Agenda:

- Basic HTML concepts
- HTML Forms and Servlets
- Tomcat Common Bean Utility library

A crash course in HTML

HTML basics

- *HyperText Markup Language* (*HTML*) is used to provide the user interface for web applications.
- To write and edit HTML code and JSPs, you can use a general text editor like NotePad, a text editor that's specifically designed for working with HTML, or an *Integrated Development Environment*, or *IDE*, that's designed for developing web applications.
- An *HTML document* is used to define each *HTML page* that's displayed by a web browser.
- Within an HTML document, *HTML tags* define how the page will look when it is displayed. Each of these HTML tags is coded within a set of brackets (< >).
- HTML tags aren't case sensitive.
- To make your code easier to read, you can use spaces, indentation, and blank lines.

Basic HTML tags

Tag	Description
`<!doctype ... >`	Identifies the type of HTML document. This tag is often inserted automatically by the HTML editor.
`<html> </html>`	Marks the beginning and end of an HTML document.
`<head> </head>`	Marks the beginning and end of the Head section of the HTML document.
`<title> </title>`	Marks the title that is displayed in the title bar of the browser.
`<body> </body>`	Marks the beginning and end of the Body section of the HTML document.

Basic HTML tags (cont.)

Tag	Description
`<h1> </h1>`	Tells the browser to use the default format for a heading-1 paragraph.
`<h2> </h2>`	Tells the browser to use the default format for a heading-2 paragraph.
`<p> </p>`	Tells the browser to use the default format for a standard paragraph.
` `	Inserts a line break.
`<b> </b>`	Marks text as bold.
`<i> </i>`	Marks text as italic.
`<u> </u>`	Marks text as underlined.
`<!-- -->`	Defines a comment that is ignored by the browser.

Anchor tags…

With URLs that are relative to the current directory

```
<a href="join.html">The Email List application 1</a><br>
<a href="email/join.html">
   The Email List application 2</a><br>
```

With relative URLs that navigate up the directory structure

```
<a href="../">Go back one directory level</a><br>
<a href="../../">Go back two directory levels</a><br>
```

With URLs that are relative to the webapps directory

```
<a href="/">Go to the default root directory for the web
server</a><br>
<a href="/musicStore">Go to the root directory of the
musicStore app</a>
```

With absolute URLs

```
<a href=
    "http://www.murach.com/email">An Internet address</a>
<a href="http://64.71.179.86/email">An IP address</a>
```

The Anchor tag

Tag	Description
`<a> </a>`	Defines a link to another URL. When the user clicks on the text that's displayed by the tag, the browser requests the page that is identified by the Href attribute of the tag.

One attribute of the Anchor tag

Attribute	Description
`href`	Specifies the URL for the link.

The HTML code for a table

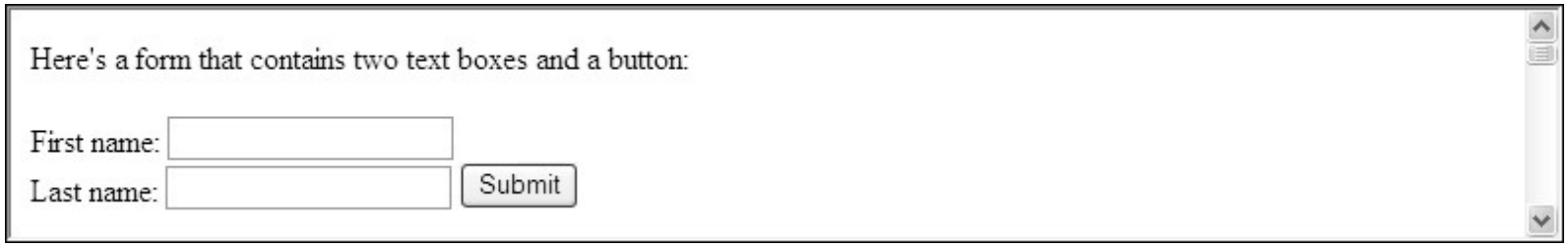
```
<p>Here is the information that you entered:</p>

<table cellspacing="5" cellpadding="5" border="1">
    <tr>
        <td align="right">First name:</td>
        <td>John</td>
    </tr>
    <tr>
        <td align="right">Last name:</td>
        <td>Smith</td>
    </tr>
    <tr>
        <td align="right">Email address:</td>
        <td>jsmith@hotmail.com</td>
    </tr>
</table>
```

The table displayed in a browser

The tags for working with tables

Tag	Description
`<table> </table>`	Marks the start and end of the table.
`<tr> </tr>`	Marks the start and end of each row.
`<td> </td>`	Marks the start and end of each data cell within a row.

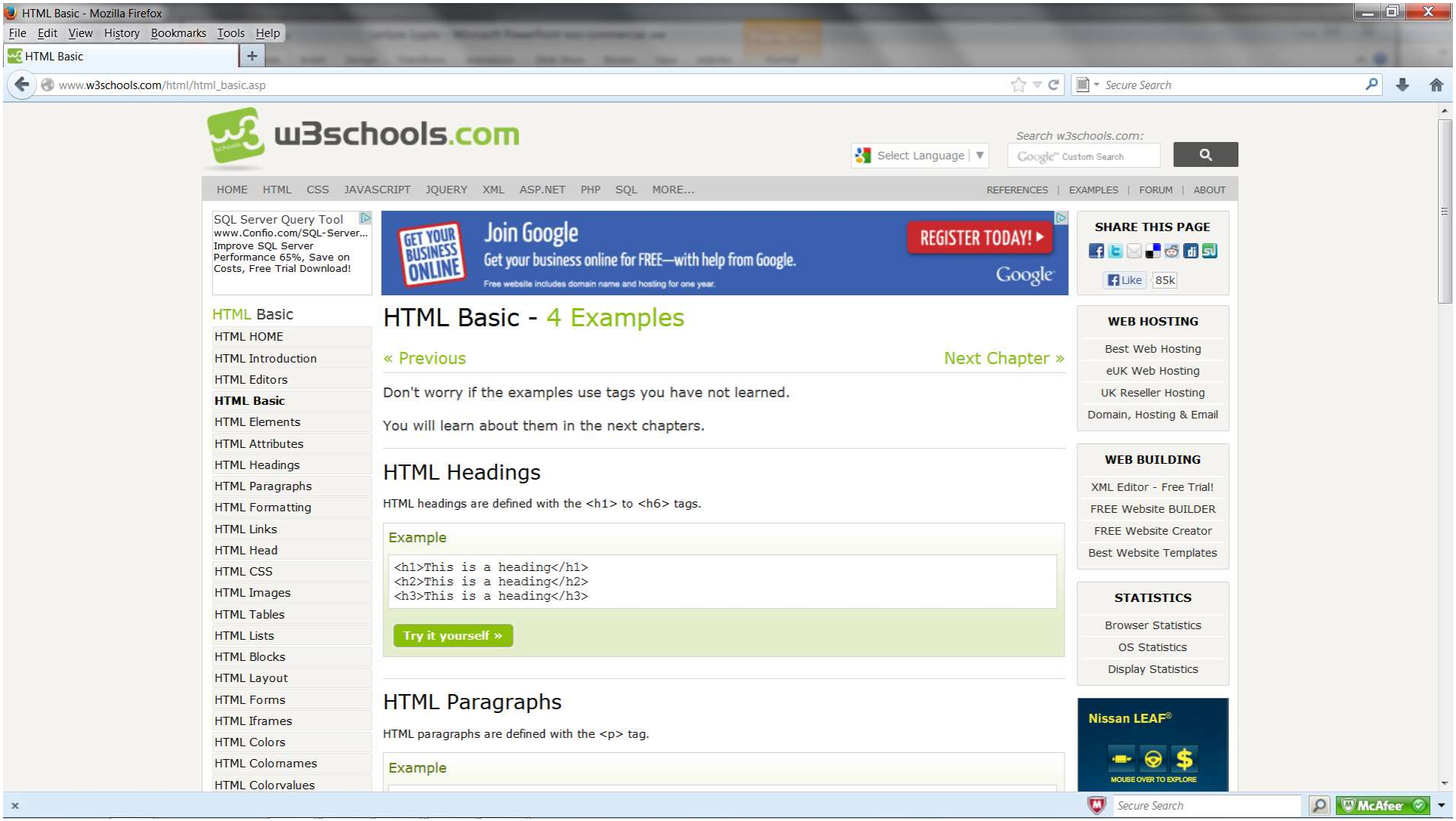
A form displayed in a browser before the user enters data



