request.getParameter("firstName");
%>
The first name is <%= firstName %>.
```

An expression that displays the value of the firstName parameter

```
The first name is <%= request.getParameter("firstName") %>.
```

Two scriptlets and an expression that display an HTML line 5 times

```
<%
    int numOfTimes = 1;
    while (numOfTimes <= 5)
    {
%>
    <h1>This line is shown <%= numOfTimes %>
        of 5 times in a JSP.</h1>
<%
        numOfTimes++;
    }
%>
```

How to code scriplets and expressions

- Within a scriptlet, you can code one or more Java statements. You must end each Java statement with a semicolon.
- Within a JSP expression, you can code any Java expression that evaluates to a Java object or to a primitive type. Since an expression isn't a statement, you don't end it with a semicolon.

Three methods available from the request object

Method	Description
`getParameter(` `  String param)`	Returns the value of the specified parameter as a string if it exists or null if it doesn't.
`getParameterValues(` `  String param)`	Returns an array of String objects containing all of the values that the given request parameter has or null if the parameter doesn't have any values.
`getParameterNames()`	Returns an Enumeration object that contains the names of all the parameters contained in the request. If the request has no parameters, the method returns an empty Enumeration object.

A scriptlet that determines if a checkbox is checked

```
<%
    // returns the value or "on" if checked, null otherwise.
    String rockCheckBox = request.getParameter("Rock");
    if (rockCheckBox != null)
    {
%>
        You checked Rock music!
<%
    }
%>
```

A scriptlet that reads and displays multiple values from a list box

```
<%
    // returns the values of items selected in a list box.
    String[] selectedCountries =
        request.getParameterValues("country");
    for (int i = 0; i < selectedCountries.length; i++)
    {
%>
        <%= selectedCountries[i] %> <br>
<%
    }
%>
```

A scriptlet that reads and displays all request parameters and values

```
<%
    Enumeration parameterNames =
        request.getParameterNames();
    while (parameterNames.hasMoreElements())
    {
        String parameterName = (String)
            parameterNames.nextElement();
        String parameterValue =
            request.getParameter(parameterName);
%>
        <%= parameterName %> has value
            <%= parameterValue %>. <br>
<%
     }
%>
```

A method of the GenericServlet class

Method	Description
`getServletContext()`	Returns a ServletContext object that contains information about the application's context.

A method of the ServletContext class for working with paths

Method	Description
`getRealPath(String path)`	Returns a String object for the real path of the specified relative path.

Code that gets the real path for a file

```
ServletContext sc = this.getServletContext();
String path = sc.getRealPath("/WEB-INF/EmailList.txt");
```

The value for the real path variable

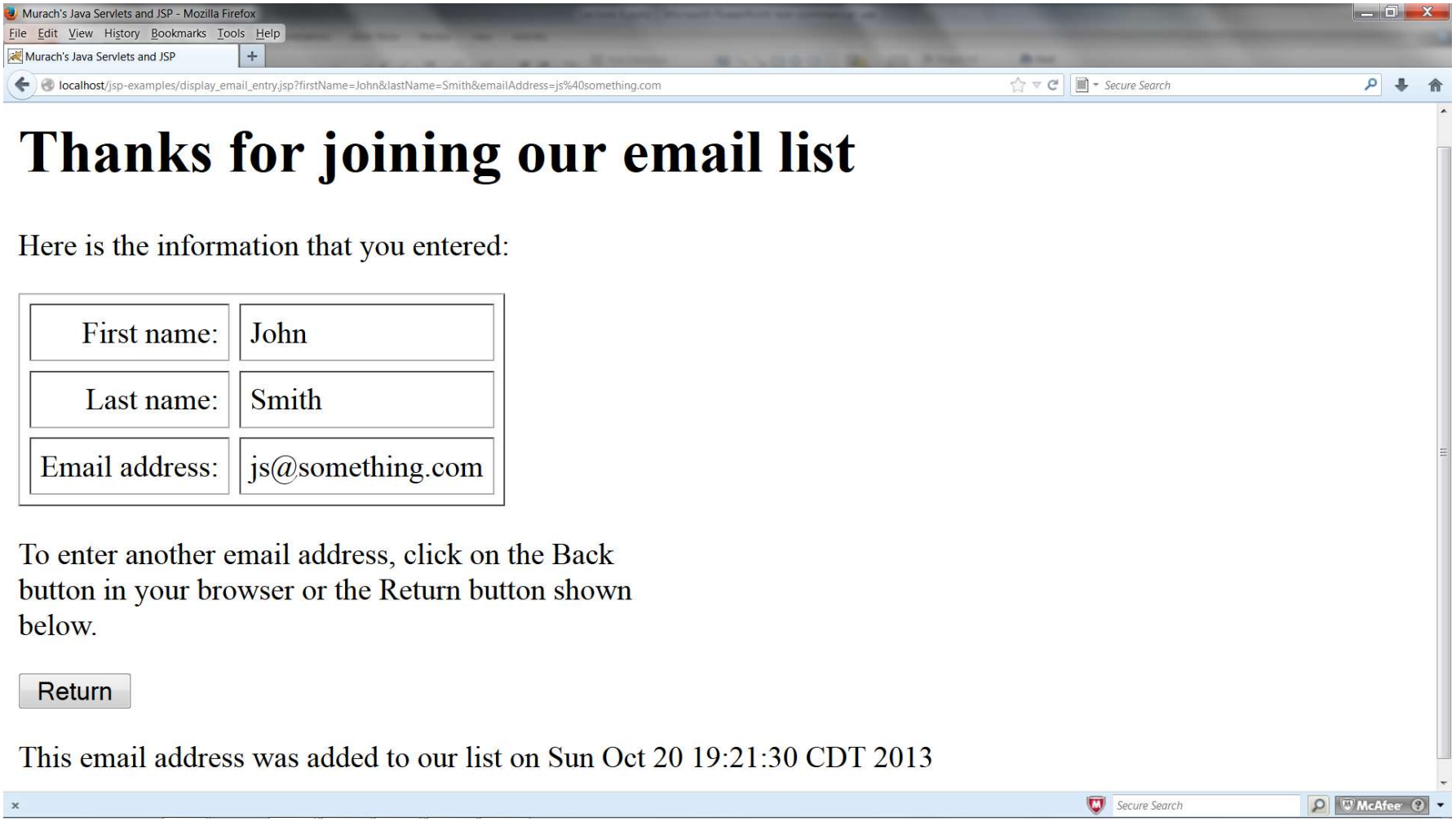
```
c:\apache-tomcat-7.0.34\webapps\jsp-examples\WEB-INF>dir
 Volume in drive C has no label.
 Volume Serial Number is 289B-5690

 Directory of c:\apache-tomcat-7.0.34\webapps\jsp-examples\WEB-INF

03/29/2013  12:51 PM    <DIR>          .
03/29/2013  12:51 PM    <DIR>          ..
03/29/2013  12:51 PM    <DIR>          classes
10/20/2013  07:07 PM               203 EmailList.txt
08/09/2009  06:45 PM             1,511 web.xml
               2 File(s)          1,714 bytes
               3 Dir(s)  467,950,661,632 bytes free

c:\apache-tomcat-7.0.34\webapps\jsp-examples\WEB-INF>
```

A JSP that's requested with the HTTP Get method



Two Form tags that use the Get method

```
<form action="display_email_entry.jsp">
<form action="display_email_entry.jsp" method="get">
```

How to append parameters to a request

```
display_email_entry.jsp?firstName=John
display_email_entry.jsp?firstName=John&lastName=Smith
```

An Anchor tag that requests a JSP with the Get method

```
<a href="display_email_entry.jsp?firstName=John&lastName=Smith">
    Display Email Entry Test
</a>
```

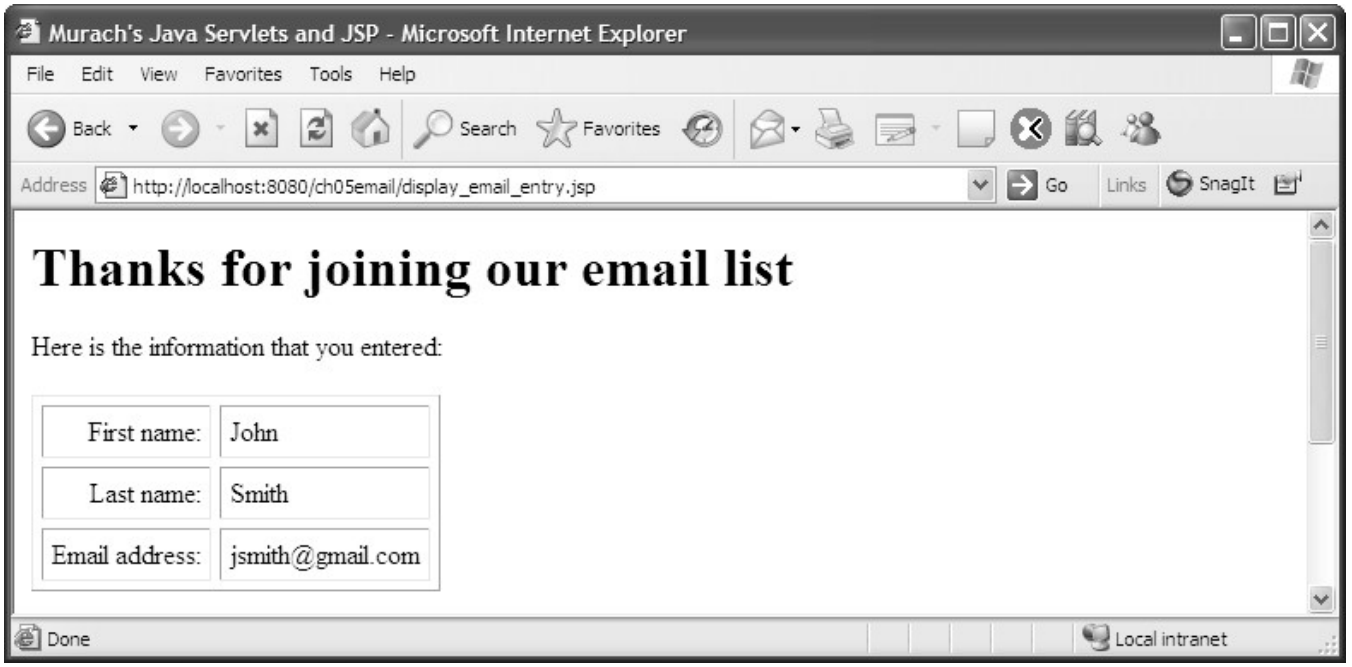
Two URLs that request a JSP with the Get method

```
http://localhost:8080/ch05email/display_email_entry.jsp?firstName=Joh
http://www.murach.com/email/display_email_entry.jsp?firstName=John
```

How to request a JSP with the HTTP Get method

- When you use the HTTP Get method to request a JSP from an HTML form, the parameters are automatically appended to the URL.
- When you code or enter a URL that requests a JSP, you can add a parameter list to it starting with a question mark and with no intervening spaces. Then, each parameter consists of its name, an equals sign, and its value.
- To code multiple parameters, use ampersands (&) to separate the parameters.
- The A tag always uses the HTTP Get method.

A JSP that's requested with the HTTP Post method



A Form tag that uses the Post method

```
<form action="display_email_entry.jsp" method="post">
```

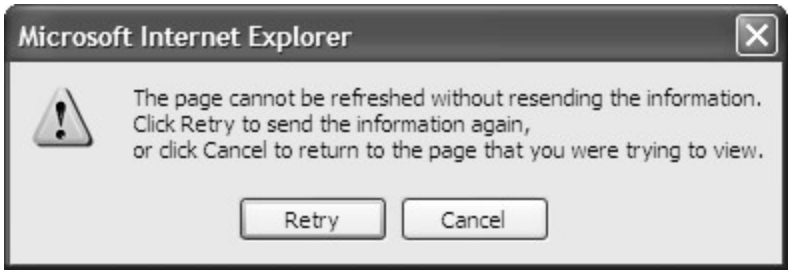
When to use the Get method

- When the request reads data from the server.
- When the request doesn't include private/personal data.

When to use the Post method

- When the request writes data to the server.
- When you don't want to include the parameters in the URL for security reasons.
- When you don't want users to be able to include parameters when they bookmark a page.
- When you need to transfer more than 4 KB of data.

A typical browser dialog that's displayed if the user tries to refresh a post



The code for the User class

```
package business;

public class User
{
    private String firstName;
    private String lastName;
    private String emailAddress;

    public User()
    {
        firstName = "";
        lastName = "";
        emailAddress = "";
    }
```

The code for the User class (continued)

```
public User(String firstName, String lastName,
String emailAddress)
{
    this.firstName = firstName;
    this.lastName = lastName;
    this.emailAddress = emailAddress;
}

public void setFirstName(String firstName)
{
    this.firstName = firstName;
}

public String getFirstName()
{
    return firstName;
}
```

The code for the User class (continued)

```
    public void setLastName(String lastName)
    {
        this.lastName = lastName;
    }

    public String getLastName()
    {
        return lastName;
    }

    public void setEmailAddress(String emailAddress)
    {
        this.emailAddress = emailAddress;
    }

    public String getEmailAddress()
    {
        return emailAddress;
    }
}
```

The code for the UserIO class

```
package data;

import java.io.*;
import business.User;

public class UserIO
{
    public static void add(User user, String filepath)
    throws IOException
    {
        File file = new File(filepath);
        PrintWriter out = new PrintWriter(
                new FileWriter(file, true));
        out.println(user.getEmailAddress()+ "|"
                + user.getFirstName() + "|"
                + user.getLastName());
        out.close();
    }
}
```

The code for a JSP that uses the User and UserIO classes

```
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
    <title>Murach's Java Servlets and JSP</title>
</head>
<body>
    <!-- import packages and classes needed by scripts -->
    <%@ page import="business.*, data.*" %>

    <%
        // get parameters from the request
        String firstName =
            request.getParameter("firstName");
        String lastName = request.getParameter("lastName");
        String emailAddress =
            request.getParameter("emailAddress");
```

The code for the JSP (continued)

```
        // get the real path for the EmailList.txt file
        ServletContext sc = this.getServletContext();
        String path =
            sc.getRealPath("/WEB-INF/EmailList.txt");

        // use regular Java objects
        User user = new User(firstName, lastName,
            emailAddress);
        UserIO.add(user, path);
    %>

    <h1>Thanks for joining our email list</h1>

    <p>Here is the information that you entered:</p>

    <table cellspacing="5" cellpadding="5" border="1">
        <tr>
            <td align="right">First name:</td>
            <td><%= user.getFirstName() %></td>
        </tr>
```

The code for the JSP (continued)

```
        <tr>
            <td align="right">Last name:</td>
            <td><%= user.getLastName() %></td>
        </tr>
        <tr>
            <td align="right">Email address:</td>
            <td><%= user.getEmailAddress() %></td>
        </tr>
    </table>

    <p>To enter another email address, click on the Back <br
    button in your browser or the Return button shown <br>
    below.</p>

    <form action="join_email_list.html" method="post">
        <input type="submit" value="Return">
    </form>

</body>
</html>
```

The five types of JSP tags

Tag	Name	Purpose
`<% %>`	JSP scriptlet	To insert a block of Java statements.
`<%= %>`	JSP expression	To display the string value of an expression.
`<%@ %>`	JSP directive	To set conditions that apply to the entire JSP.
`<%-- --%>`	JSP comment	To tell the JSP engine to ignore code.
`<%! %>`	JSP declaration	To declare instance variables and methods for a JSP.

JSP code that imports Java classes

```
<%@ page import="business.*, data.*, java.util.Date" %>
<%
    // get parameters from the request
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");
    String emailAddress =
        request.getParameter("emailAddress");

    // get a relative file name
    ServletContext sc = this.getServletContext();
    String path =
        sc.getRealPath("/WEB-INF/EmailList.txt");

    // use regular Java objects
    User user =
        new User(firstName, lastName, emailAddress);
    UserIO.add(user, path);
%>

<p>This email address was added to our list on
    <%= new Date() %></p>
```

How to import classes

- To define the conditions that the JSP engine should follow when converting a JSP into a servlet, you can use a *JSP directive*.
- To import classes in a JSP, you use the import attribute of the *page directive*. This makes the imported classes available to the entire page.

An HTML comment in a JSP

```
<!--
<p>This email address was added to our list on
    <%= new Date() %></p>
-->
```

A JSP comment

```
<%--
<p>This email address was added to our list on
    <%= new Date() %></p>
--%>
```

Java comments in a JSP scriptlet

```
<%
    // get parameters from the request
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");
    String emailAddress =
        request.getParameter("emailAddress");

    /*
    User user =
        new User(firstName, lastName, emailAddress);
    UserIO.add(user, path);
    */
%>
```

How to code comments in a JSP

- When you code HTML comments, the comments are compiled and executed, but the browser doesn't display them.
- When you code *JSP comments*, the comments aren't compiled or executed.
- When you code Java comments within a scriptlet, the comments aren't compiled or executed.

JSP code that declares an instance variable and a method

```
<%-- import any packages needed by the page --%>
<%@ page import="business.*, data.*, java.util.Date,
    java.io.*" %>
<%!
    // declare an instance variable for the page
    int globalCount = 0;  // this is not thread-safe
%>
<%!
    // declare a method for the page
    public void add(User user, String filename)
            throws IOException
    {
        PrintWriter out = new PrintWriter(
                new FileWriter(filename, true));
        out.println(user.getEmailAddress()+ "|"
                + user.getFirstName() + "|"
                + user.getLastName());
        out.close();
    }
%>
```

JSP code that declares an instance variable and a method (continued)

```
<%
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");
    String emailAddress =
        request.getParameter("emailAddress");

    ServletContext sc = getServletContext();
    String path = sc.getRealPath("/WEB-INF/EmailList.txt");

    User user = new User(firstName, lastName, emailAddress);

    // use the declared method
    this.add(user, path);

    // update the instance variable
    globalCount++;  // this is not thread-safe
%>
.
.
.
```

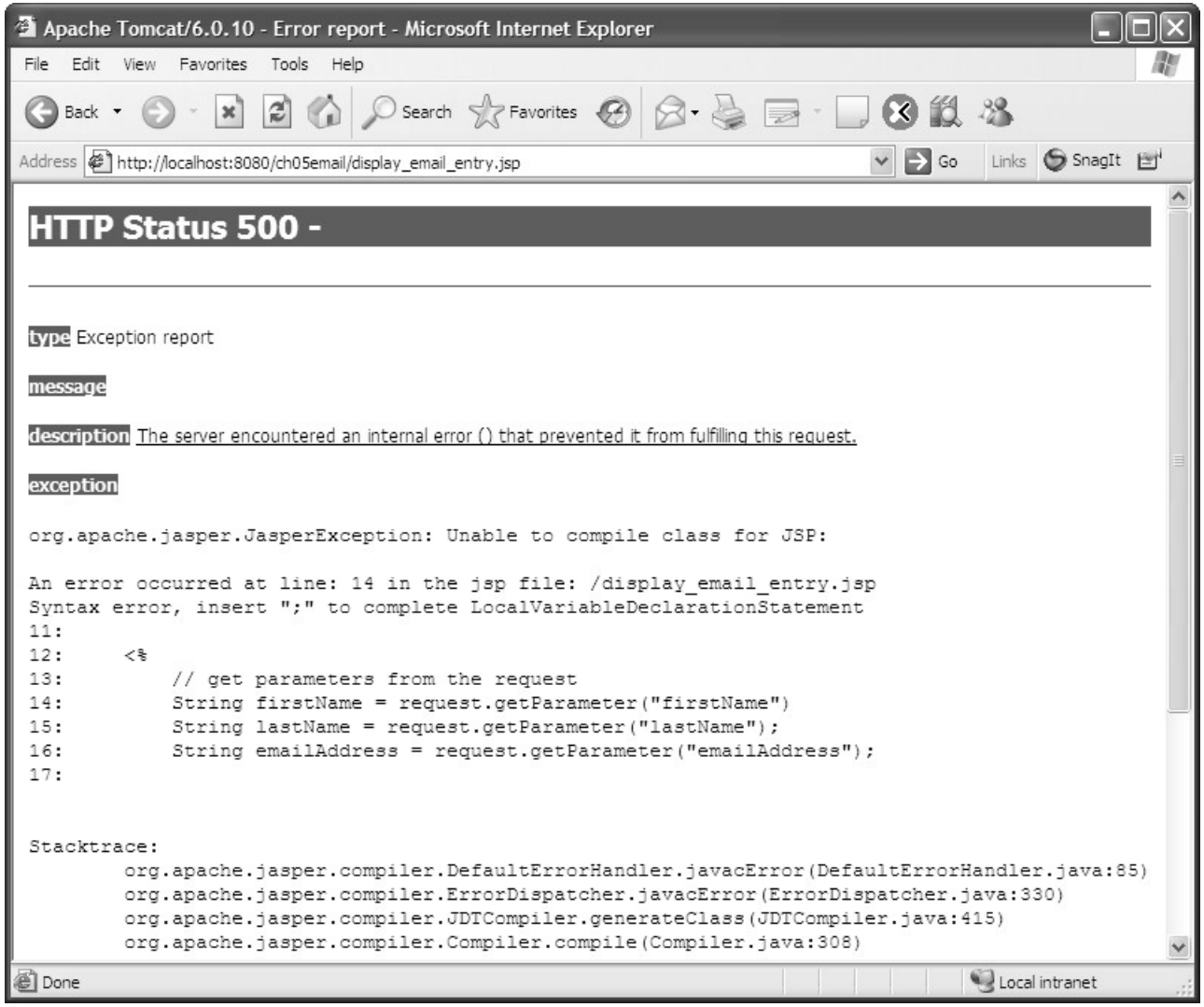
JSP code that declares an instance variable and a method (continued)

```
<p>
    This email address was added to our list on
        <%= new Date() %><br>
    This page has been accessed <%= globalCount %> times.
</p>
```

How to declare instance variables and methods

- You can use *JSP declarations* to declare instance variables and methods.
- Since instance variables aren't *thread-safe*, two threads may conflict when they try to read, modify, and update the same instance variable at the same time.
- In general, you should avoid coding instance variables for JSPs and servlets. Instead, you should use other thread-safe techniques for working with global variables.
- In general, you should avoid coding methods within JSPs. Instead, you should use some combination of JSPs, servlets, and regular Java classes.

An error page for a common JSP error



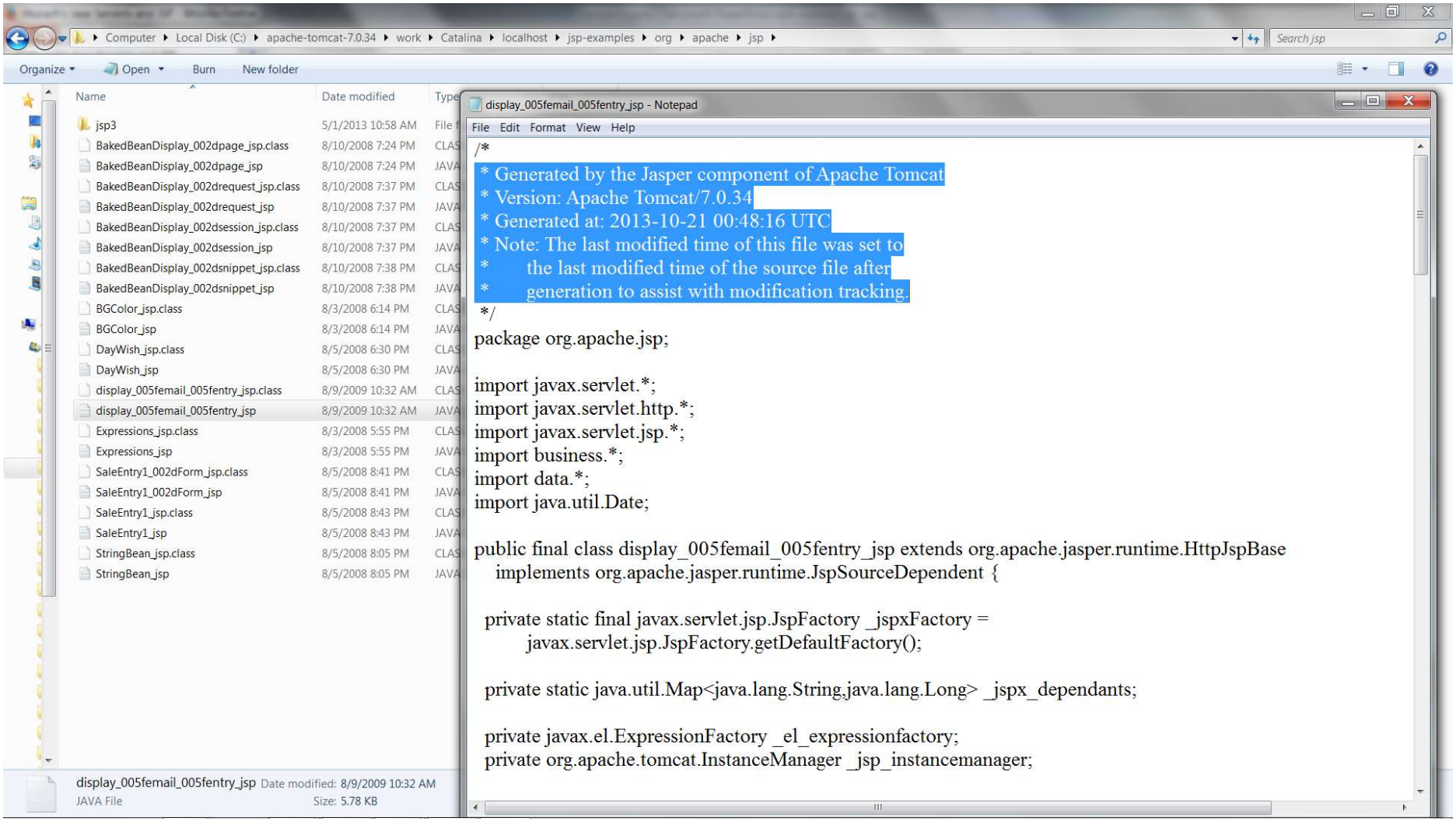
Common JSP errors

- HTTP Status 404 – File Not Found Error
- HTTP Status 500 – Internal Server Error

Tips for fixing JSP errors

- Make sure the Tomcat server is running.
- Make sure that the URL is valid and that it points to the right location for the requested page.
- Make sure all of the HTML, JSP, and Java class files are in the correct locations.
- Read the error page carefully to get all available information about the error.

Part of the servlet class that's generated from the JSP for the Email List application



The JSP work directory for email application

```
C:\apache-tomcat-7.0.34\work\Catalina\localhost\jsp-
examples\org\apache\jsp
```

Part of the servlet class that's generated from the

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import business.*;
import data.*;
import java.util.Date;

public final class display_005femail_005fentry_jsp extends
org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent
{

  private static final javax.servlet.jsp.JspFactory
_jspxFactory =
```

Part of the servlet class (continued)

```
public void _jspService(HttpServletRequest request,
HttpServletResponse response)
throws java.io.IOException, ServletException {
    ...
    response.setContentType("text/html");
    ...
    out.write("<html>\n");
    out.write("<head>\n");
    out.write("    <title>Murach's Java Servlets and
        JSP</title>\n");
    out.write("</head>\n");
    out.write("<body>\n");
    ...
      // get parameters from the request
      String firstName =
          request.getParameter("firstName");
      String lastName = request.getParameter("lastName");
      String emailAddress =
          request.getParameter("emailAddress");
```

Part of the servlet class (continued)

```
        // get the real path for the emaillist file
        ServletContext sc = this.getServletContext();
        String path =
            sc.getRealPath("/WEB-INF/EmailList.txt");

        // use regular Java objects
        User user =
            new User(firstName, lastName, emailAddress);
        UserIO.add(user, path);
    ...
    out.write("    <table cellspacing=\"5\"
                   cellpadding=\"5\"
                   border=\"1\">\n");
    out.write("        <tr>\n");
    out.write("            <td align=\"right\">
                           First name:</td>\n");
    out.write("            <td>");
    out.print( user.getFirstName() );
    out.write("</td>\n");
    out.write("        </tr>\n");
    ...
```

Part of the servlet class (continued)

```
        out.write("</body>\n");
        out.write("</html>");
        ...
    }
}
```

Handover and Redirects between JSP and Servlet

Bean Scopes

- How to use the same bean in multiple pages
- We want to extend the lifespan of a Bean
- Bottom Line we want to move data from a servlet to a JSP and vice versa for a number of good reasons

JSP has 4 types of Bean Scopes

- The Page Scope
- The Request Scope
- The Session Scope
- The Application Scope

The Page Scope

- In the Page Scope, any object whose scope is the page will disappear as soon as the current page finishes generating.

- The Bean properties can be altered but as soon as a next person accesses the page, the old values will be displayed.

- By default a bean has a page scope

<jsp:useBean id = “id” class=“beanClass” scope=“page”/>

The Request Scope

- It may look similar to Page scope, but it is different.

- The intent is to pass this object through the use of the JSP construct forward to another JSP page

<jsp:useBean id = “id” class=“beanClass” scope=“request”/>

The Session Scope

- Previous scopes associate data with pages

- The sessions scope associates data to users regardless of the pages that a user may visit. An obvious example is the shopping cart.

- Multiple users may see the same checkout page, but each user will see his own selections.

<jsp:useBean id = “id” class=“beanClass” scope=“session”/>

The Application Scope

- Scope session valid only within a single page.