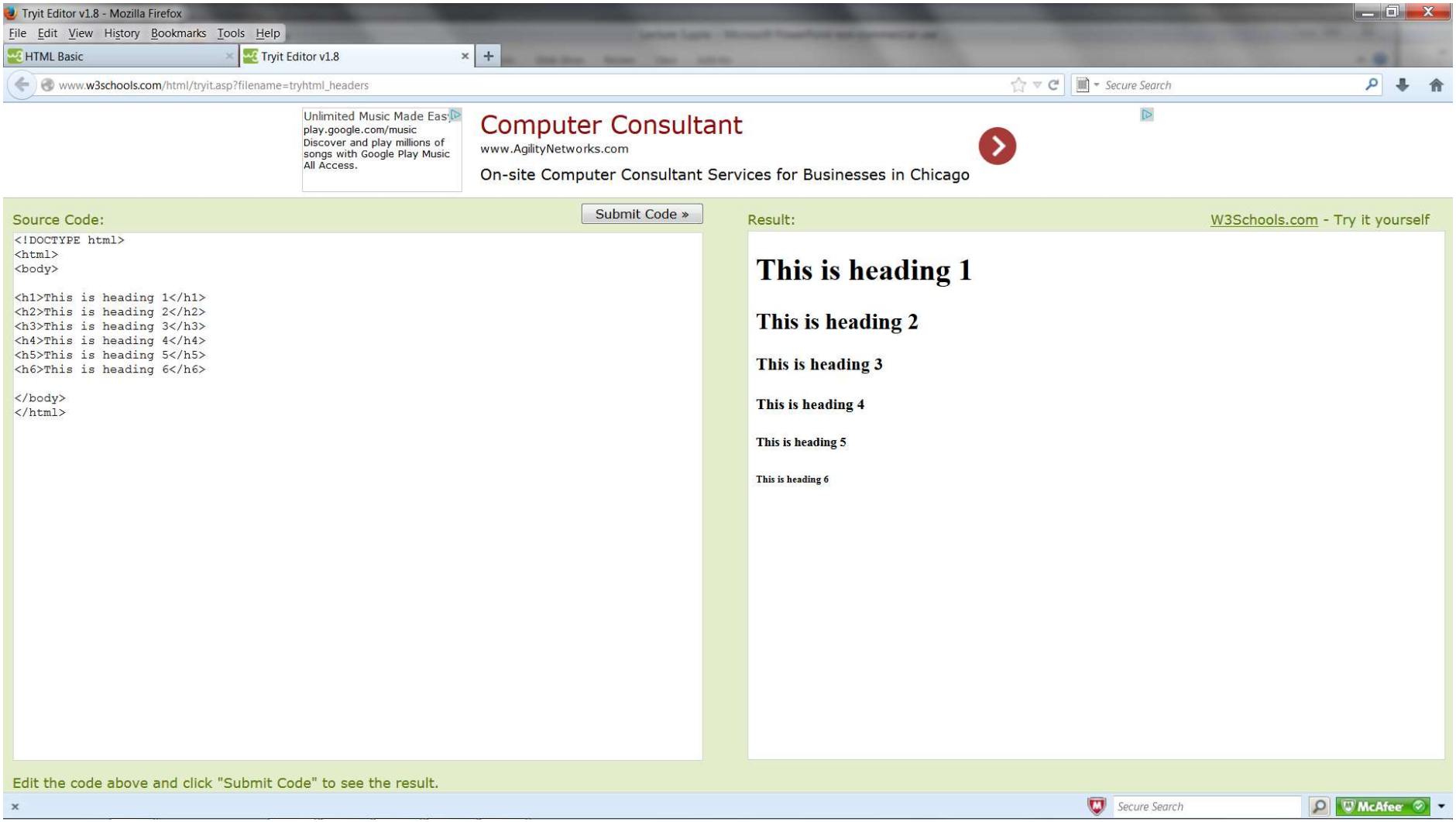
Description

- A *form* contains one or more *controls* such as text boxes, buttons, check boxes, and list boxes.

Web Site for HTML tags Demos

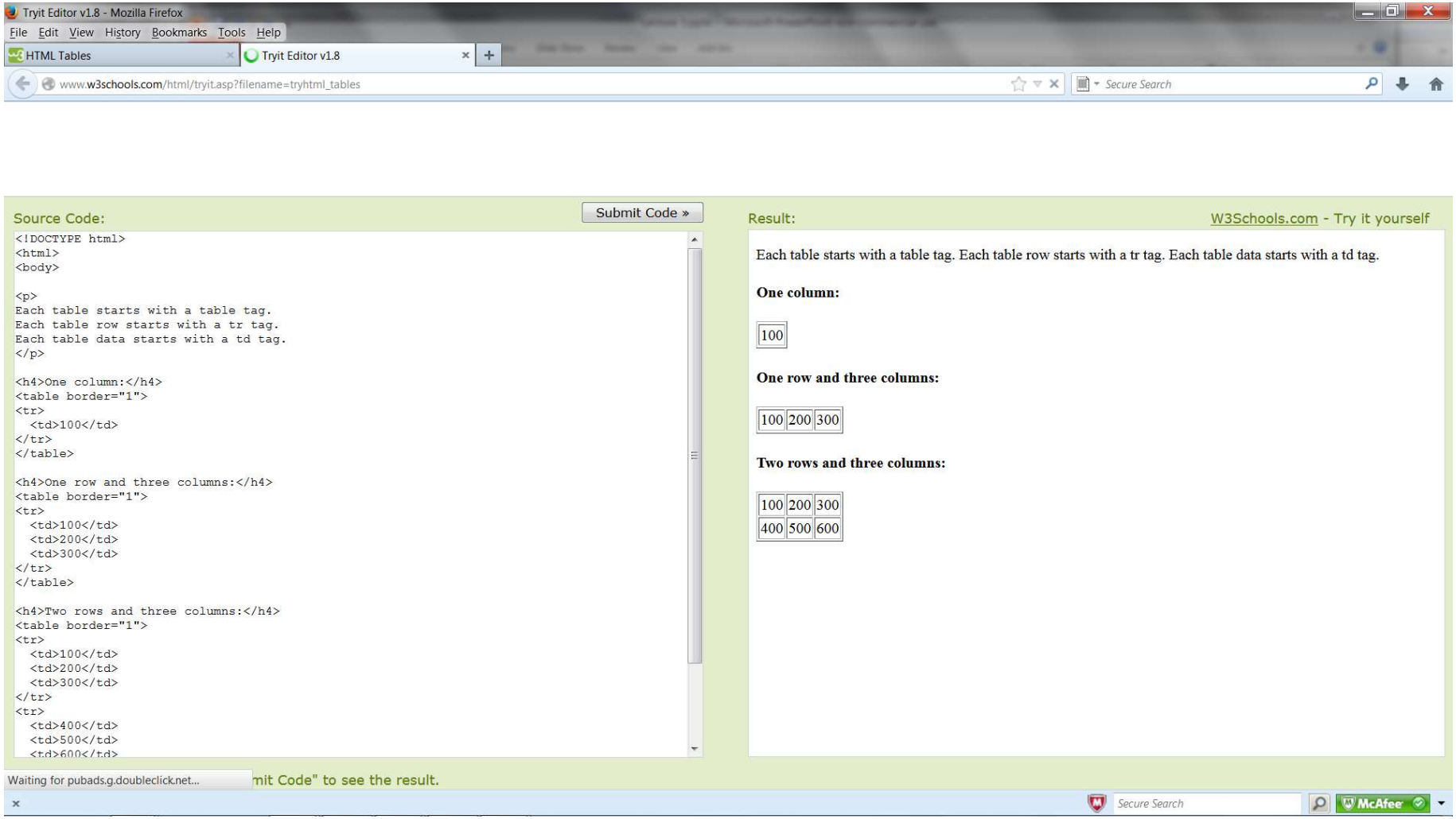


Web Site for HTML tags Demos

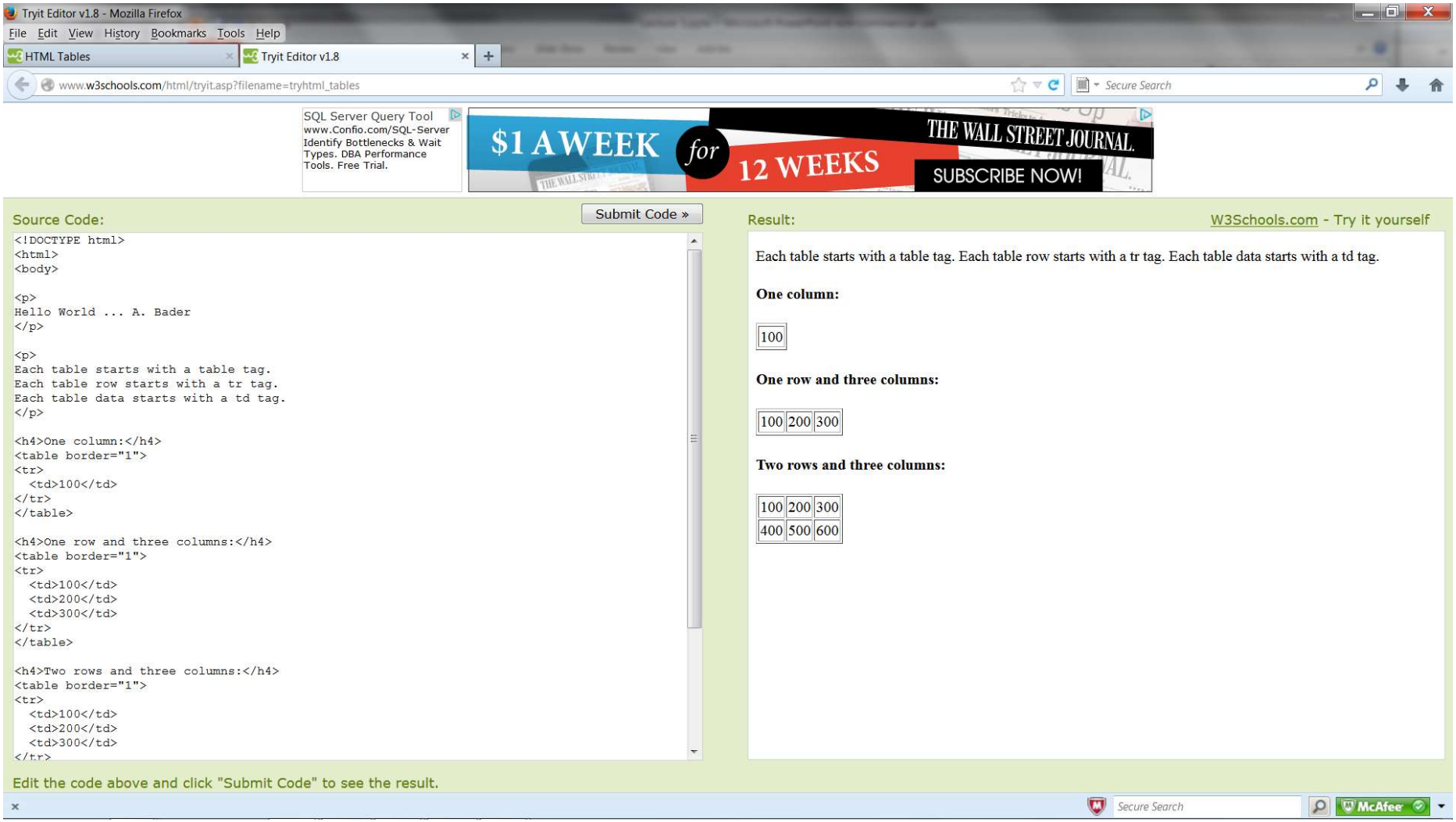


Web Site for HTML tags Demos

Web Site for HTML tags Demos



Web Site for HTML tags Demos



HTML forms

- An HTML form is identified in HTML source by the `<form>` tag

```
<form method=m action=uri>
   … more html, including fields
</form>
```