- The application scope is used to retain data across multiple pages as long as the same bean id is used

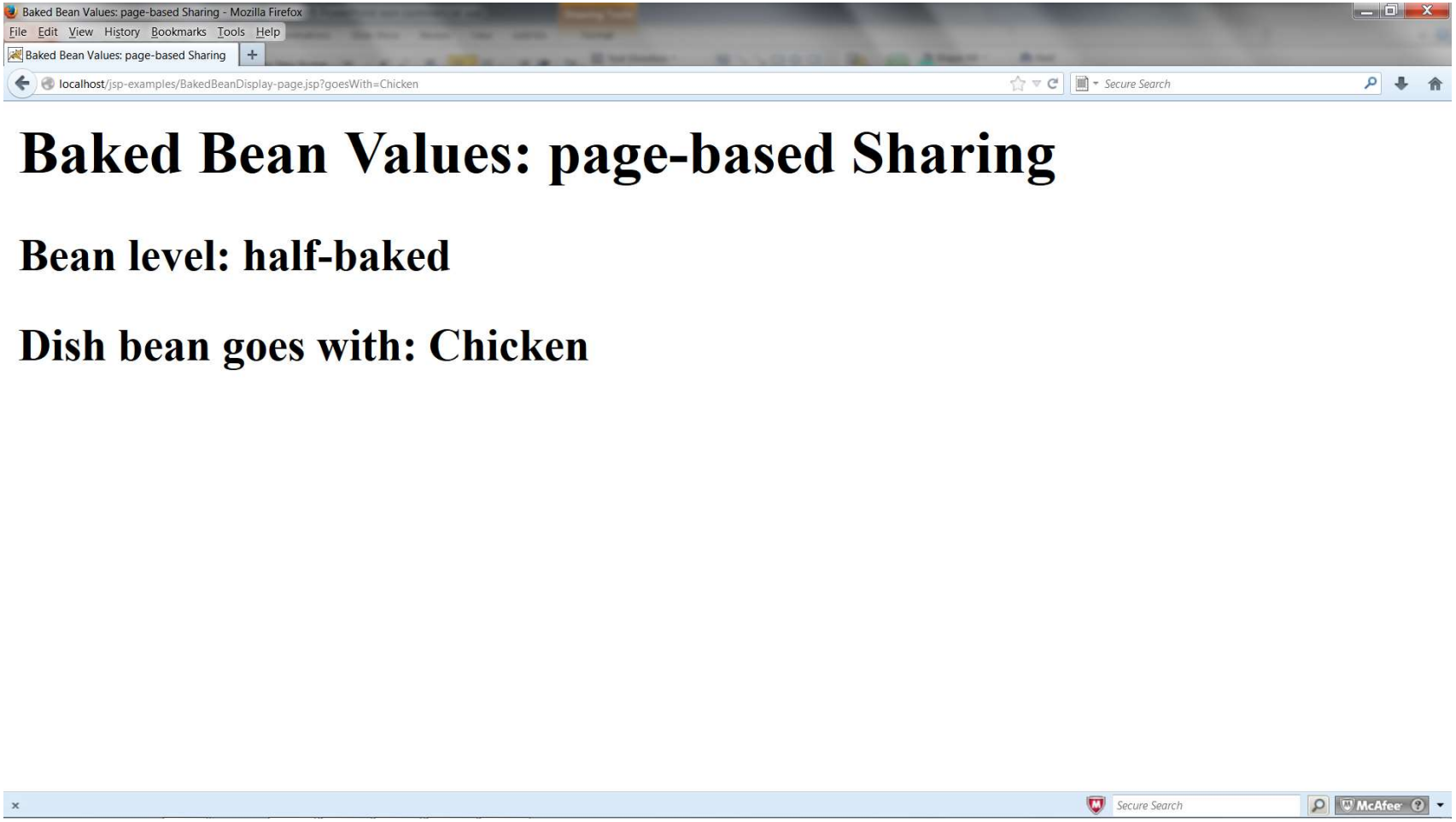
- The application scope is needed to control the data that multiple users will see on multiple pages.

- The bean will be stored in the ServletContext. The ServletContext is shared by all servlets and JSP pages in the Web application.

<jsp:useBean id = “id” class=“beanClass” scope=“application”/>

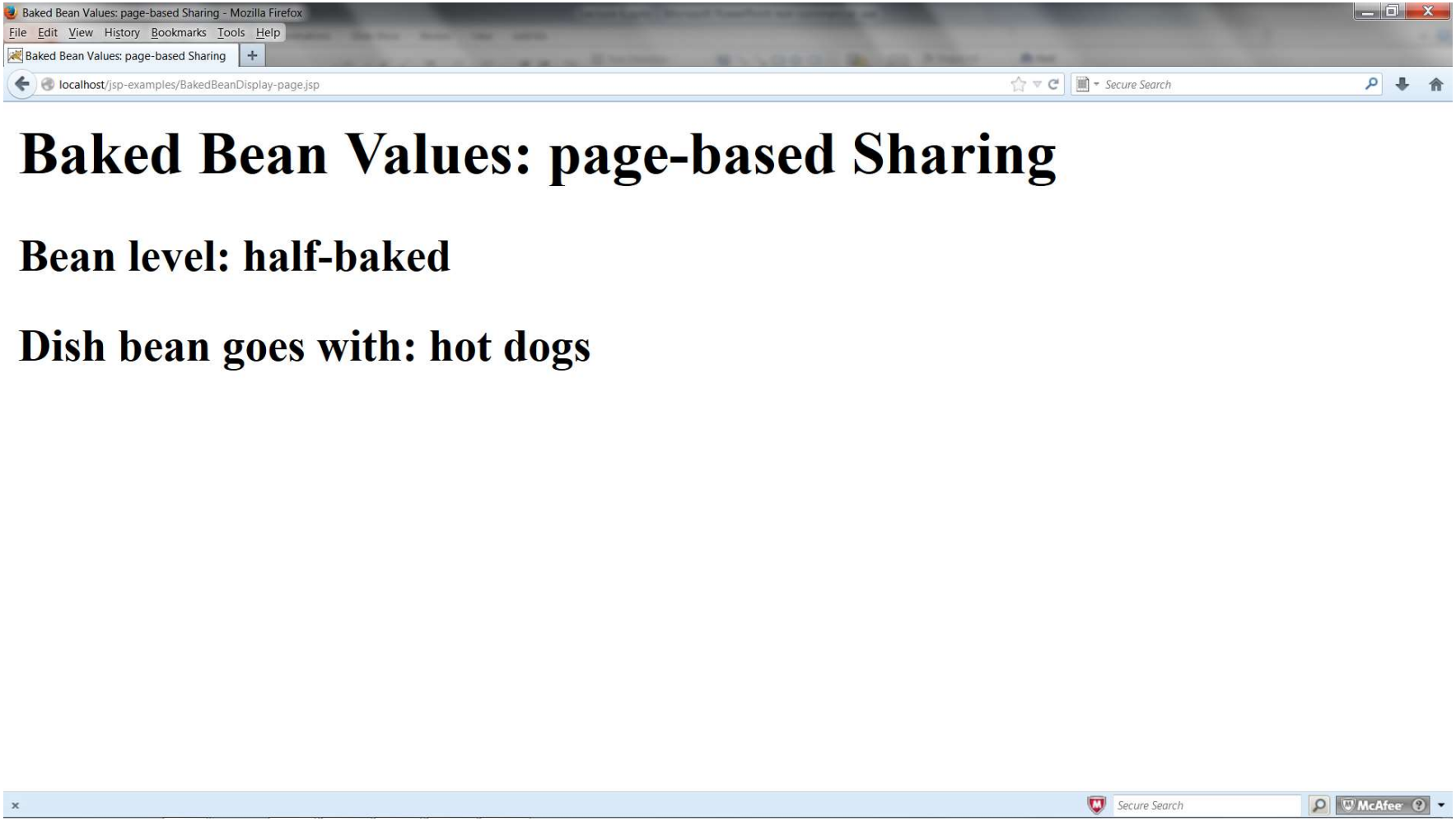
Using scope="page"—No Sharing

- Initial request to BakedBeanDisplay-page.jsp BakedBean properties persist within the page. http://localhost/jsp-examples/BakedBeanDisplay-page.jsp



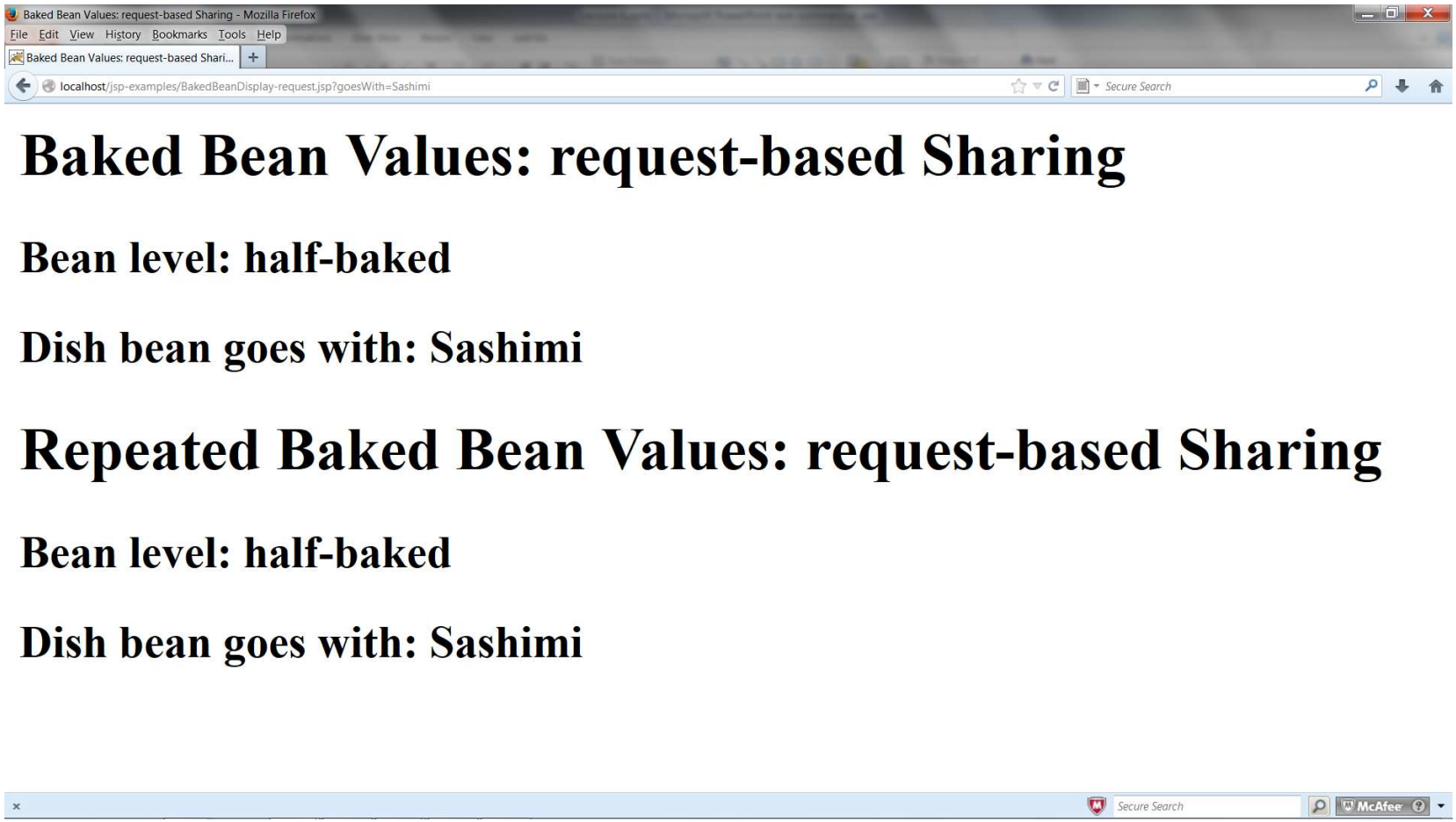
Using scope="page"—No Sharing

- Subsequent request to BakedBeanDisplay-page.jsp—BakedBean properties do not persist between requests



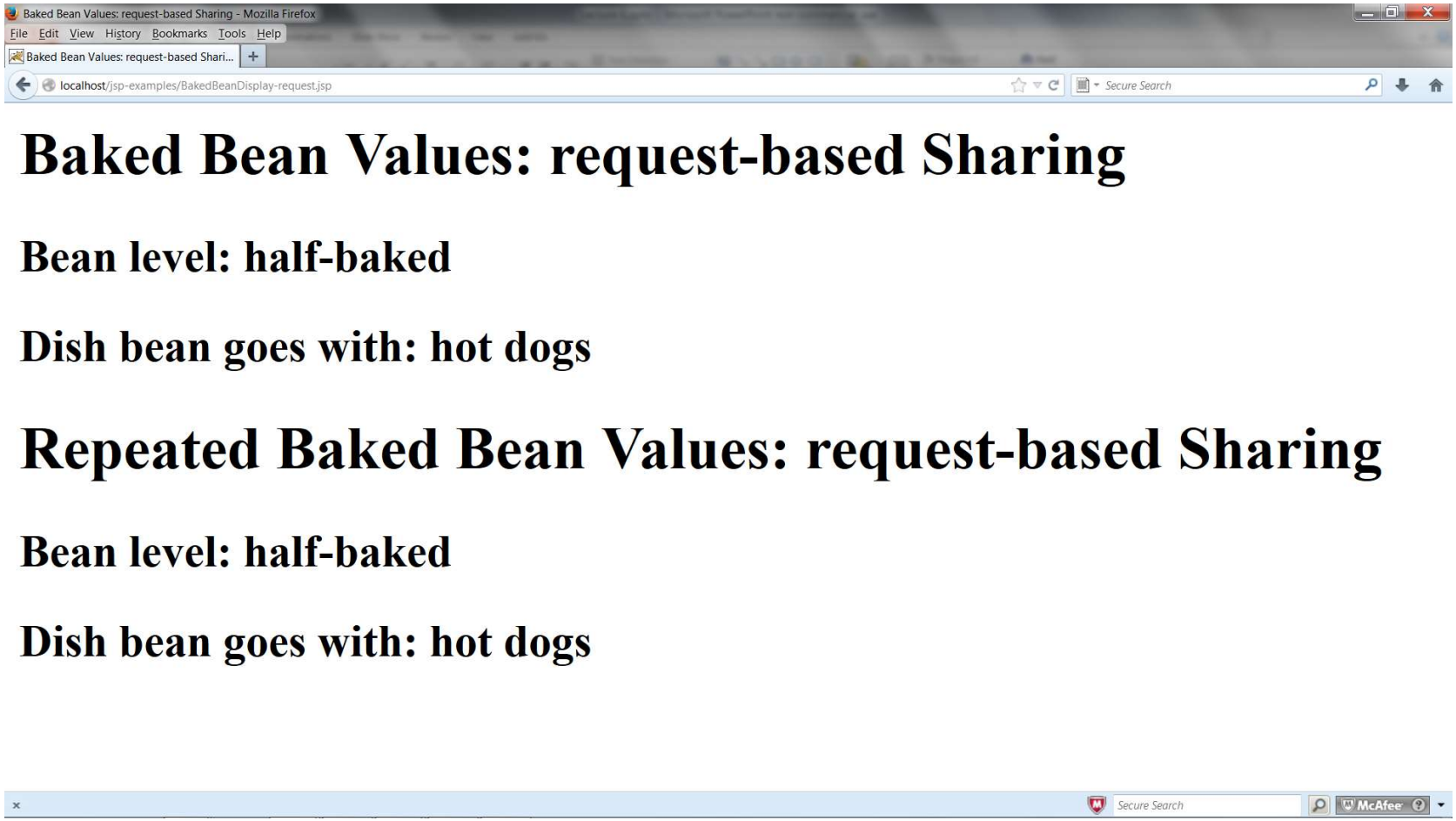
Using Request-Based Sharing

- Initial request to BakedBeanDisplay-request.jsp—BakedBean properties persist to included pages. http://localhost/jsp-examples/BakedBeanDisplay-request.jsp?goesWith=Sashimi