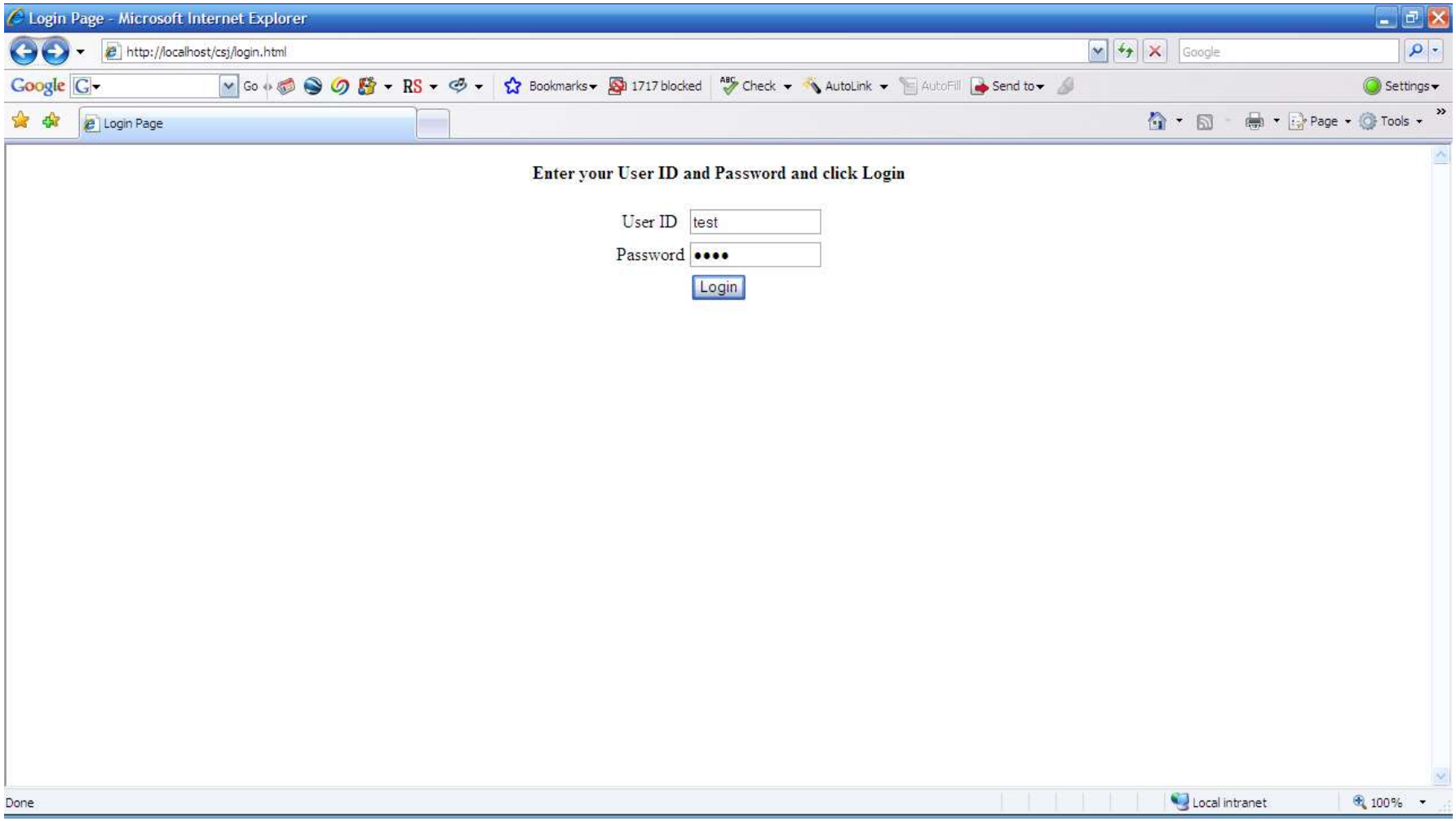
- The method is the HTTP request method to use when submitting the data (POST or GET)

- The action attribute is the server resource that will handle the request. This points to the servlet URL that we will write to handle this form.

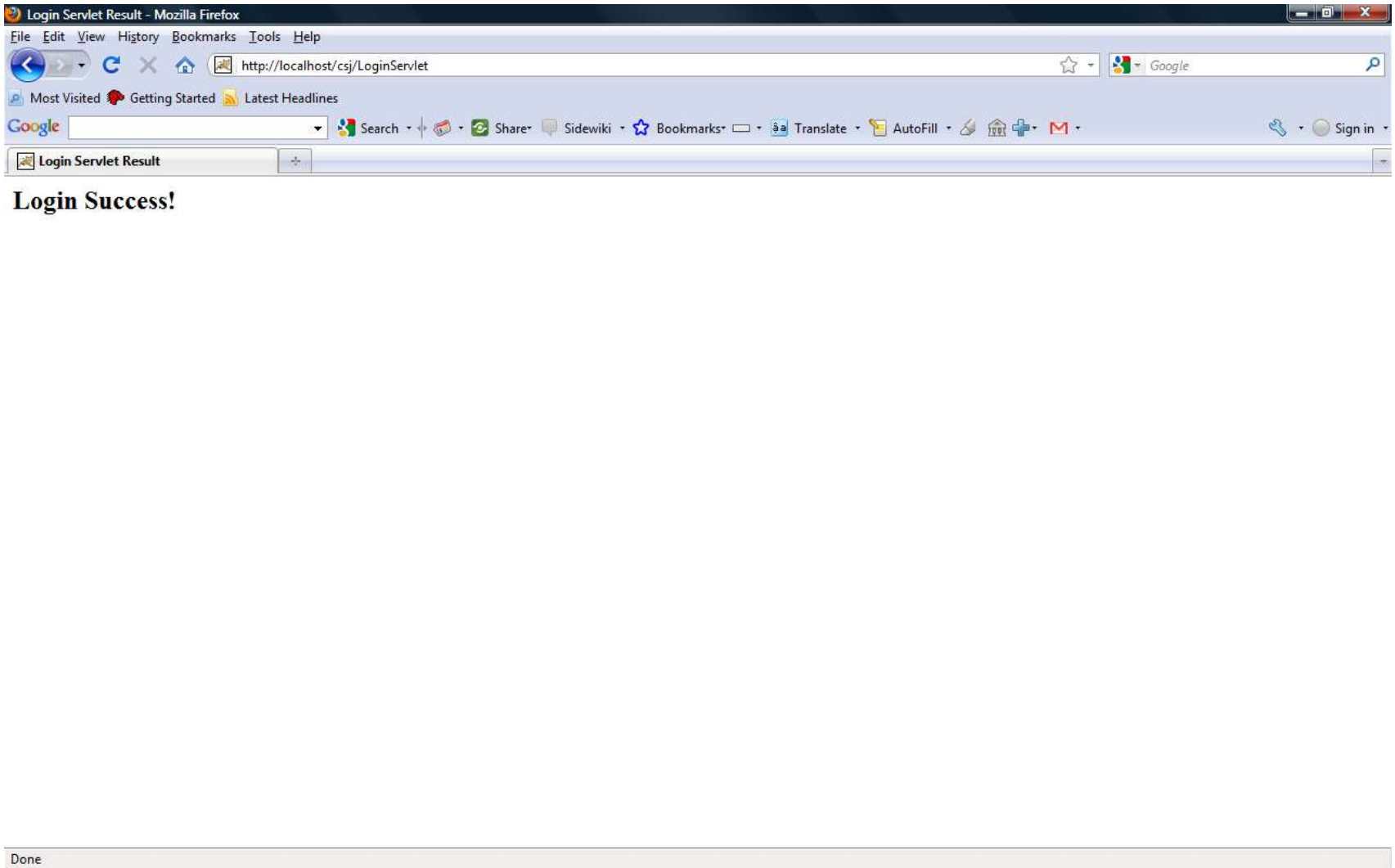
GET vs. POST

- The form data is submitted as part of the HTTP request as a sequence of *URL-encoded* name-value pairs.
- The GET method encodes the name-value pairs onto the request URL (i.e. Visible in the Location of the browser)
- The POST method encodes the name-value pairs in the message body of the request. (invisible to the user, but not encrypted)
- GET is the default method

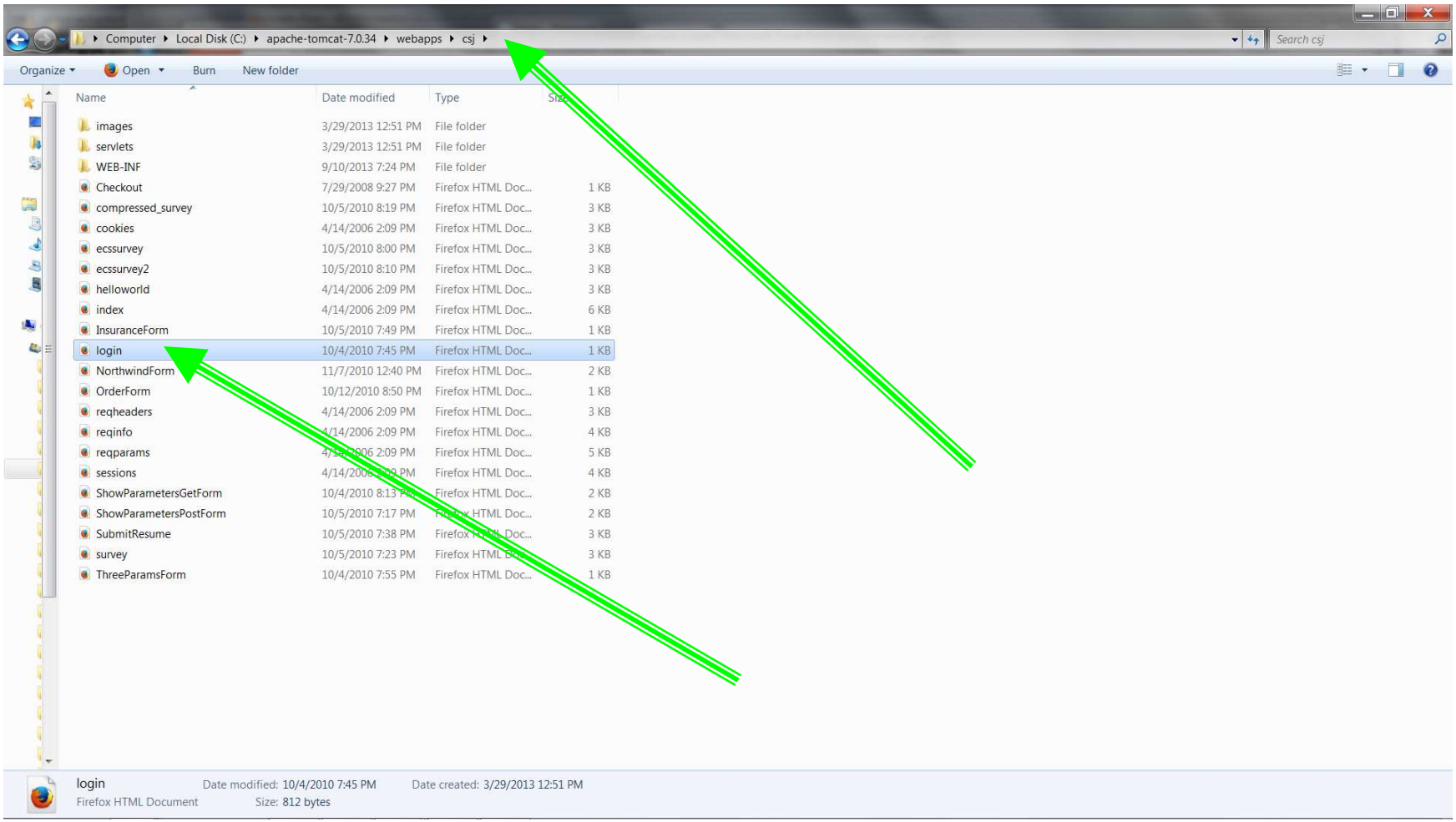
A Login HTML form



A Login HTML form



A Login HTML form



login.html

```
<html>
 <head><title>Login Page</title></head>
 <body>
  <form method="post" action="/csj/LoginServlet">
  <h4>Enter your User ID and Password and click Login</h4>
  <table cellpadding='2' cellspacing='1'>
    <tr><td>User ID</td>
        <td><input type="TEXT" size="15"
   name="userid"></input></td></tr>
    <tr><td>Password</td>
        <td><input type="PASSWORD" size="15"
   name="password"/></td></tr>
    <tr><td colspan='2'>
            <center><input type="SUBMIT" value="Login" /></center>
   </td></tr>
 </table>
 </form>
</body>
</html>
```

LoginServlet.java

LoginServlet - Notepad

File Edit Format View Help

```
import java.util.HashMap;
import java.util.Map;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoginServlet extends HttpServlet {

   protected Map users = new HashMap();
   /**
    * Initializes the servlet with some usernames/password
   */
   public void init() {
            users.put("test", "TEST");
   }

   /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
   * @param request servlet request
   * @param response servlet response
   */
   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, java.io.IOException {
      String userid = request.getParameter("userid");
      String password = request.getParameter("password");
      if(userid != null && userid.length() != 0) {
         userid = userid.trim();
      }
      if(password != null && password.length() != 0) {
         password = password.trim();
      }
      if(userid != null &&
```

Ln 34, Col 18

Request Parameters

- A servlet can access the data that is in the form via Request Parameters

  ```
  String getParameter(String name)
  ```

- Returns the value of a Request Parameter as a string, or null if the parameter doesn't exist
- Works the same for GET or POST
- The name is case sensitive
- Does the hard work of parsing the QUERY_STRING environment variable for you.

HTML Forms: Radio Buttons

- Only one option can be selected at a time

```
Company Type
<input type="radio" name="co_type"
value="private" />Private  
<input type="radio" name="co_type"
value="public" checked="true" />Public  
<input type="radio" name="co_type"
value="government" />Government
```

HTML Forms: Text Input

- Allows arbitrary data to be entered
- The amount can be limited, but doesn't prevent the client from submitting more than the `size` in the request

```
First Name
<input type="text" name="first" size="20"/>
```

HTML Forms: Combo Box

- Allows single or multiple selection via the multiple option.
- Without multiple, it displays a Combo Box
- With multiple, it displays a Select Box

```
<select name="state">
   <option value="IL">Illinois</option>
   <option value="OH">Ohio</option>
   <option value="IN">Indiana</option>
   <option value="WI">Wisconsin</option>
   <option value="MI">Michigan</option>
 </select>
```

HTML Forms: CheckBoxes

- Like the Radio buttons, but allows more than one to be selected

```
<input type="checkbox" name="java">Java 
<input type="checkbox" name="cpp">C++ 
<input type="checkbox" name="c">C 
<input type="checkbox" name="vb">Visual Basic 
<input type="checkbox" name="csharp">C# 
<input type="checkbox" name="perl">Perl 
```

HTML Forms: TextArea

- Allows for more text to be entered than text inputs, over multiple rows.

```
Comments
<textarea name="comments" rows="2" cols="40">
</textarea>
```

HTML Forms: Password Fields

- Just like a text input, but will display a ‘*’ instead of the character typed.
- WARNING! – the value of the field is not encrypted. The data is just not displayed to the user. To encrypt your data, use SSL (Secure Sockets Layer).

More ServletRequest methods

`Enumeration` **`getParameterNames`**`()`

- Returns an Enumeration of the parameter names in this request

`String[]` **`getParameterValues`**`(String name)`

- Returns a String array of values for a given Parameter name. This method is needed if you have parameters with multiple results, like a Select input.
- Returns null if the Parameter doesn't exist

Retrieving all form data

```
Enumeration parameters = request.getParameterNames();
if(parameters.hasMoreElements()) {
   out.println("<tr><th>Parameter Name</th><th>Parameter" +
               "Value</th></tr>");
}
while(parameters.hasMoreElements()) {
   String parameter = (String)parameters.nextElement();
   // get the parameter values
   String[] values = request.getParameterValues(parameter);
   if(values != null) {
        for(int i = 0; i < values.length; i++) {
                out.println("<tr><td><b><fontcolor=\"blue\">" +
                        parameter + "</font></b></td>" +
                        "<td>" + values[i] + "</td></tr>");
        }
   }
}
```

A Login Servlet

```
import java.util.HashMap;
import java.util.Map;
import javax.servlet.*;
import javax.servlet.http.*;
public class LoginServlet extends HttpServlet {
   protected Map users = new HashMap();
   /**
    * Initializes the servlet with some usernames/passwords
    */
   public void init() {

        users.put("test", "TEST");
}
```

A Login Servlet (cont.)

- Since we want to handle both GET and POST requests, just use the same boilerplate code for doGet() and doPost(), calling the same processRequest() method.

```
protected void doGet(
  HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, java.io.IOException {
      processRequest(request, response);
}
```

A Login Servlet (cont.)

```
protected void processRequest(
   HttpServletRequest request,
   HttpServletResponse response)
   throws ServletException, java.io.IOException {
   String userid = request.getParameter("userid");
   String password = request.getParameter("password");
   if(userid != null && userid.length() != 0) {
       userid = userid.trim();
   }
   if(password != null && password.length() != 0) {
       password = password.trim();
   }
```

A Login Servlet (cont.)

- processRequest() continued…

```
if(userid != null && password != null) {
        String realpassword = (String)users.get(userid);
        if(realpassword != null &&
        realpassword.equals(password)) {
                showPage(response, "Login Success!");
        } else {
                showPage(response, "Login Failure! Username or"
                + "password is incorrect");
        }
} else {
        showPage(response, "Login Failure! You must supply a"
                + "username and password");
}
}
```

A Login Servlet (cont.)

- Showing the result

```
protected void showPage(HttpServletResponse response, String
  message)
  throws ServletException, java.io.IOException {
  response.setContentType("text/html");
  java.io.PrintWriter out = response.getWriter();
  out.println("<html>");
  out.println("<head>");
  out.println("<title>Login Servlet Result</title>");
  out.println("</head>");
  out.println("<body>");
  out.println("<h2>" + message + "</h2>"); out.println("</body>");
  out.println("</html>");
  out.close();
}
```