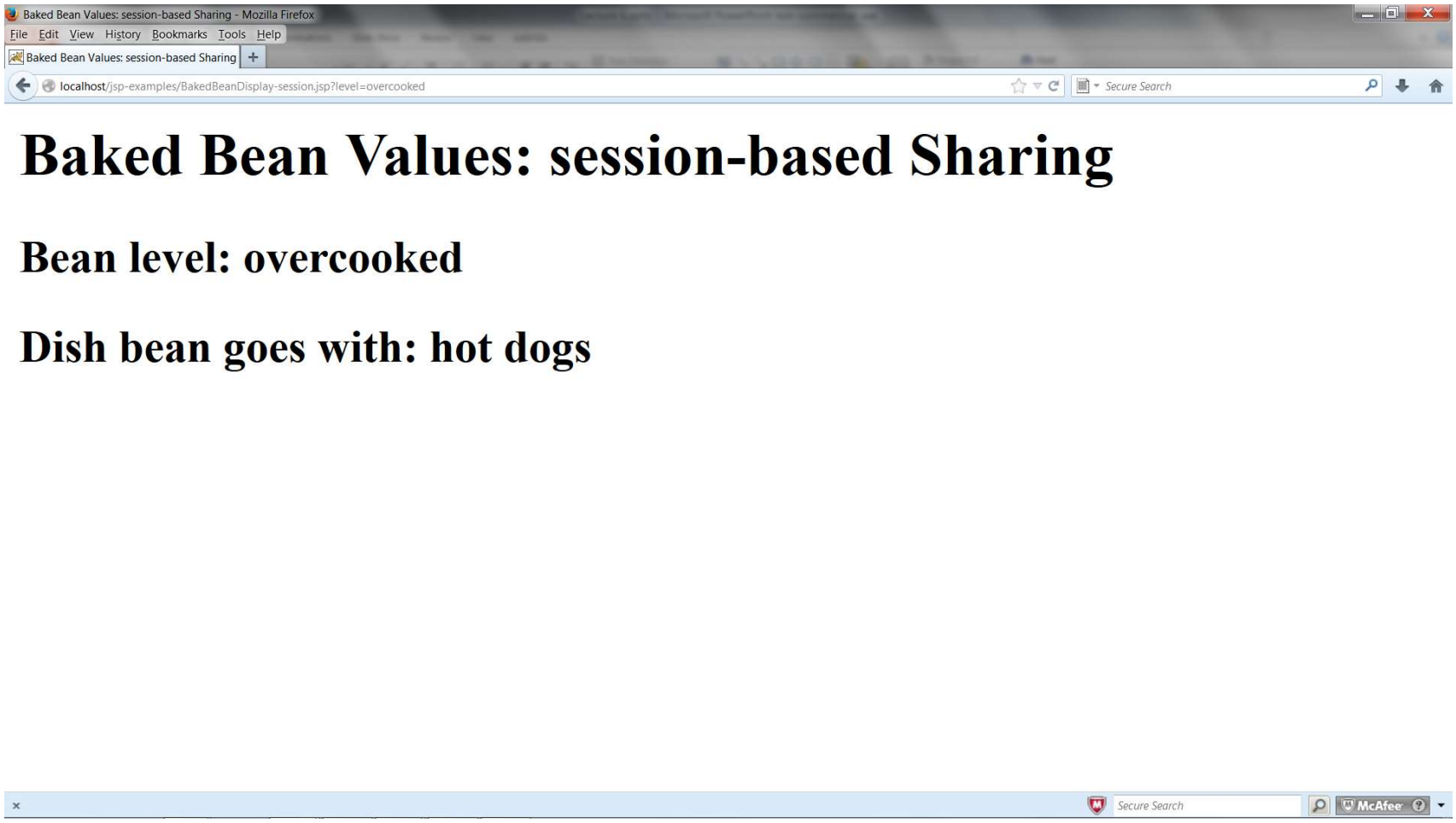
Using Request-Based Sharing

- Subsequent request to BakedBeanDisplay-request.jsp—BakedBean properties do not persist between requests



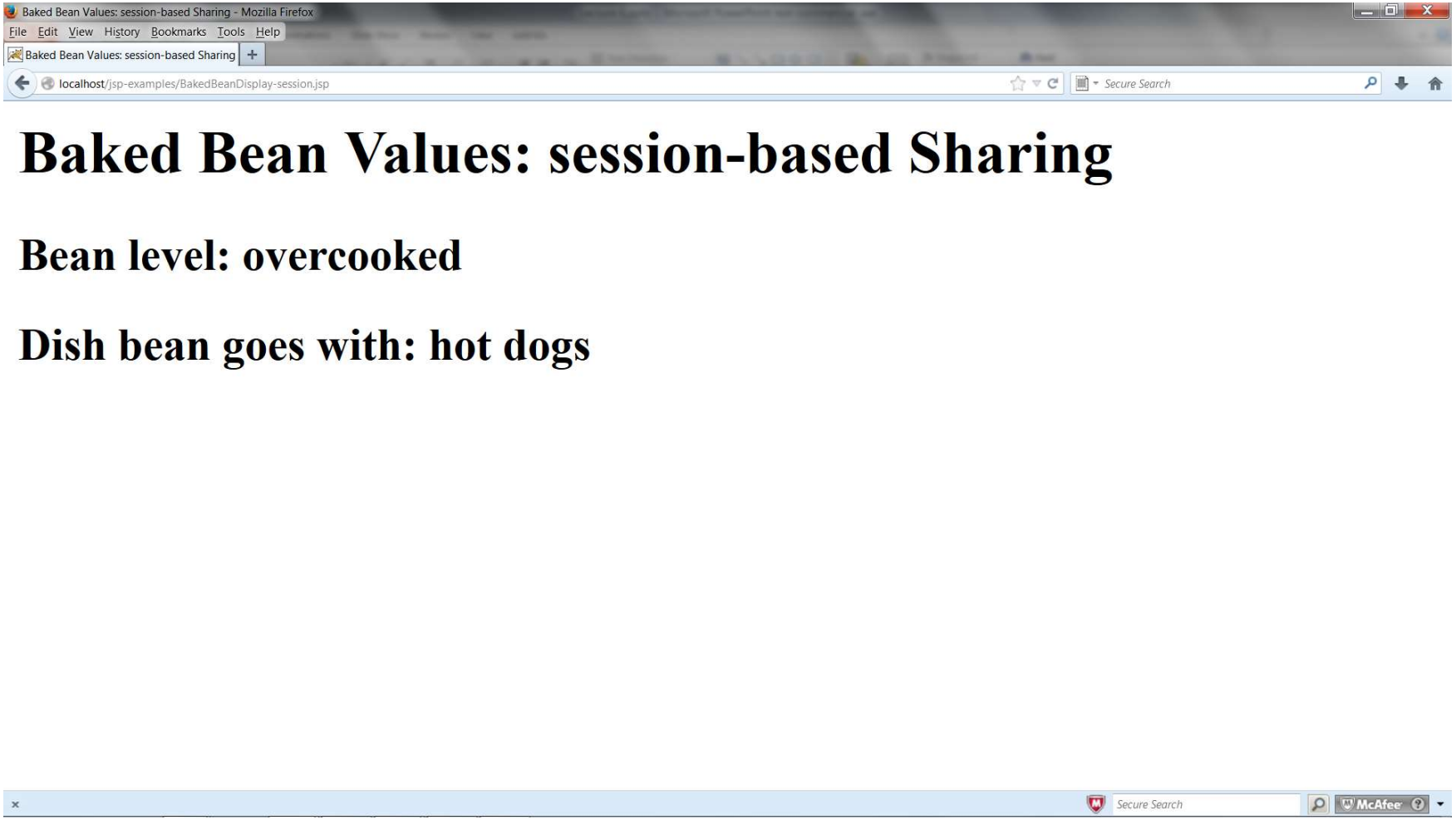
Using Session-Based Sharing

- Initial request to
- http://localhost/jsp-examples/BakedBeanDisplay-session.jsp?level=overcooked.



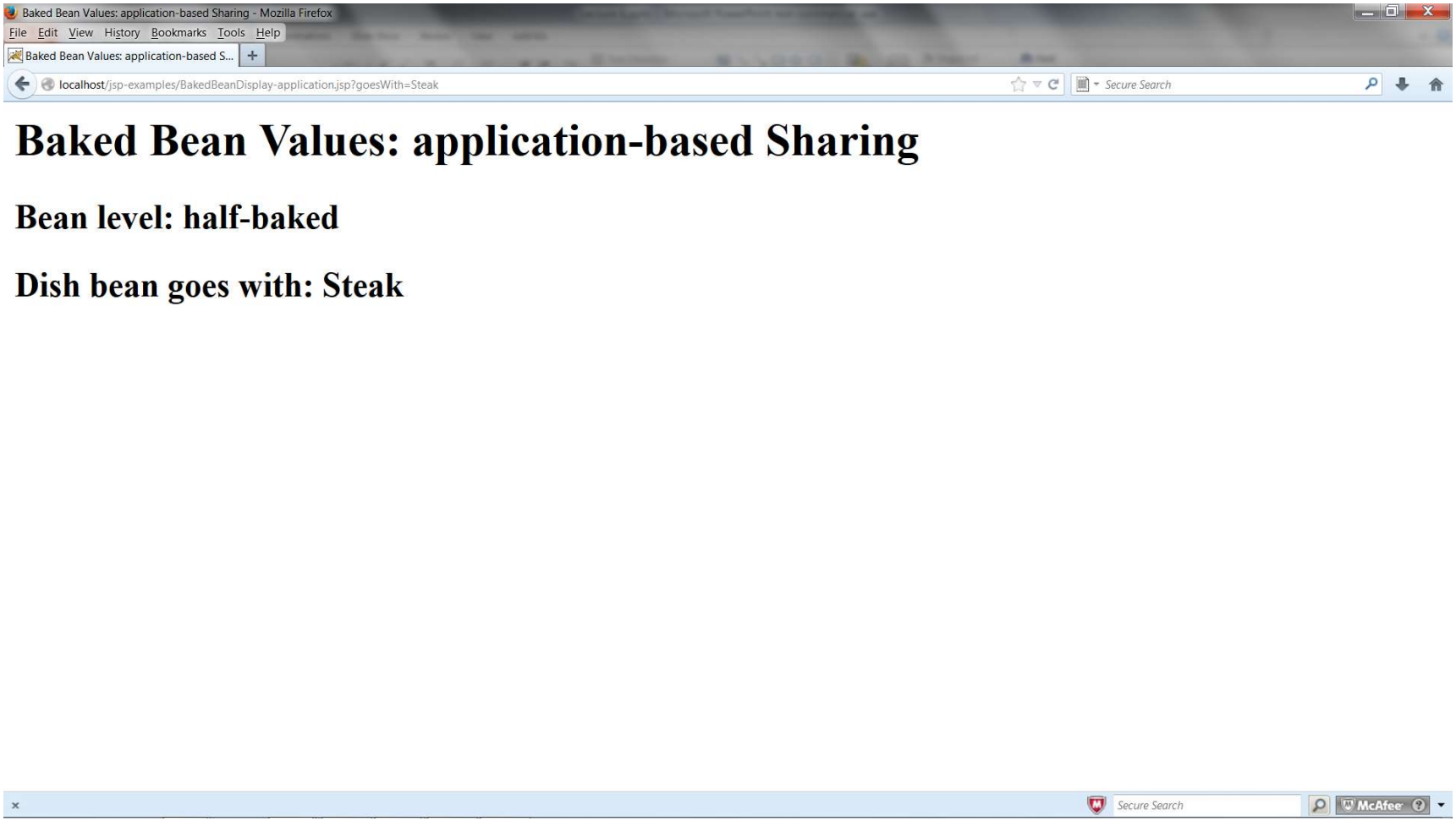
Using Session-Based Sharing

- Subsequent request to BakedBeanDisplay-session.jsp—BakedBean properties persist between requests if the request is from the same client in the same session



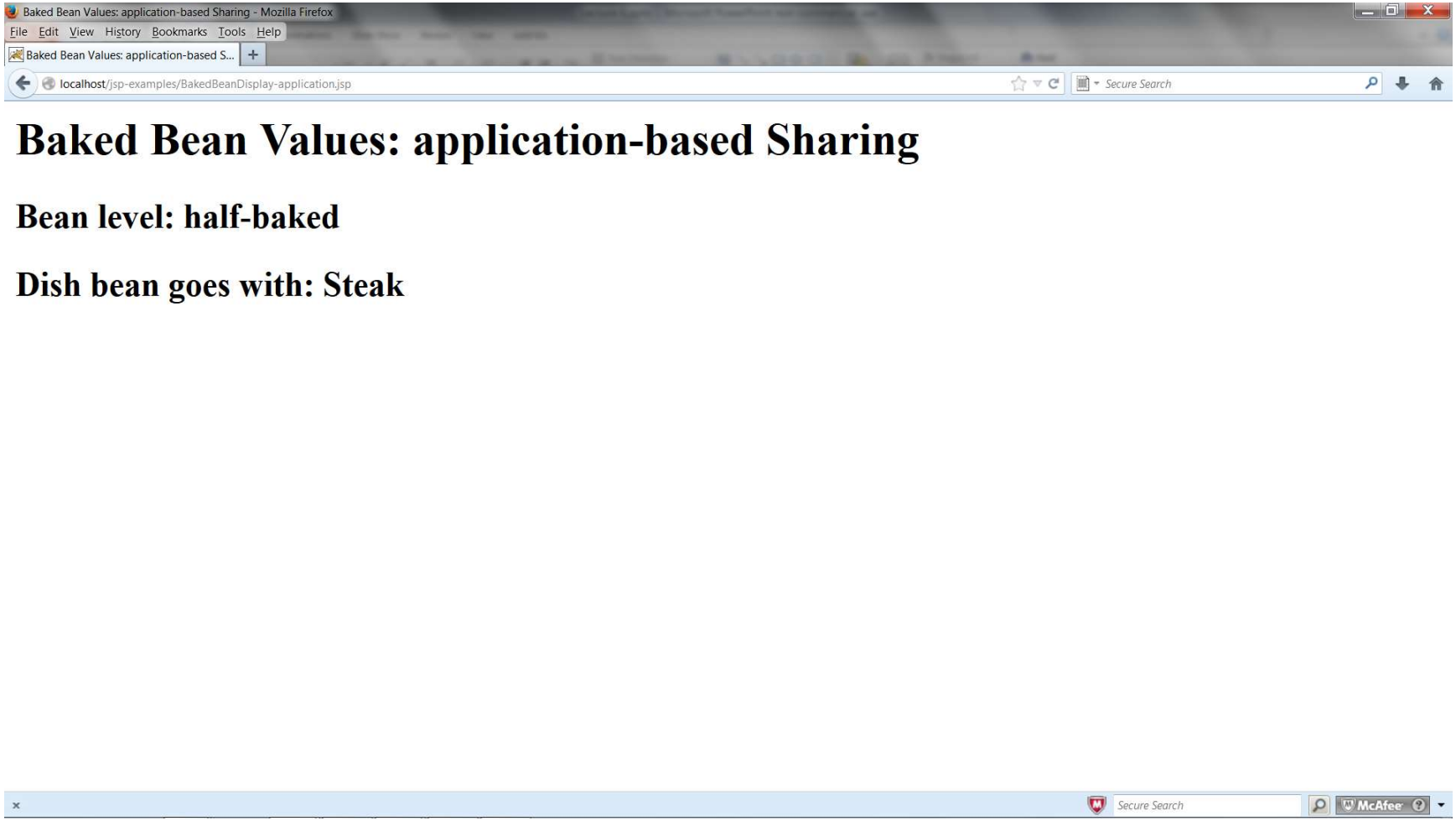
Using Application-Based Sharing

- Initial request to
- http://localhost/jsp-examples/BakedBeanDisplay-application.jsp?goesWith=Steak