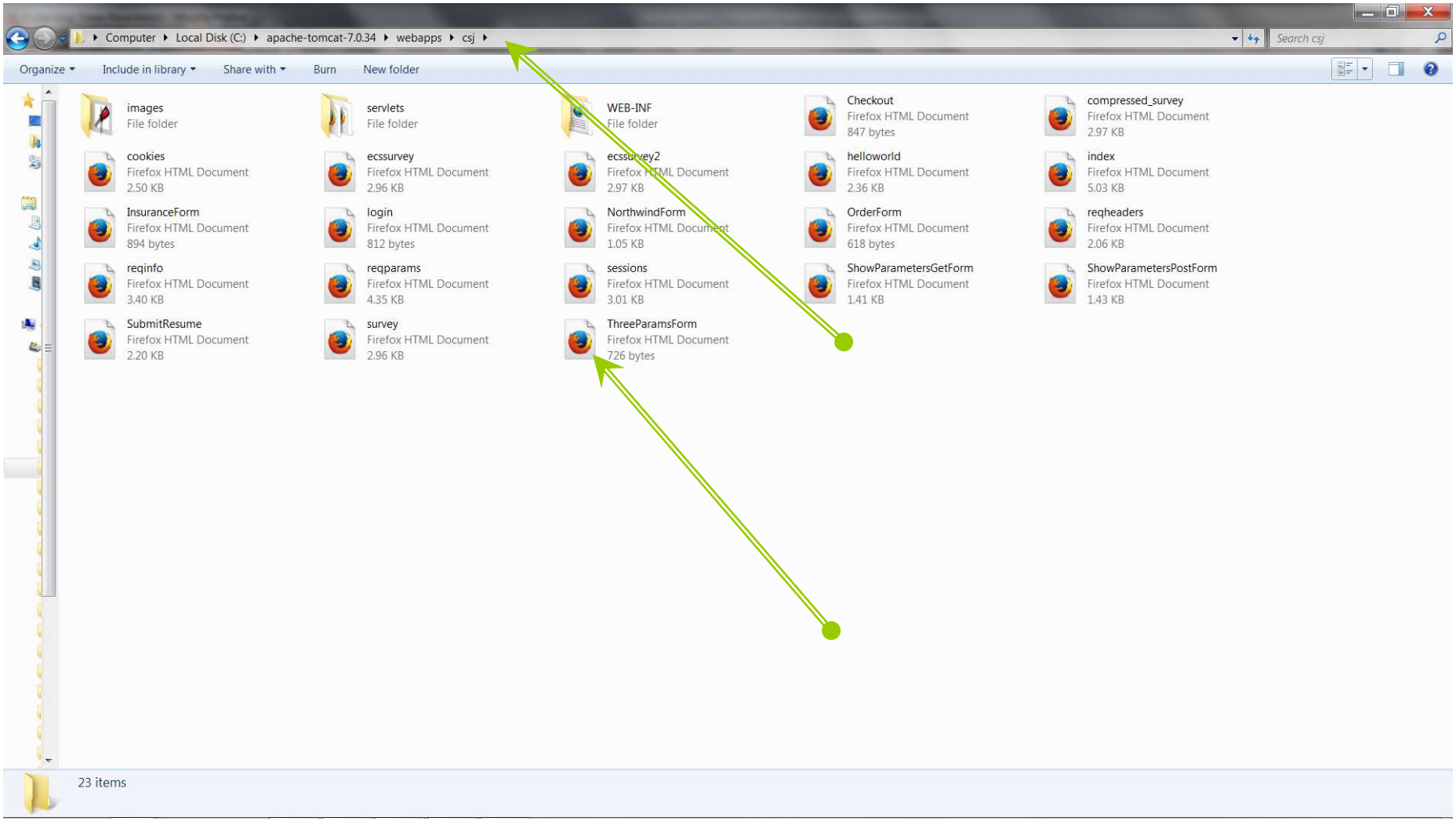
Deploy the Login Servlet

- Compile the code
- Move the LoginServlet.class file to `<app_home>/WEB-INF/classes/`
- Move the login.html file to `<app_home>`
- Point your browser to:
  - http://localhost/csj/login.html
- Try to log in!

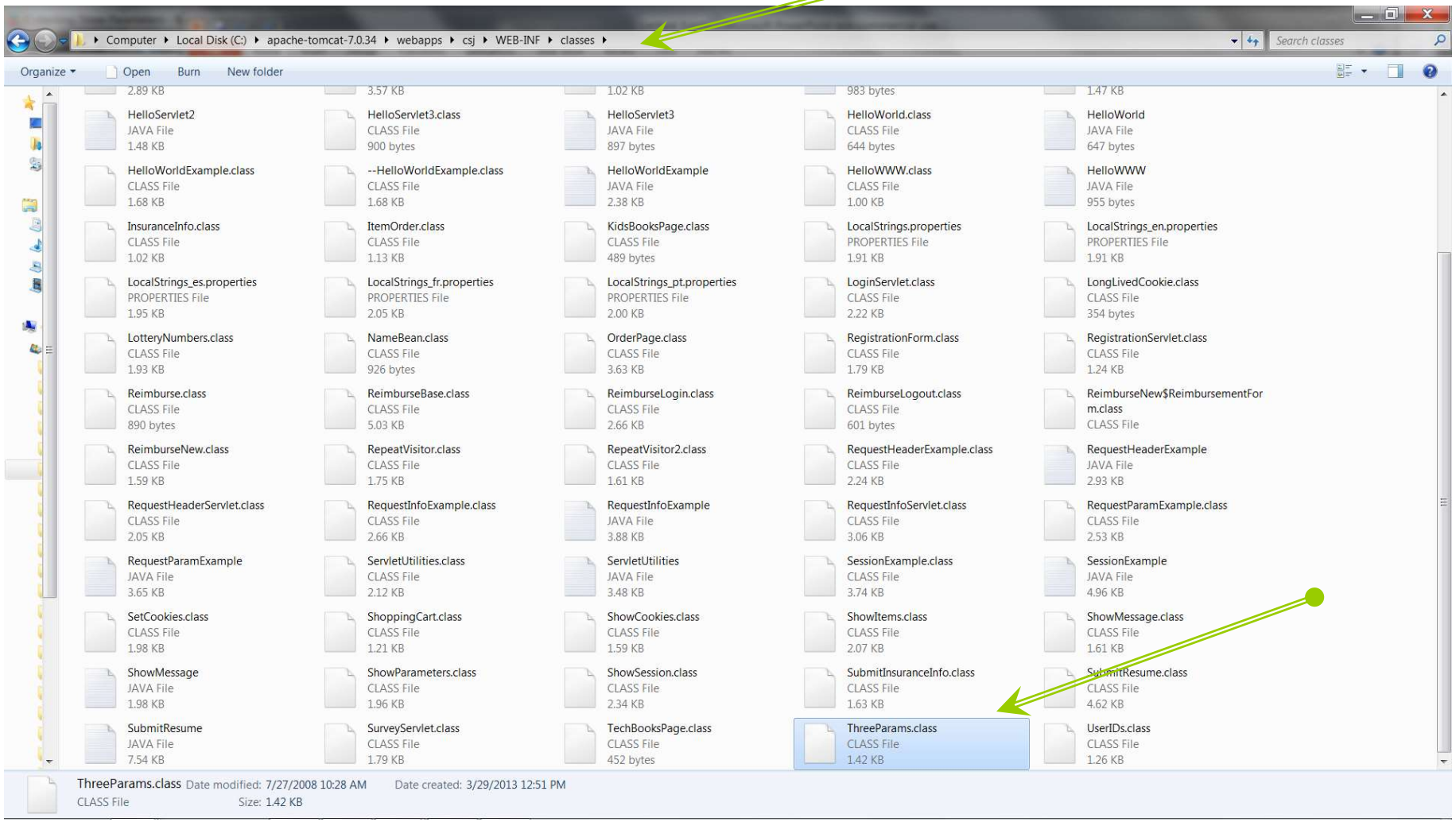
ThreeParams Servlet

- How to read 3 parameters passed from html page?
  - ThreeParams.java
  - ThreeParamsForm.html
    - http://localhost/csj/ThreeParamsForm.htm

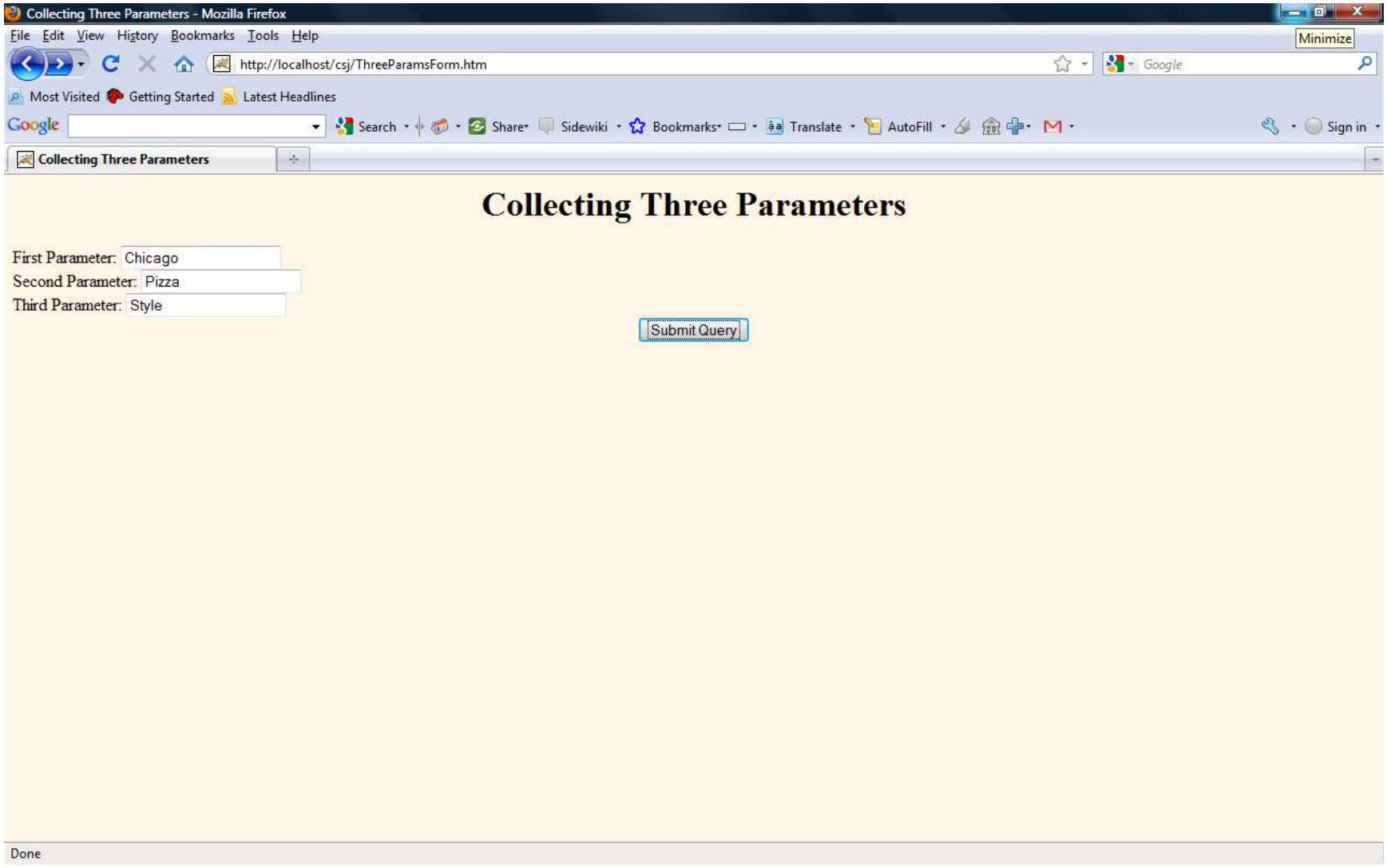
ThreeParams Servlet



ThreeParams Servlet



ThreeParams Servlet



ThreeParams Servlet

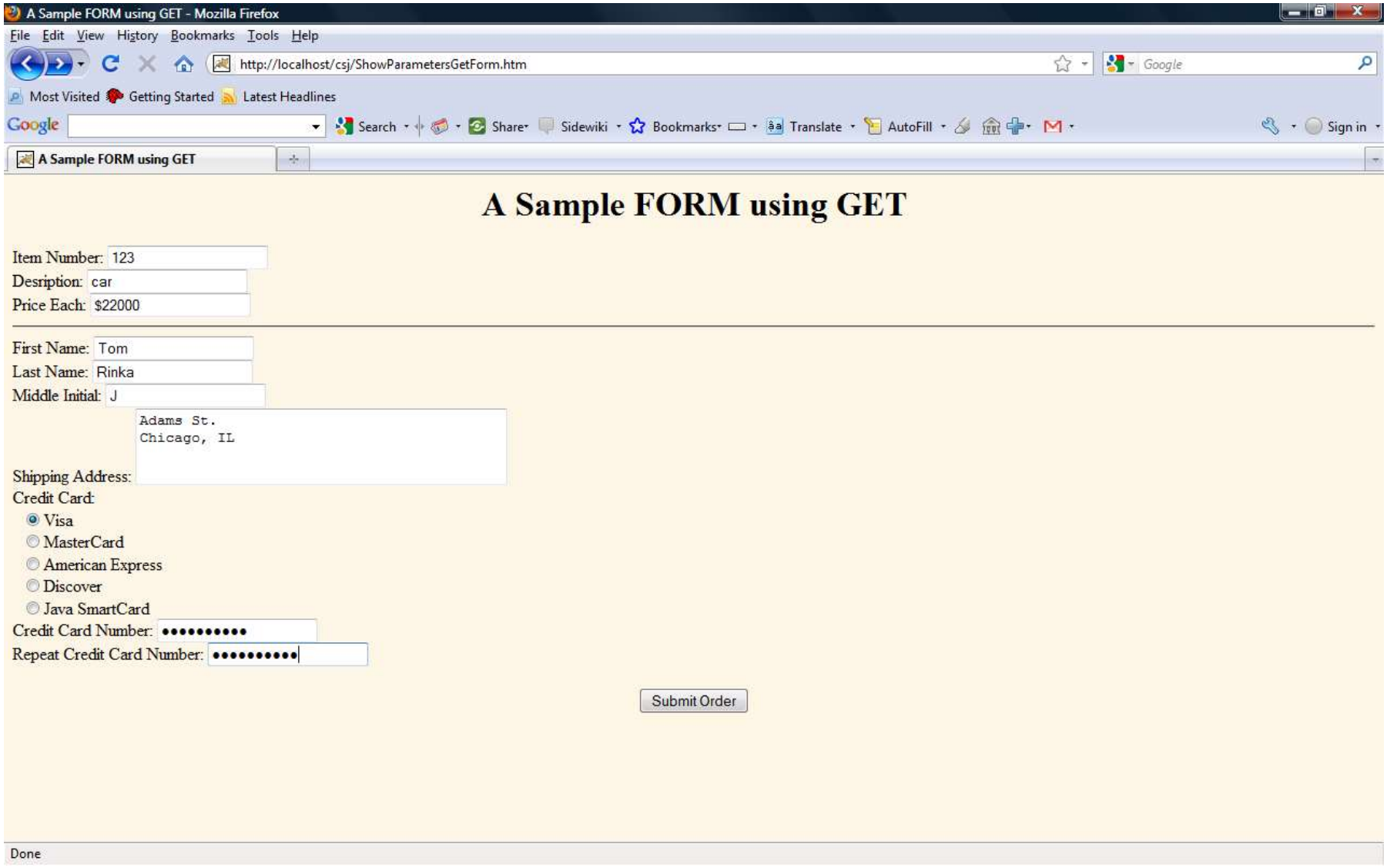
http://localhost/csj/ShowParametersGetForm.htm

- How to read N parameters passed from html page?
  - http://localhost/csj/ShowParametersGetForm.htm
  - ShowParameters.java

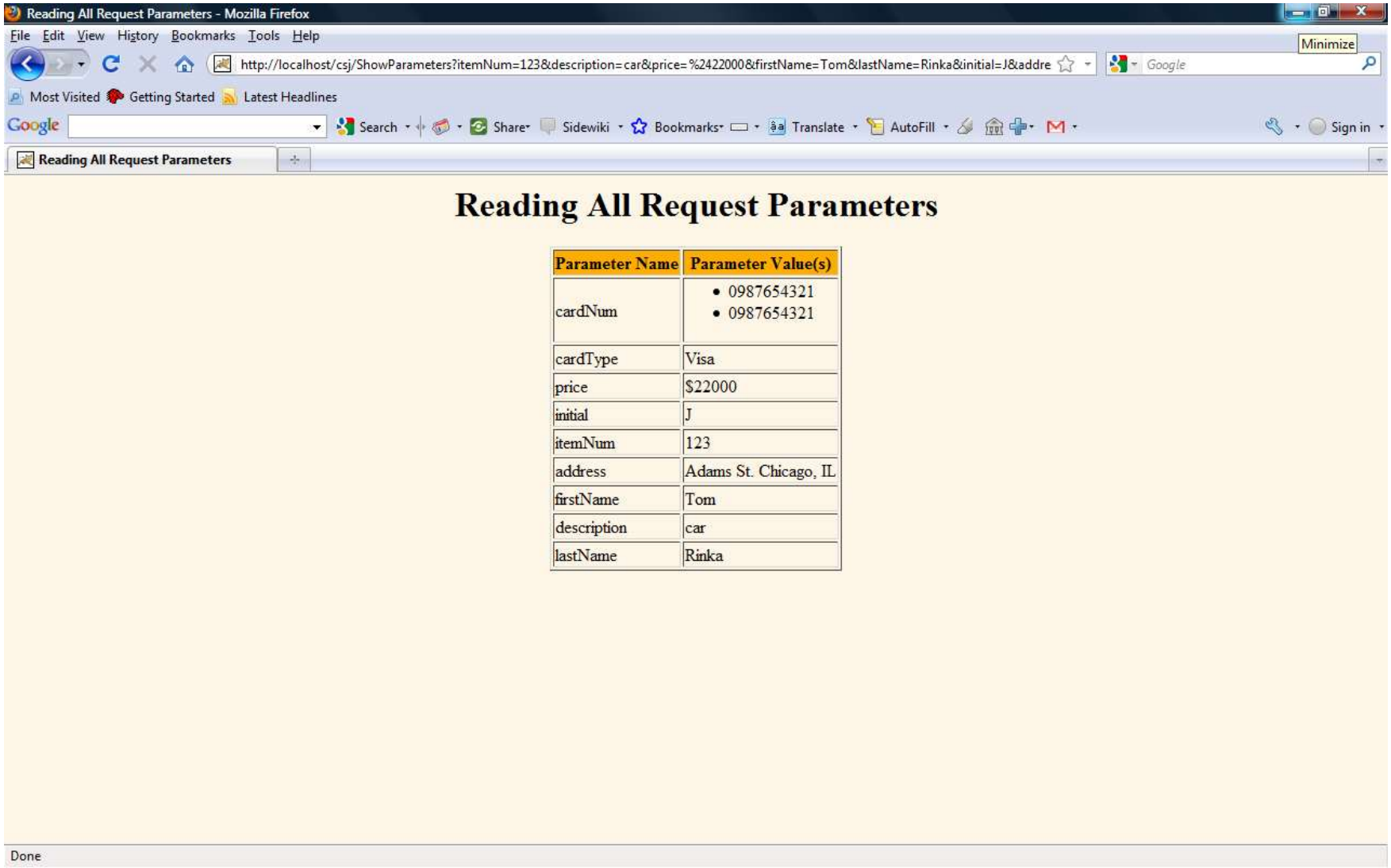
http://localhost/csj/ShowParametersGetForm.htm

- We will look at two examples
  - Using Get
  - Using Post
- When using Get in the form …
  - Watch how the parameters are passed next to the URL

http://localhost/csj/ShowParametersGetForm.htm



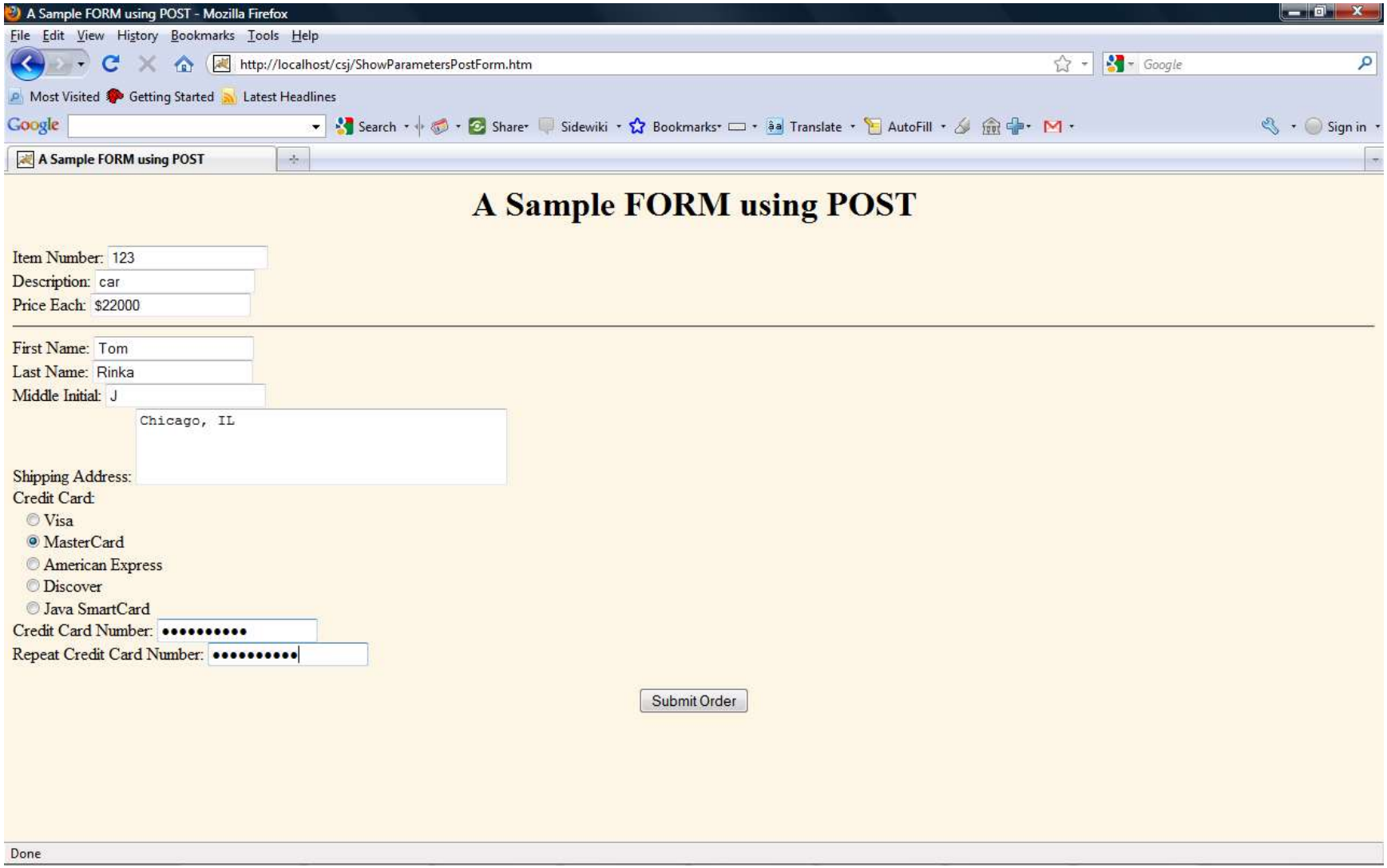
http://localhost/csj/ShowParametersGetForm.htm