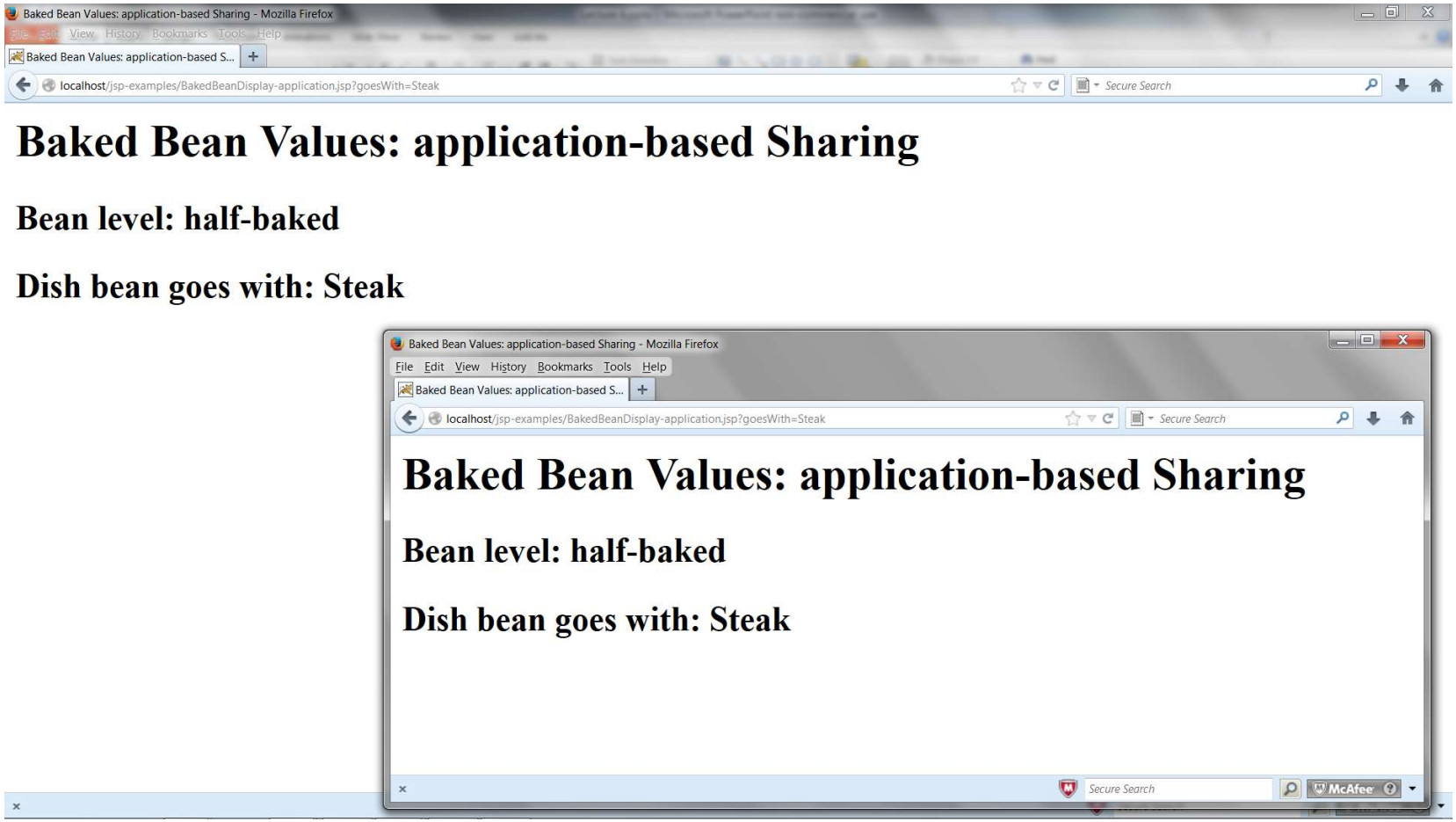
Using Application-Based Sharing

- Subsequent request to BakedBeanDisplay-application.jsp—BakedBean properties persist between requests.



Using Application-Based Sharing

- Subsequent request to BakedBeanDisplay-application—BakedBean properties persist between requests even if the request is from a different client (as here) or is in a different session



Creating Beans Conditionally

- The jsp:useBean element results in a new bean being instantiated only if no bean with the same id and scope can be found. If a bean with the same id and scope *is* found, the preexisting bean is simply bound to the variable referenced by id.
- Second, instead of
  - <jsp:useBean ... />
- you can use
  - <jsp:useBean ...>*statements*</jsp:useBean>

Creating Beans Conditionally

- The statements between the jsp:useBean start and end tags are executed *only* if a new bean is created, *not* if an existing bean is used.
- Because jsp:useBean invokes the default (zero-argument) constructor, you frequently need to modify the properties after the bean is created.
- To mimic a constructor, you should make these modifications only when the bean is first created, not when an existing bean is accessed.
- Multiple pages can contain jsp:setProperty statements between the start and end tags of jsp:useBean; only the page first accessed executes the statements.

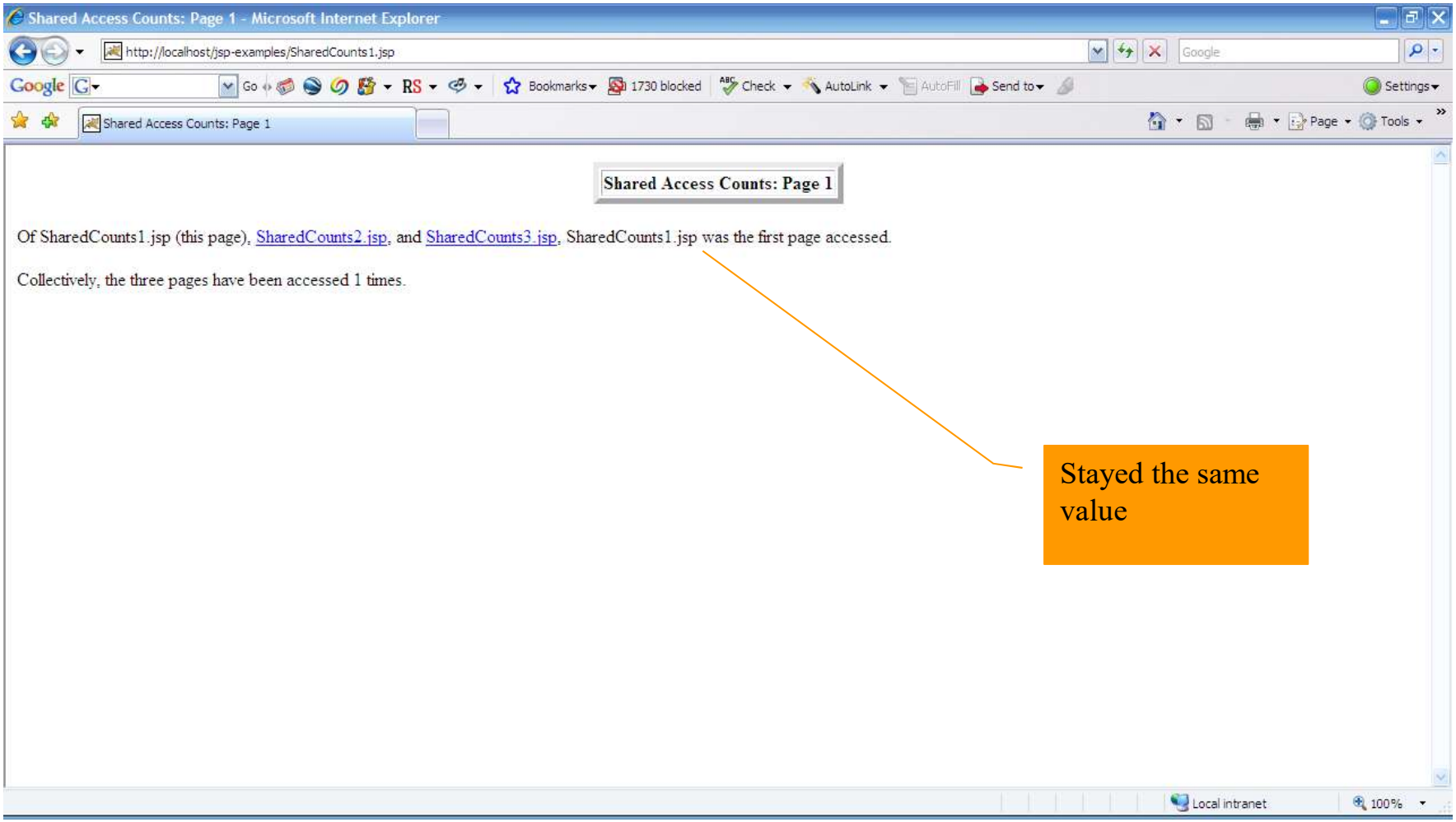
Creating Beans Conditionally

- The following example shows a simple bean that defines two properties: accessCount and firstPage.
- The accessCount property records cumulative access counts to any of a set of related pages and thus should be executed for all requests.
- The firstPage property stores the name of the first page that was accessed and thus should be executed only by the page that is first accessed.
- To enforce the distinction, we place the jsp:setProperty statement that updates the accessCount property in **unconditional code** and place the jsp:setProperty statement for firstPage between the start and end tags of jsp:useBean

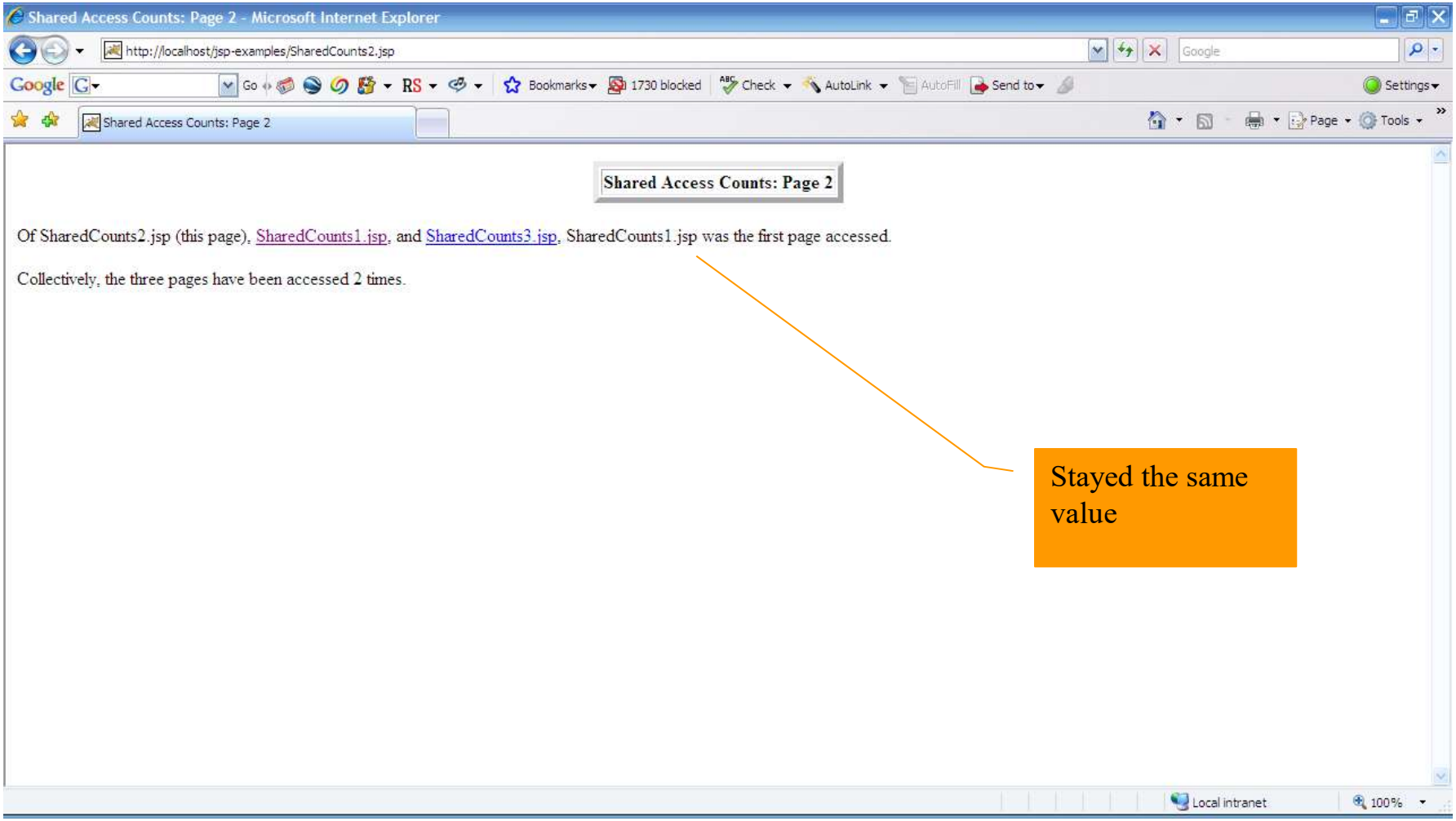
Creating Beans Conditionally

- AccessCountBean.java. Bean to illustrate sharing beans through use of the scope attribute of jsp:useBean.
- SharedCounts1.jsp. The first of three pages that uses the AccessCountBean.
- SharedCounts2.jsp. The second of three pages that uses the AccessCountBean.
- SharedCounts3.jsp. The third of three pages that uses the AccessCountBean.