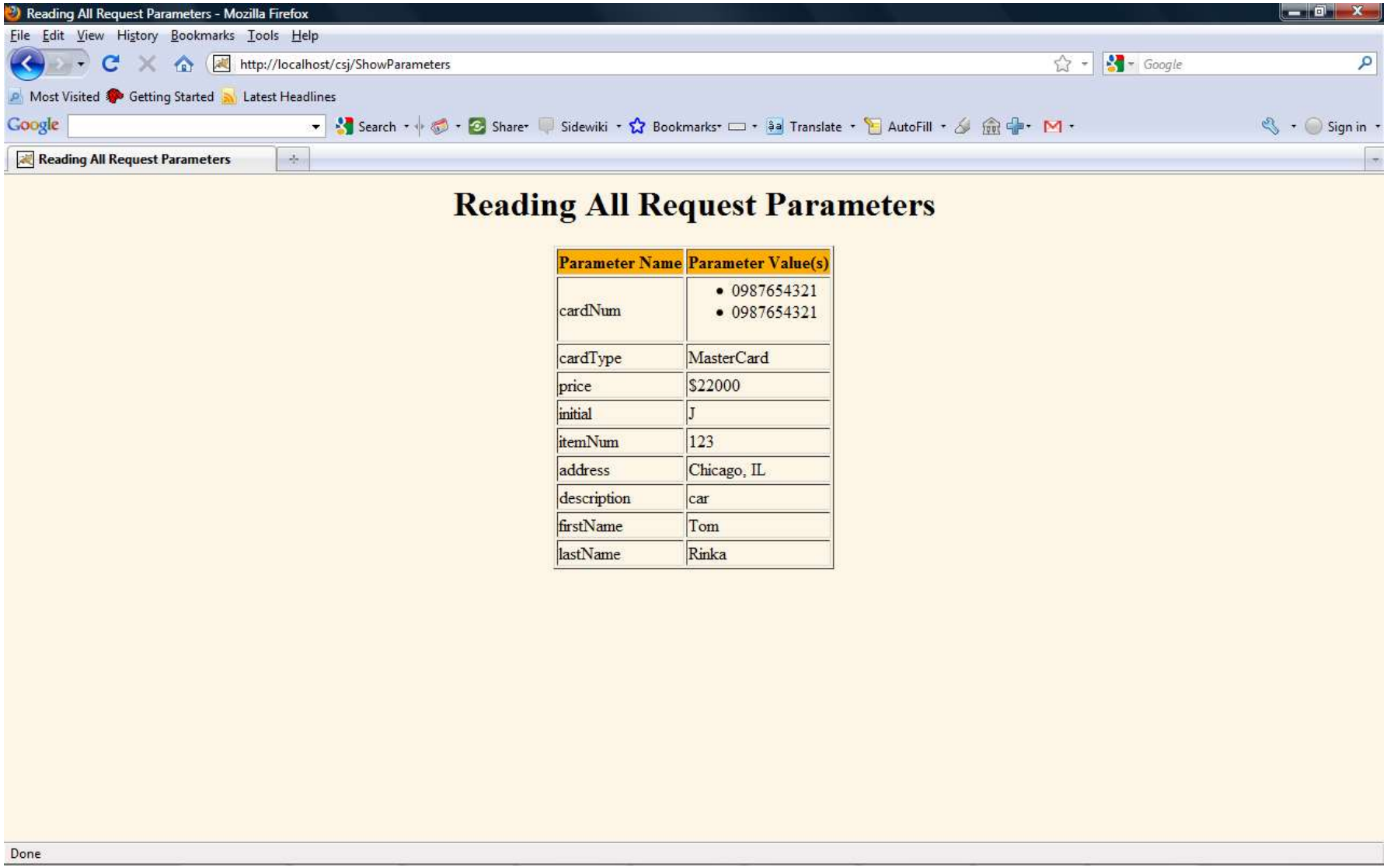
http://localhost/csj/ShowParametersPostForm.htm

- When using Post in the form …
  - Watch how the parameters are passed separate from the URL
  - Note that doPost called doGet to avoid repeating the same code

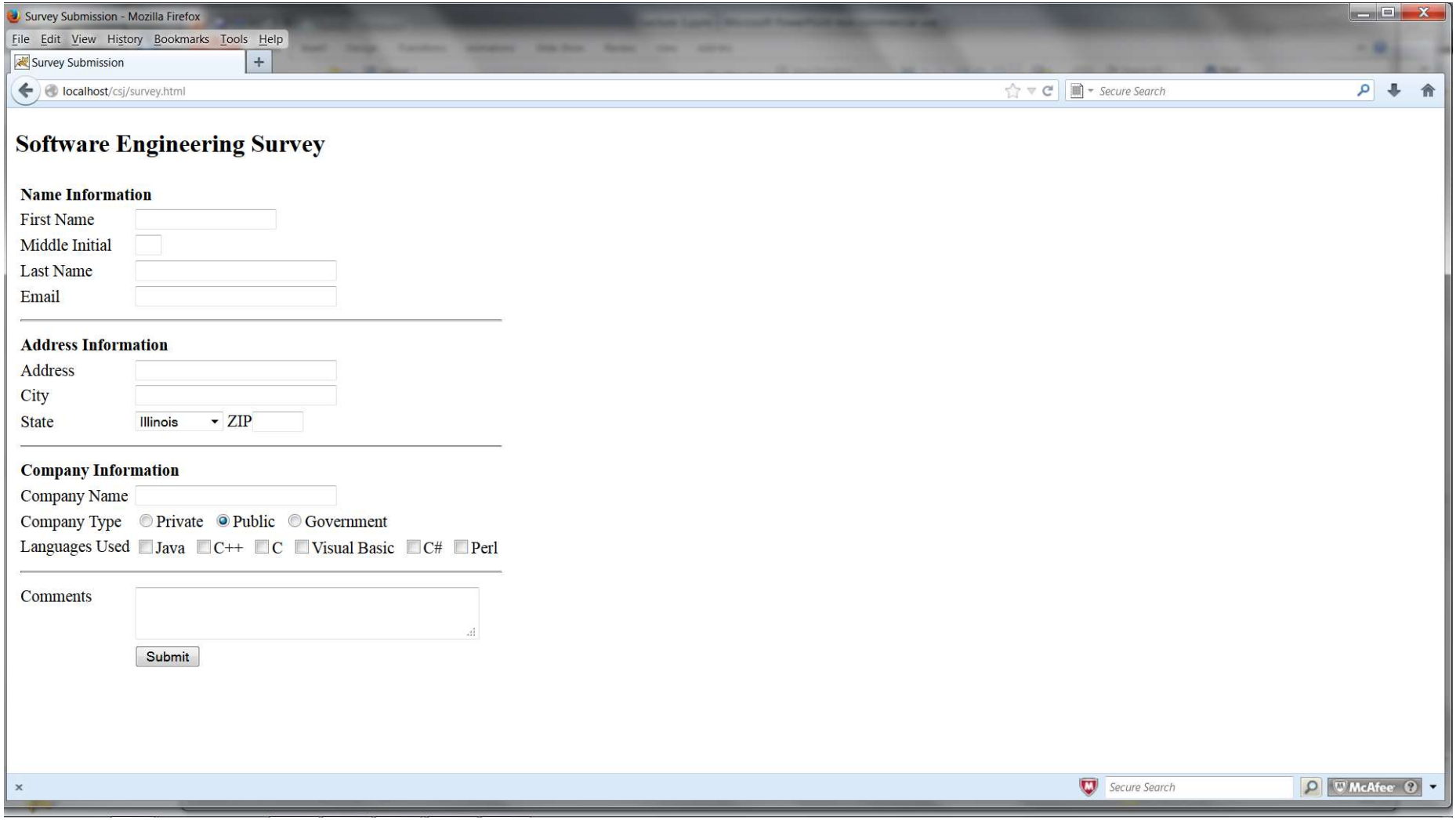
http://localhost/csj/ShowParametersPostForm.htm