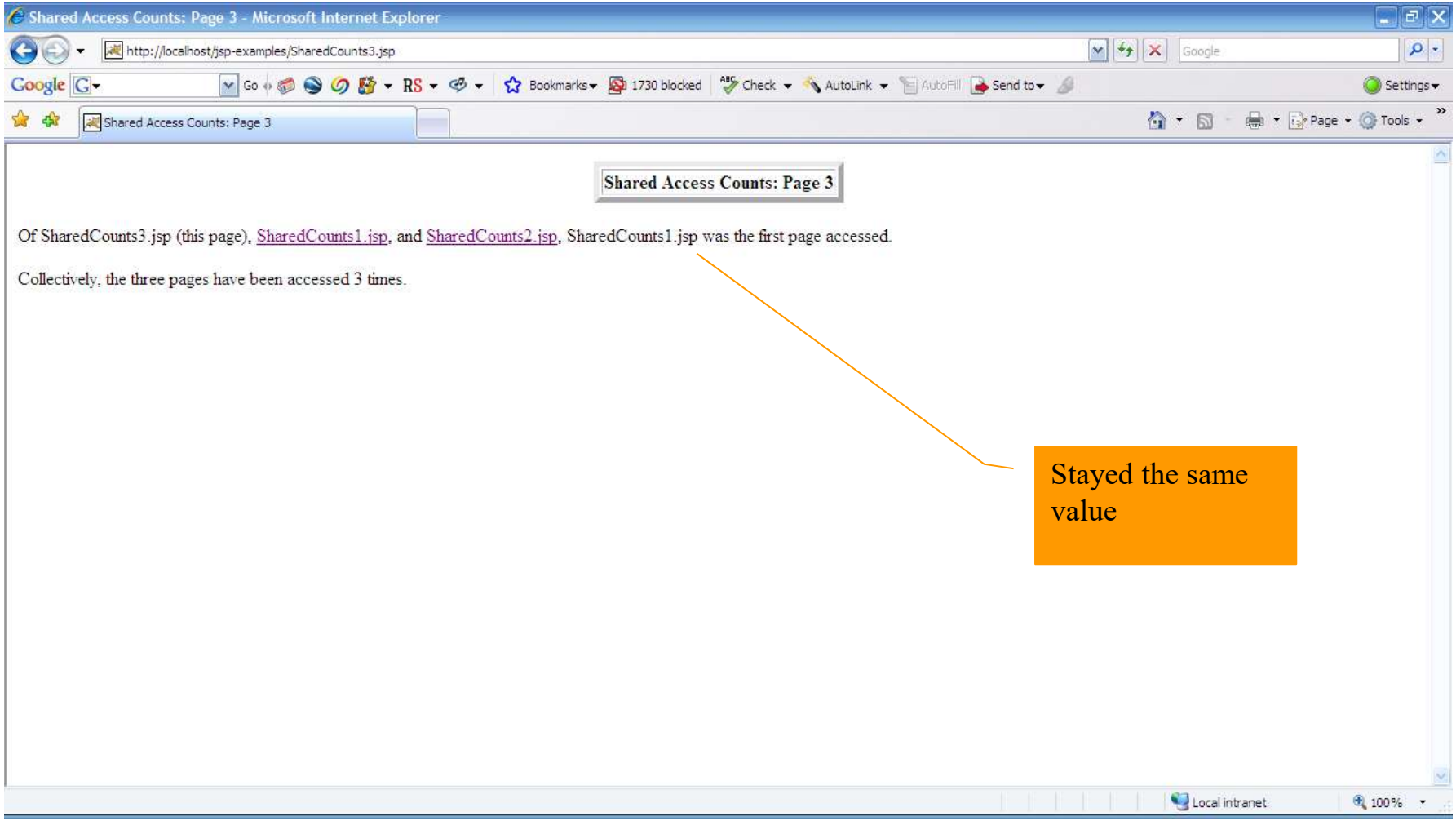
Creating Beans Conditionally



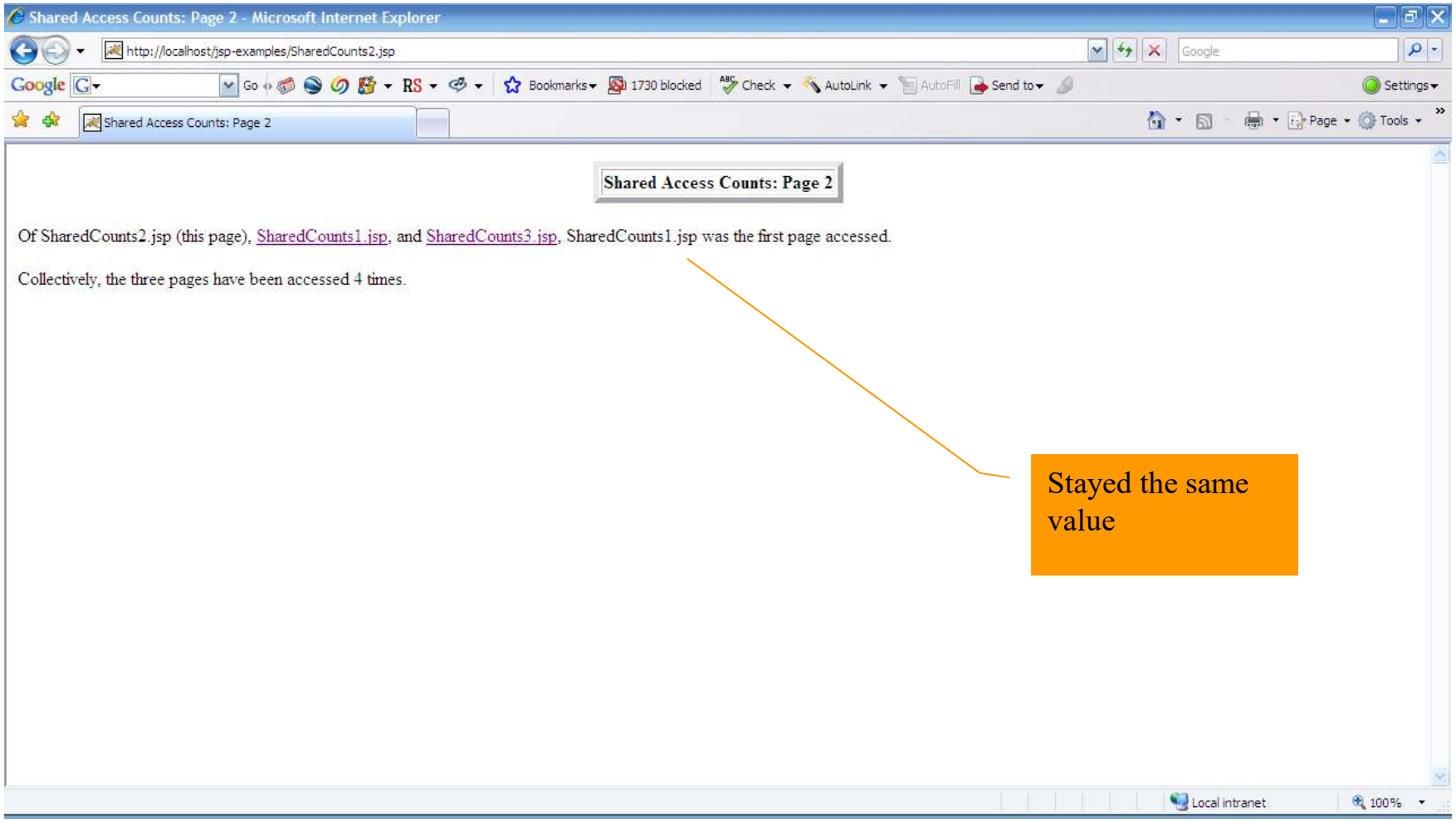
Creating Beans Conditionally



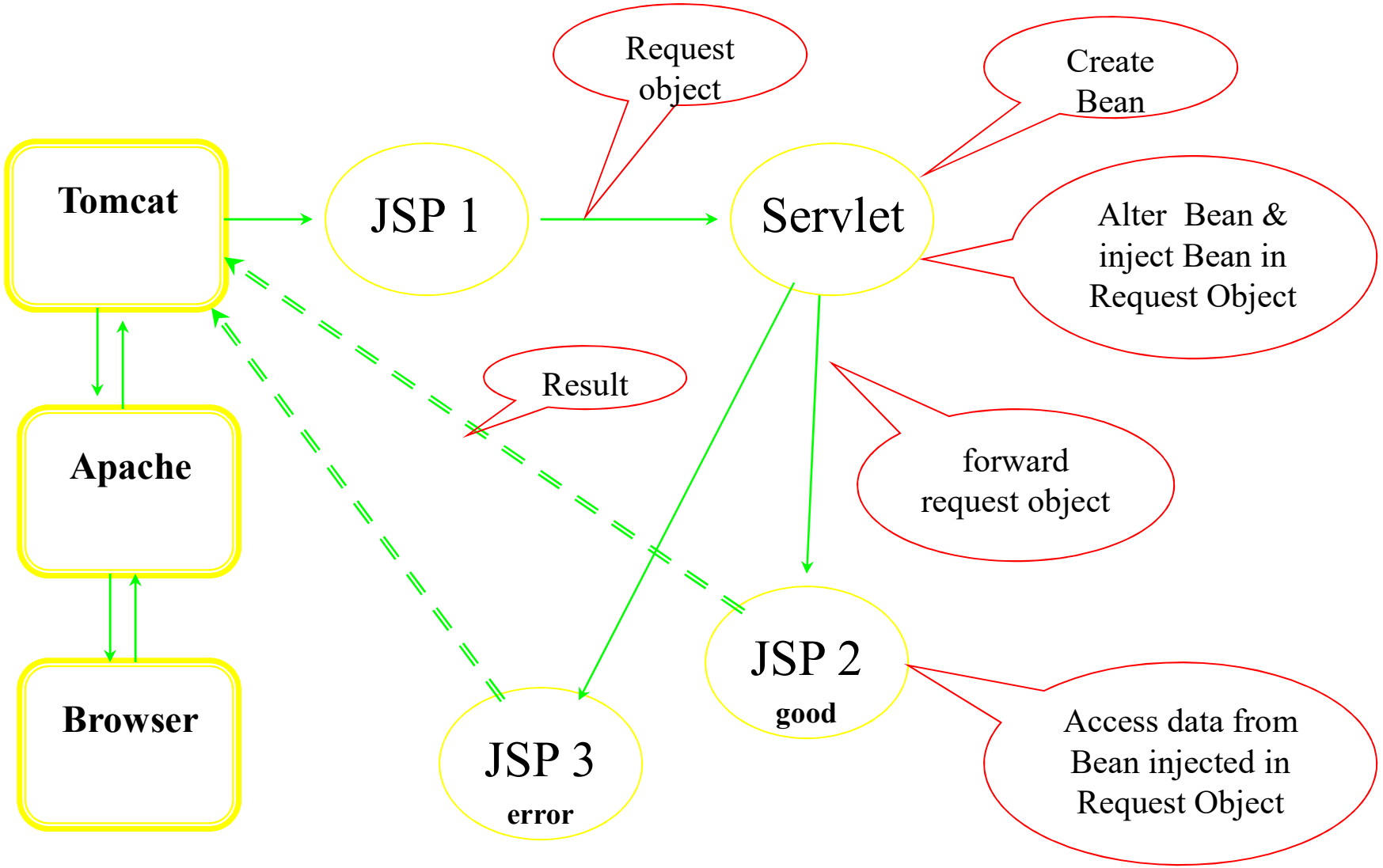
Creating Beans Conditionally



Creating Beans Conditionally



Servlets, and JSP: redirect by Beans



The Counter Example

- **In the following example the front page will be a simple JSP page that will collect the user input: numbers separated by comma: 1,5**
- **The JSP page will hand over data in the request object to Servlet;**
- **The servlet will calculate the sum and avg, create a bean, and then update the bean, and inject the bean in the request object.**
- **The servlet will do the computations and defensive checking, and then pass the result to one of 2 JSP pages: one for good input, and the other if the user didn't enter good data**

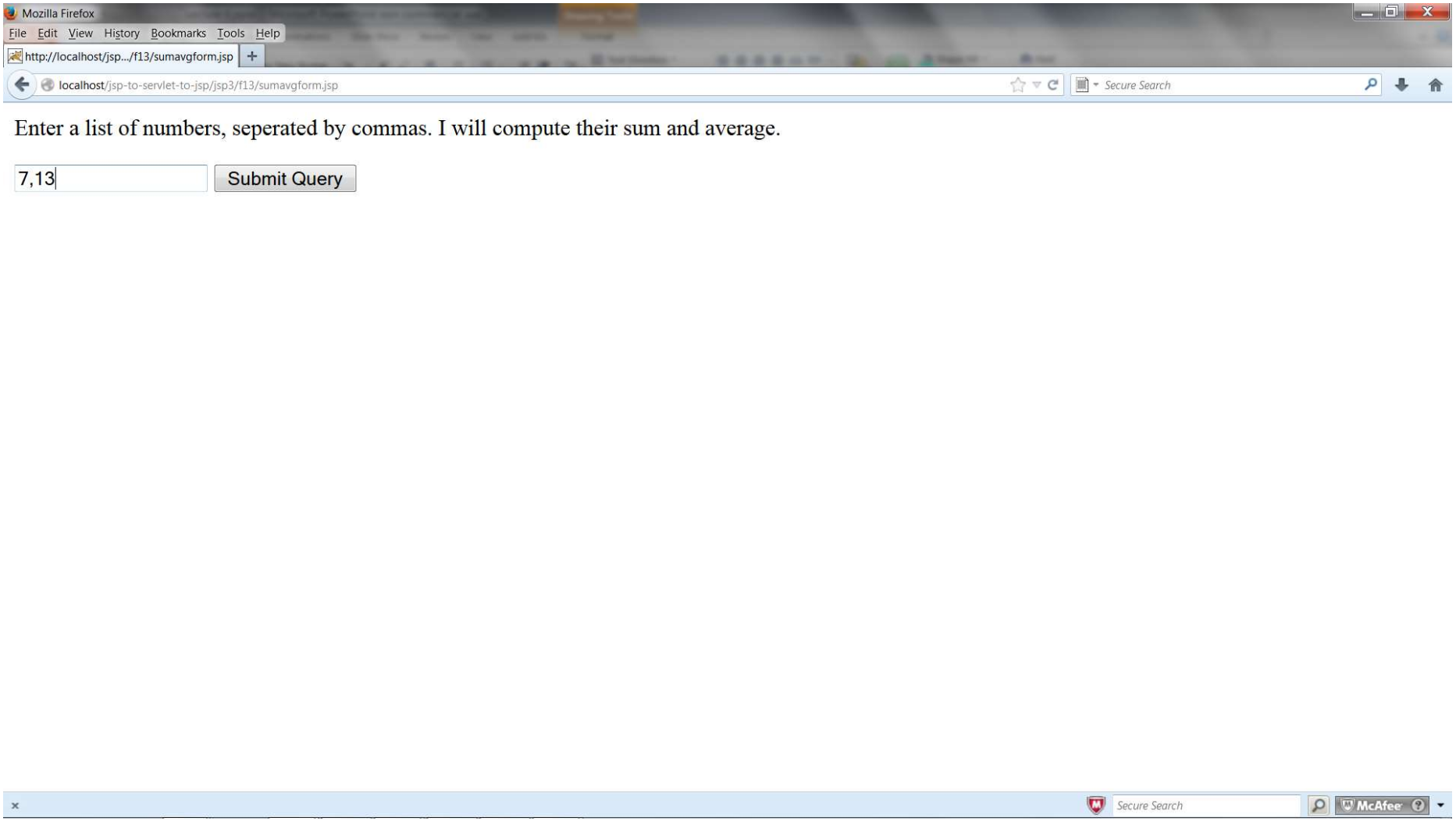
http://localhost/jsp-to-servlet-to-jsp/jsp3/f13/sumavgform.jsp

The following files are used in this example:

- sumavgform.jsp
- sumavgresult.jsp
- sumavgrerror.jsp
- SumAvgServlet.java
- SumAvgBean.java

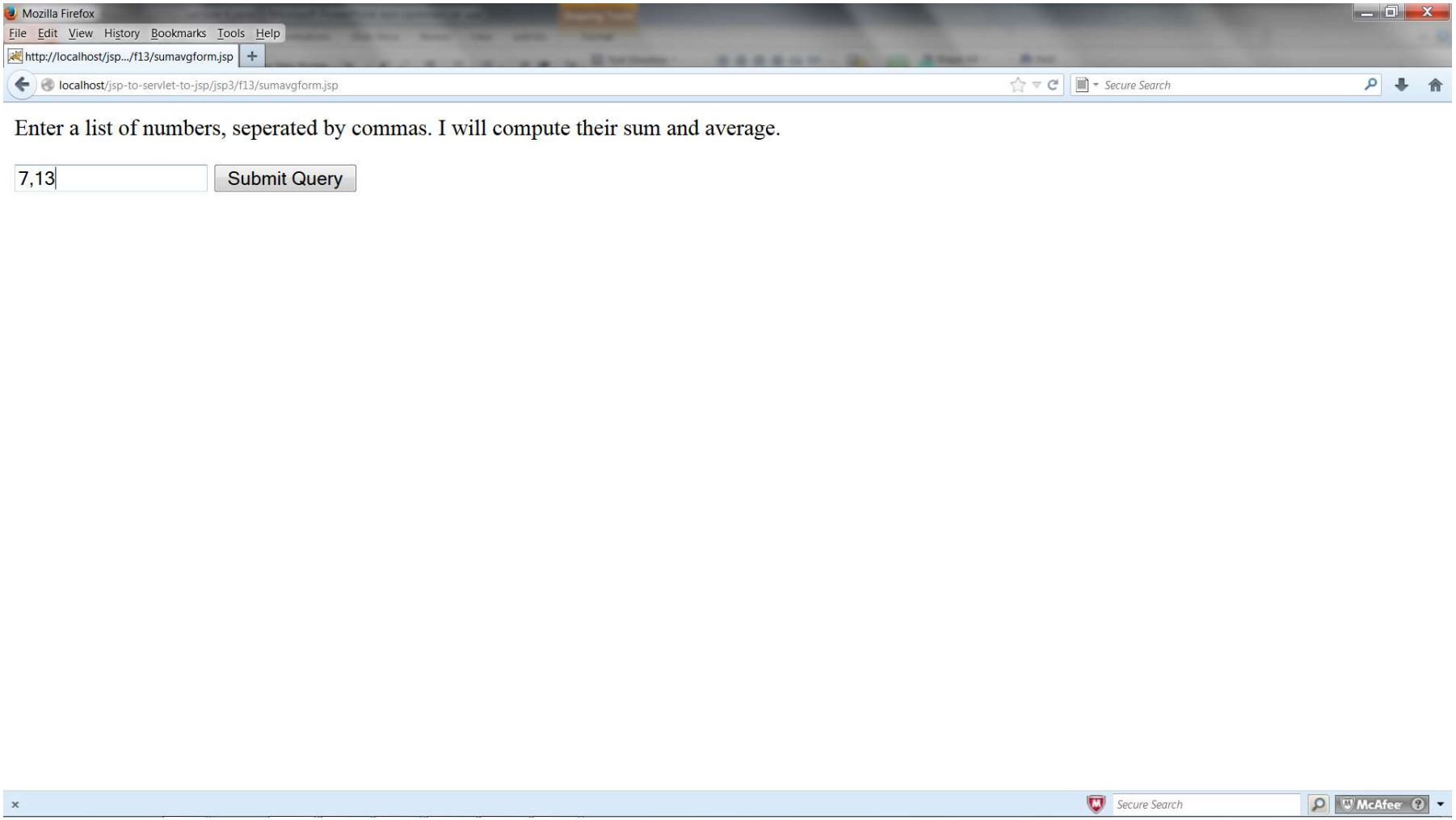
http://localhost/jsp-to-servlet-to-jsp/jsp3/f13/sumavgform.jsp

Here is the screen-shot



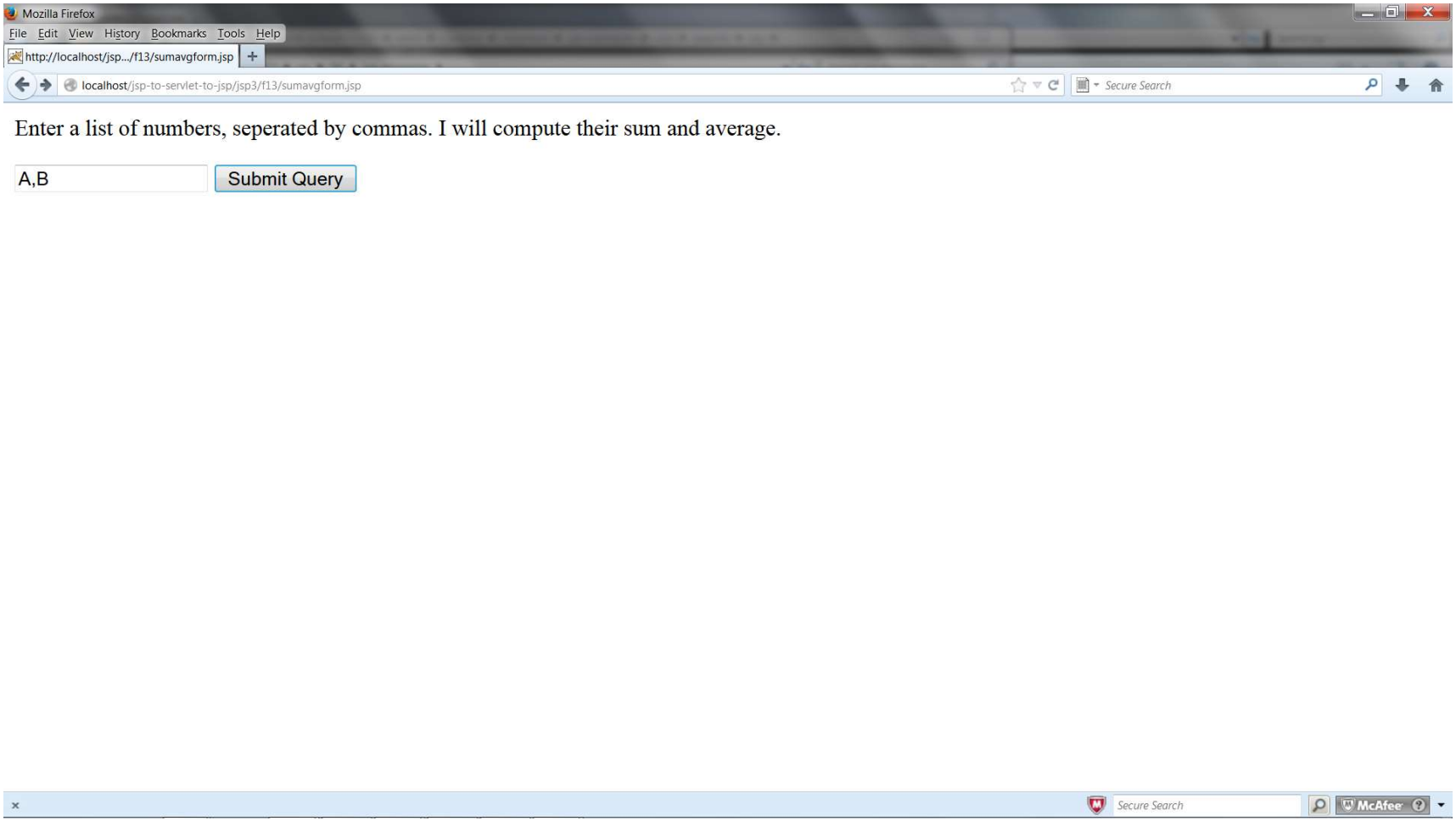
http://localhost/jsp-to-servlet-to-jsp/jsp3/f13/sumavgform.jsp

Here is the screen-shot after submit



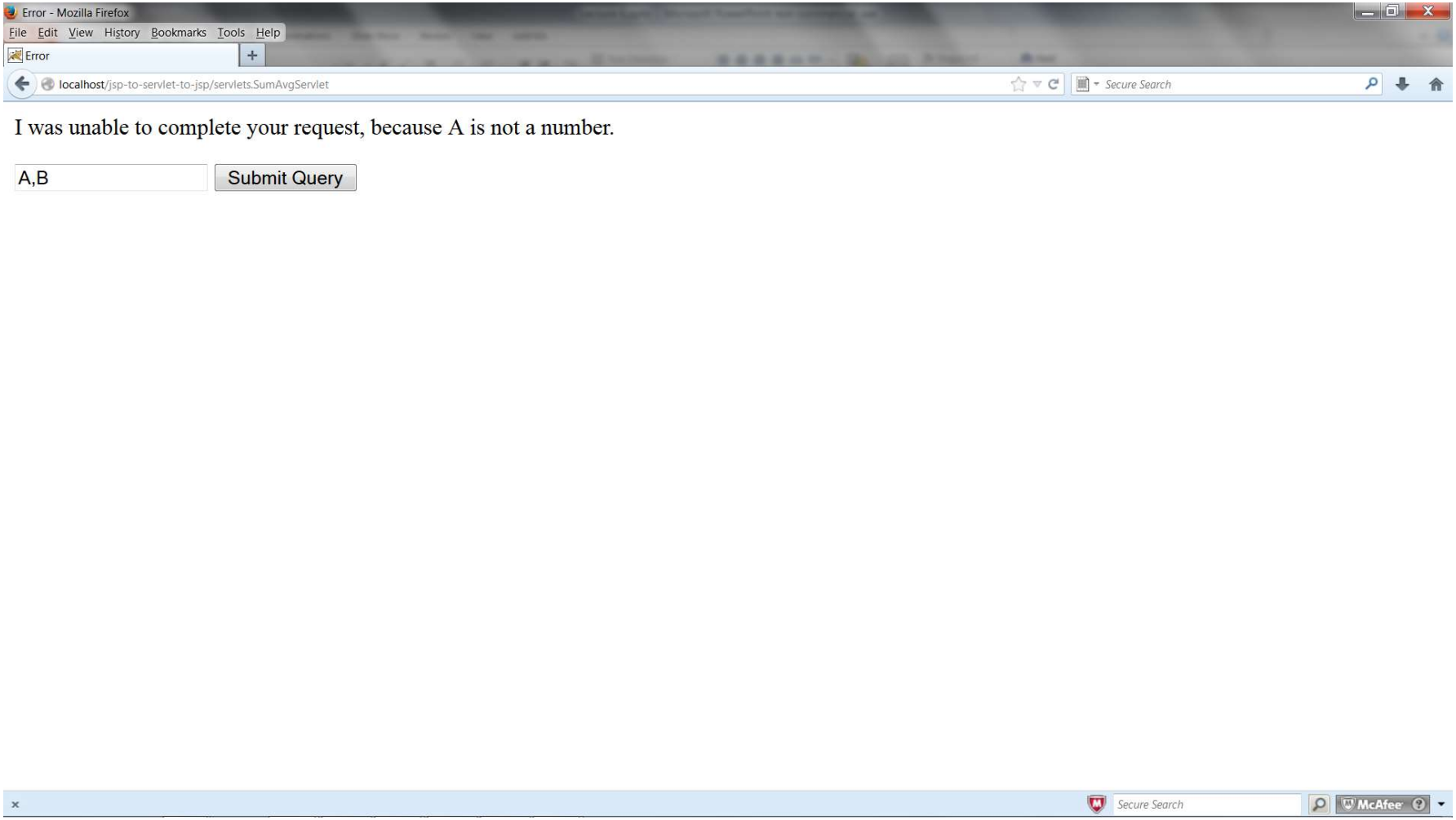
http://localhost/jsp-to-servlet-to-jsp/jsp3/f13/sumavgform.jsp

Here is the screen-shot for an attempt to add A and B



http://localhost/jsp-to-servlet-to-jsp/jsp3/f13/sumavgform.jsp

Here is the screen-shot for submit



How to structure a web application with the MVC pattern

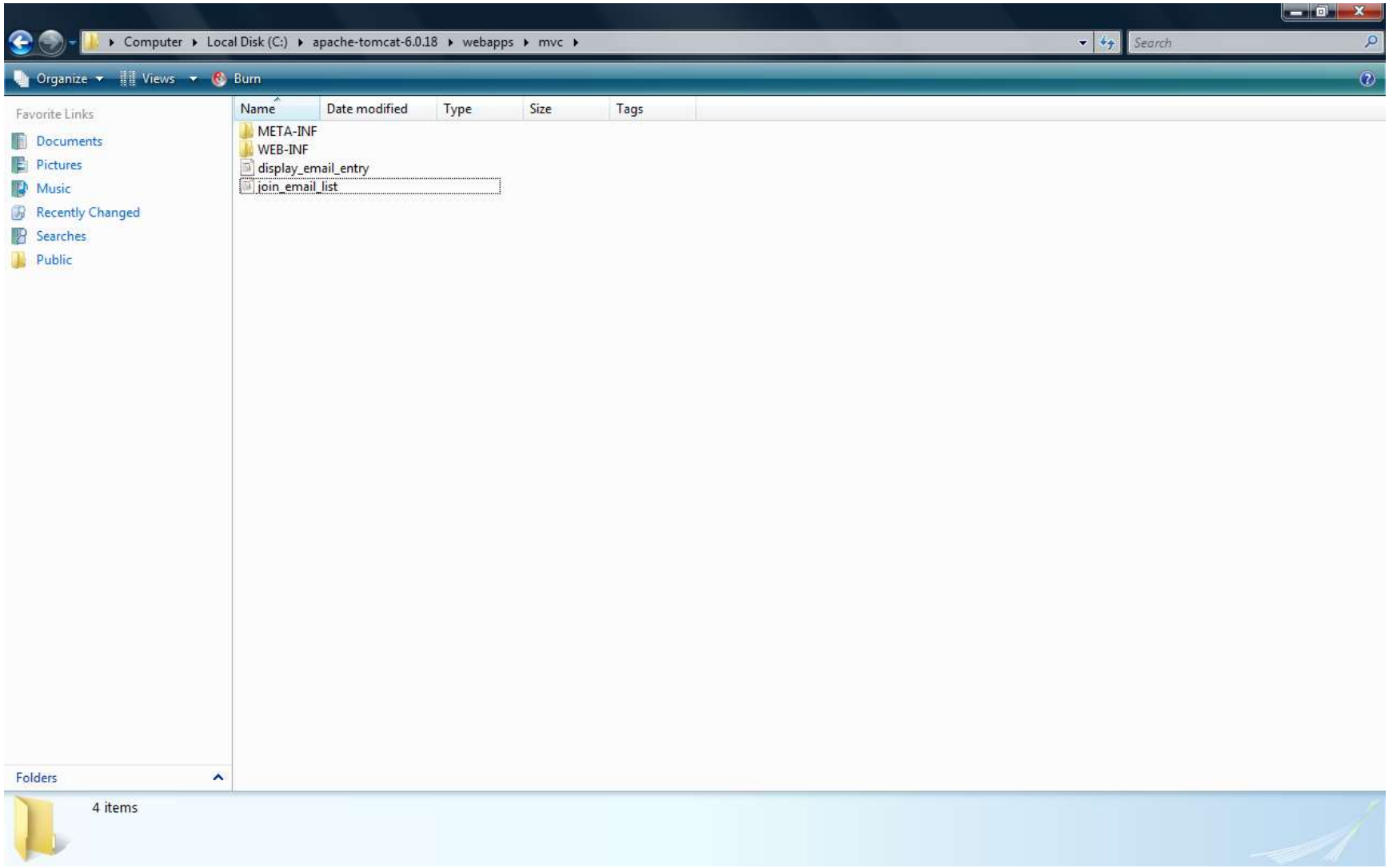
- Use the MVC pattern to develop your web applications so servlets control the processing and JSPs do the presentation.

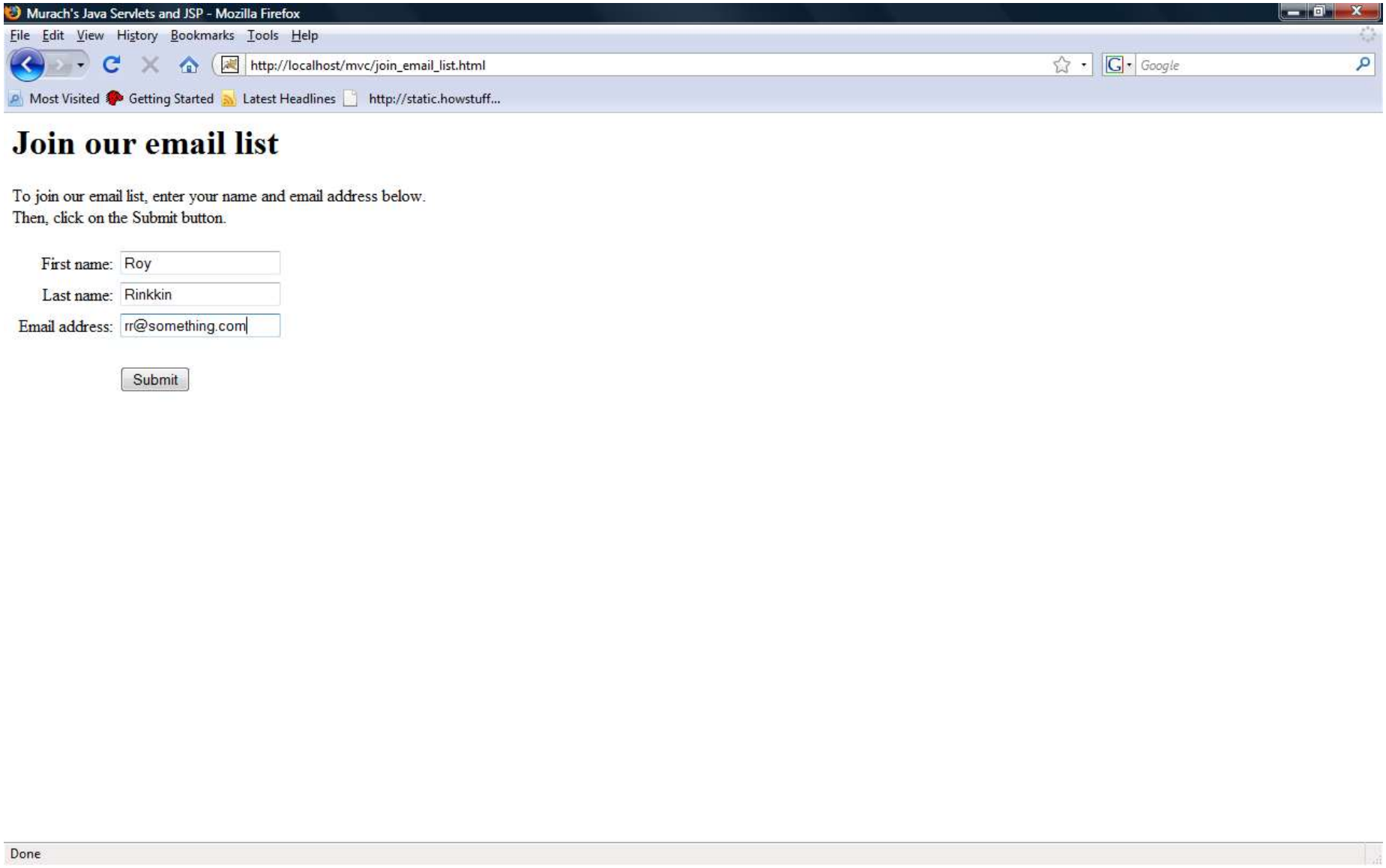
- Provide for server-side data validation in your applications.

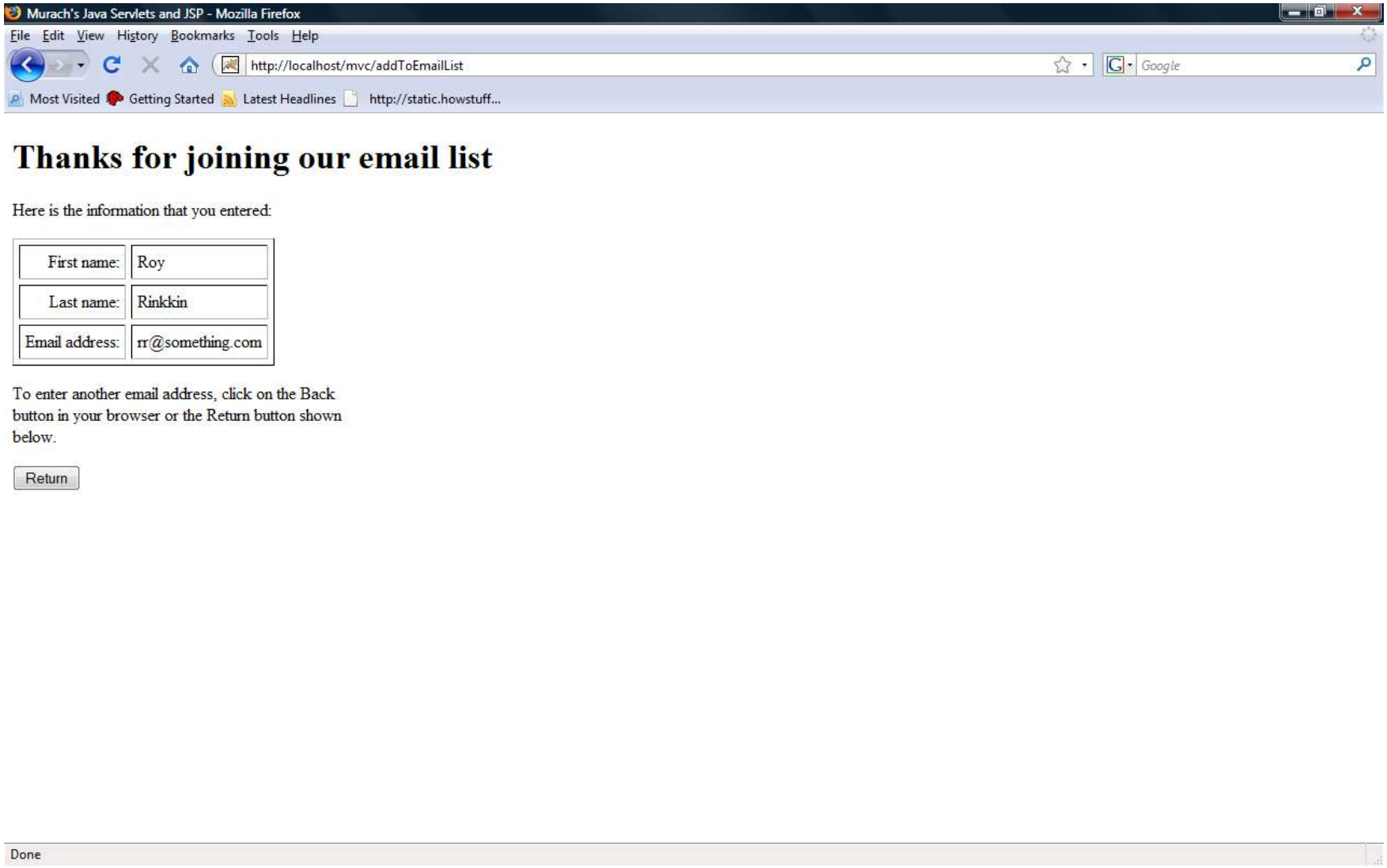
- Use include files in your JSPs at compile-time or runtime.

- Use the web.xml file to set initialization parameters and use your servlets to get the parameters.

➢Lets look at the following example:




Murach's Java Servlets and JSP - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/mvc/join_email_list.html
Google
Most Visited
Getting Started
Latest Headlines
http://static.howstuff...
Join our email list
To join our email list, enter your name and email address below.
Then, click on the Submit button.
First name: Roy
Last name: Rinkkin
Email address: rr@something.com
Submit
Done


Murach's Java Servlets and JSP - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/mvc/addToEmailList
Google
Most Visited
Getting Started
Latest Headlines
http://static.howstuff...
Thanks for joining our email list
Here is the information that you entered:
First name: Roy
Last name: Rinkkin
Email address: rr@something.com
To enter another email address, click on the Back button in your browser or the Return button shown below.
Return
Done

The code for the servlet

```
package email;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import business.User;
import data.UserIO;

public class AddToEmailListServlet extends HttpServlet
{
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
                          throws ServletException,
                                 IOException
    {
        // get parameters from the request
        String firstName =
            request.getParameter("firstName");
        String lastName = request.getParameter("lastName");
        String emailAddress =
            request.getParameter("emailAddress");
```

The code for the servlet (cont.)

```
        // get a relative file name
        ServletContext context = getServletContext();
        String path =
            context.getRealPath("/WEB-INF/EmailList.txt");

        // use regular Java classes
        User user = new User(firstName, lastName,
            emailAddress);
        UserIO.addRecord(user, path);

        // store the User object in the request object
        request.setAttribute("user", user);

        // forward request and response objects to JSP page
        String url = "/display_email_entry.jsp";
        RequestDispatcher dispatcher =
             getServletContext().getRequestDispatcher(url);
        dispatcher.forward(request, response);
    }
}
```

The code for the JSP

```
<!DOCTYPE HTML PUBLIC
    "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
    <title>Murach's Java Servlets and JSP</title>
</head>

<body>
    <h1>Thanks for joining our email list</h1>

    <p>Here is the information that you entered:</p>

    <%@ page import="business.User" %>
    <% User user = (User) request.getAttribute("user"); %>
    <table cellspacing="5" cellpadding="5" border="1">
        <tr>
            <td align="right">First name:</td>
            <td><%= user.getFirstName() %></td>
        </tr>
```

The code for the JSP (cont.)

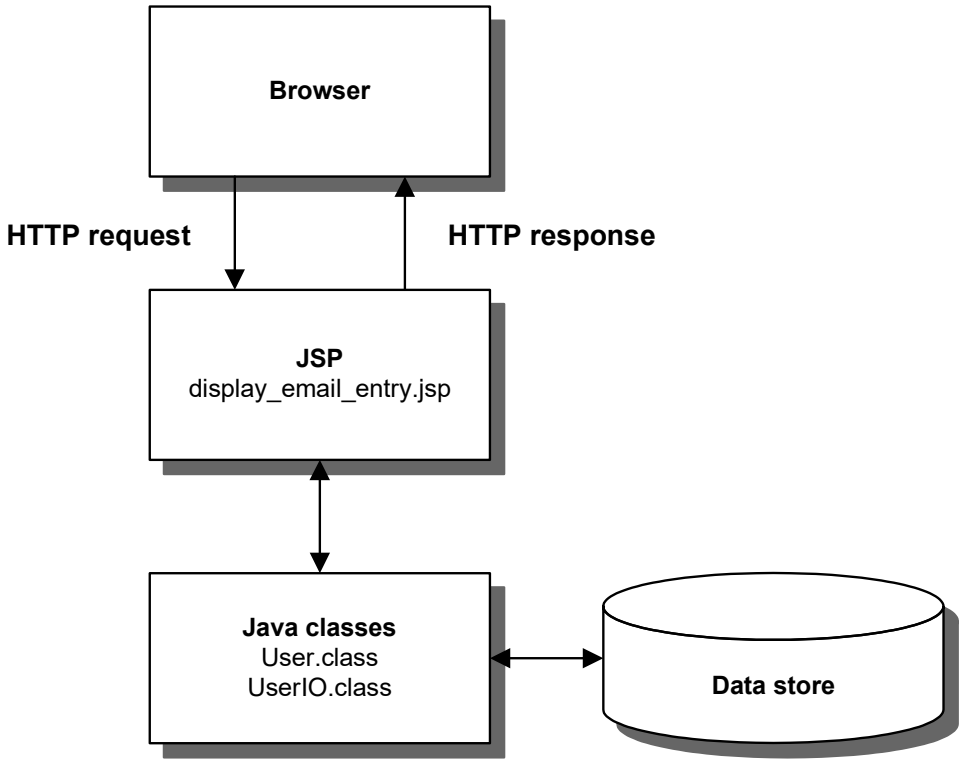
```
        <tr>
            <td align="right">Last name:</td>
            <td><%= user.getLastName() %></td>
        </tr>
        <tr>
            <td align="right">Email address:</td>
            <td><%= user.getEmailAddress() %></td>
        </tr>
    </table>

    <p>To enter another email address, click on the Back <br
    button in your browser or the Return button shown <br>
    below.</p>

    <form action="join_email_list.html" method="post">
        <input type="submit" value="Return">
    </form>

</body>
</html>
```

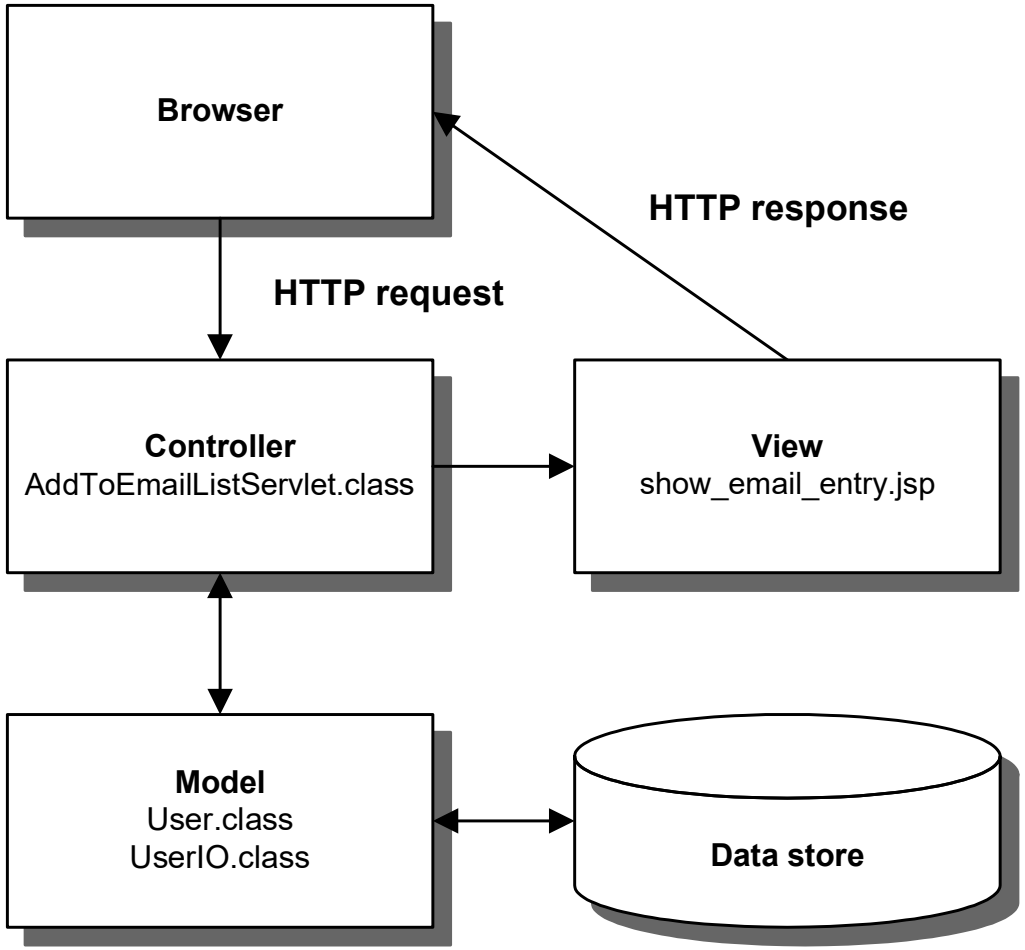
The Model 1 architecture



The Model 1 architecture

- The *Model 1 architecture* is sometimes adequate for web applications with limited processing requirements. With this architecture, JSPs handle all of the processing and presentation for the application.
- In the Model 1 architecture, the JSPs can use regular Java classes to store the data of the application and to do the business processing of the application.
- The *data store* can be a database or one or more disk files. This is often referred to as *persistent data storage* because it exists after the application ends.

The Model-View-Controller pattern



The Model-View-Controller (MVC) pattern

- The *Model-View-Controller (MVC) pattern* is commonly used to structure web applications that have significant processing requirements. That makes them easier to code and maintain. This pattern is also known as the *Model 2 architecture*.
- In the MVC pattern, the *model* consists of business objects like the User object, the *view* consists of HTML pages and JSPs, and the *controller* consists of servlets.
- Usually, the methods of data classes like the UserIO class are used to read and write business objects like the User object to and from the data store.
- When you use the MVC pattern, you try to construct each layer so it's as independent as possible. Then, if you need to make changes to one layer, any changes to the other layers are minimized.

Two methods available from the request object

Method	Description
setAttribute(String name, Object o**)**	Stores any object in the request as an attribute and specifies a name for the attribute. Attributes are reset between requests.
getAttribute(String name**)**	Returns the value of the specified attribute as an Object type. If no attribute exists for the specified name, this method returns a null value.

How to set a request attribute for User-Defined data type

```
User user = new User(firstName, lastName, emailAddress);
request.setAttribute("user", user);
```

How to get a request attribute for User-Defined data type

```
User user = (User) request.getAttribute("user");
```

How to set a request attribute for a primitive type

```
int id = 1;
request.setAttribute("id", id);
```

How to get a request attribute for a primitive type

```
int id = (Integer) request.getAttribute("id");
```