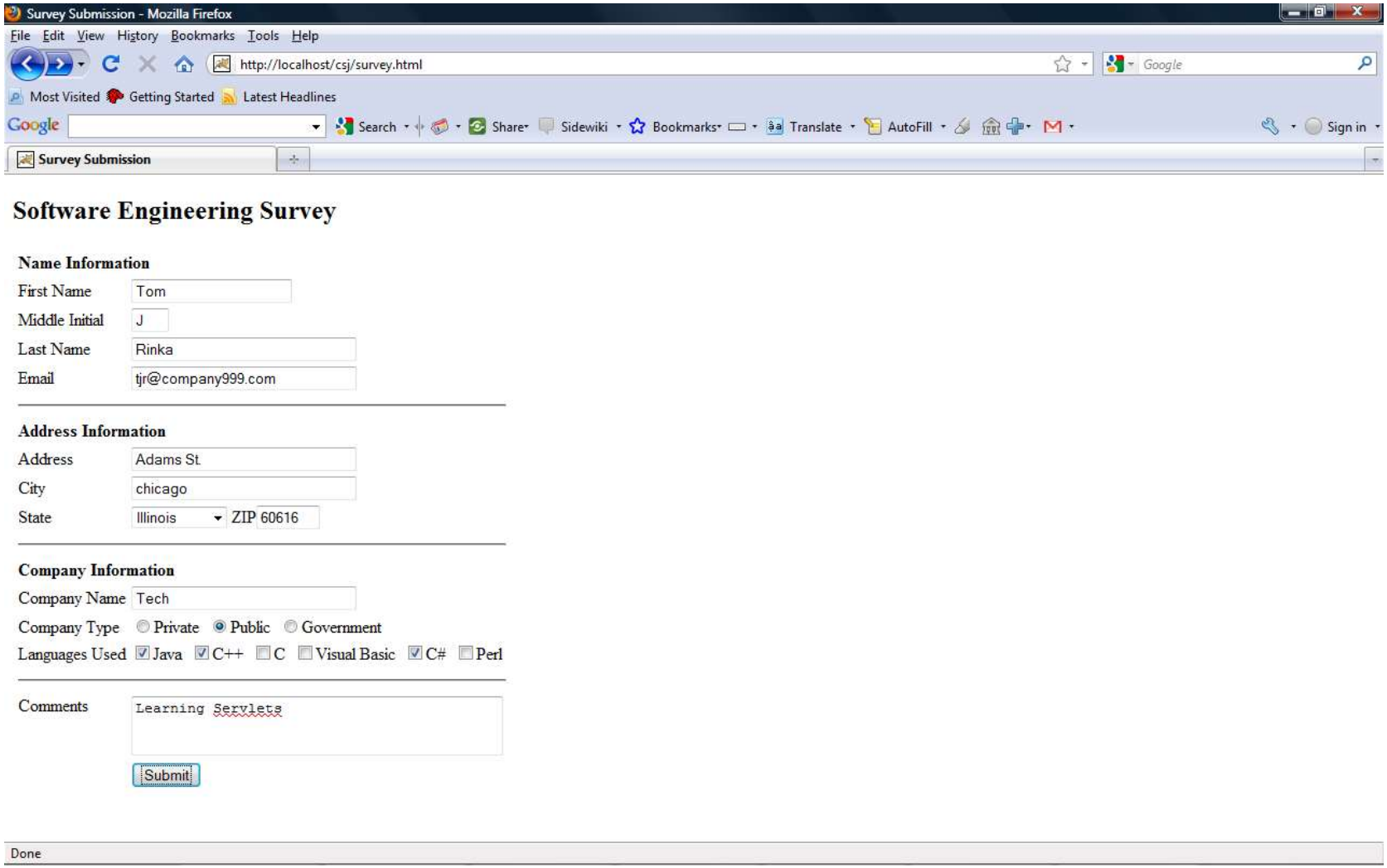
http://localhost/csj/ShowParametersPostForm.htm



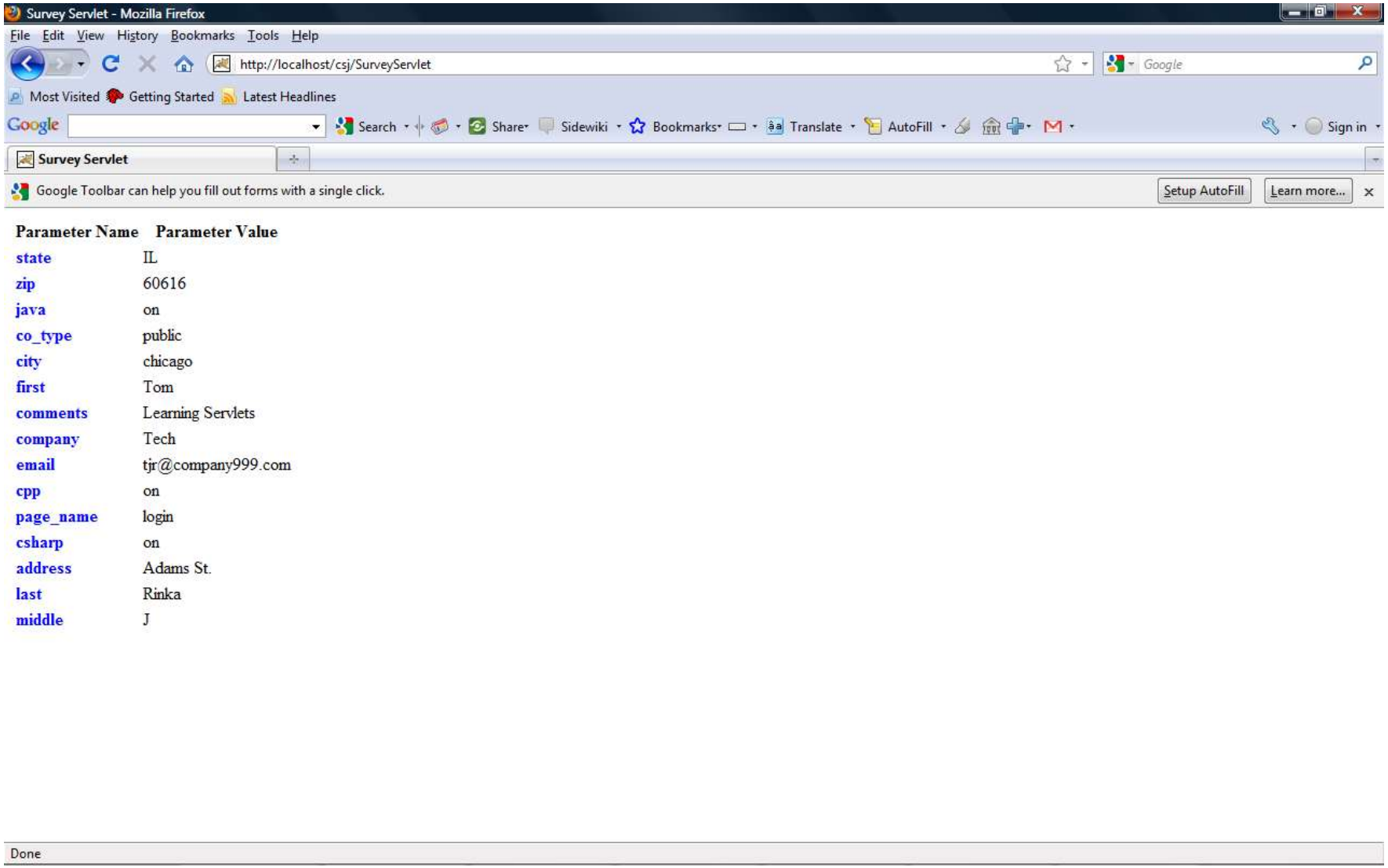
A more advanced Form, A Survey
http://localhost/csj/survey.html



After submitting the form…



After submitting the form…



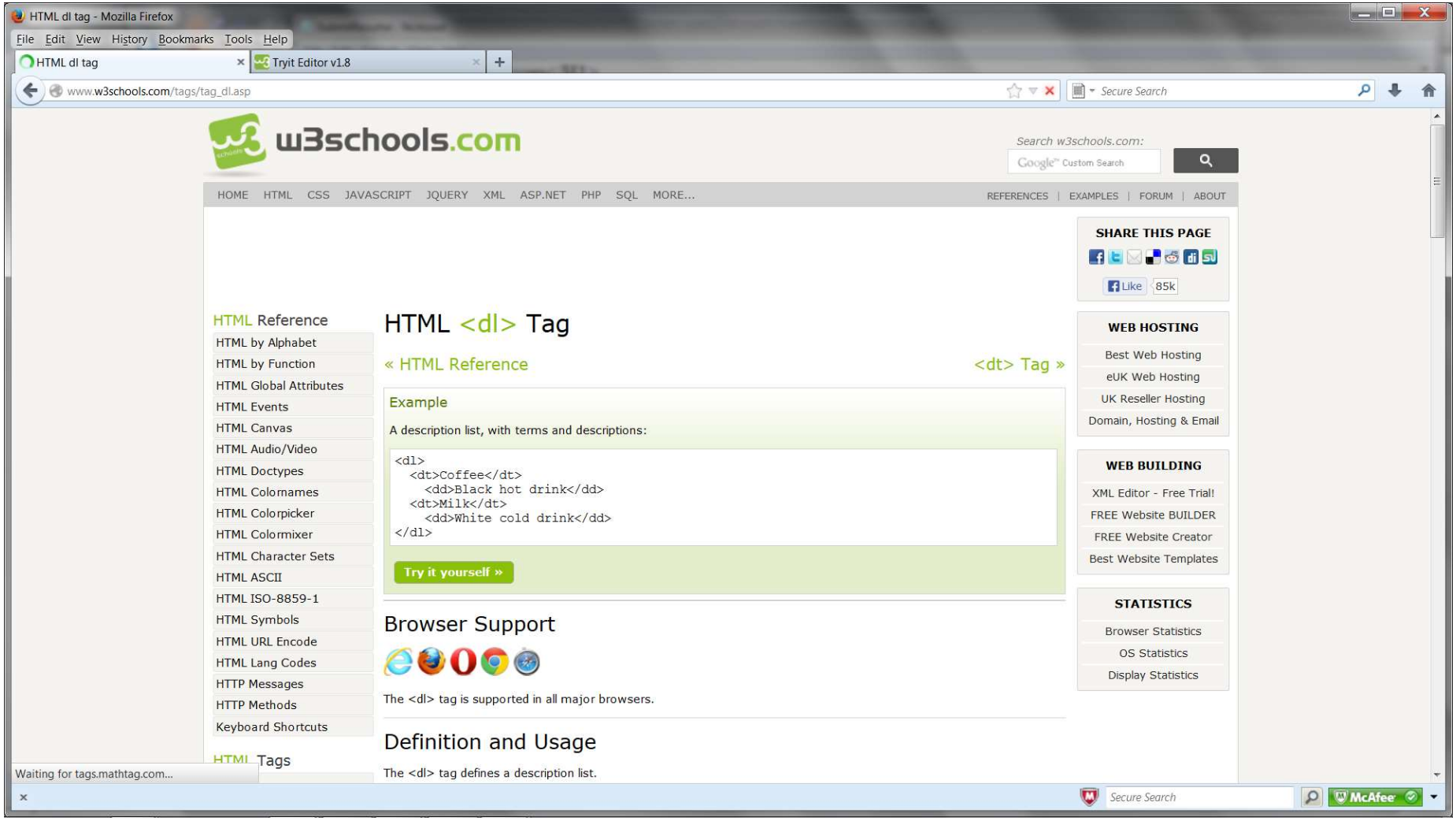
Using Default Values When Parameters are missing or Malformed

- What the servlet should do when dealing with bad input?
  1. Use Default Values
  2. Redisplay the form for the user to type again

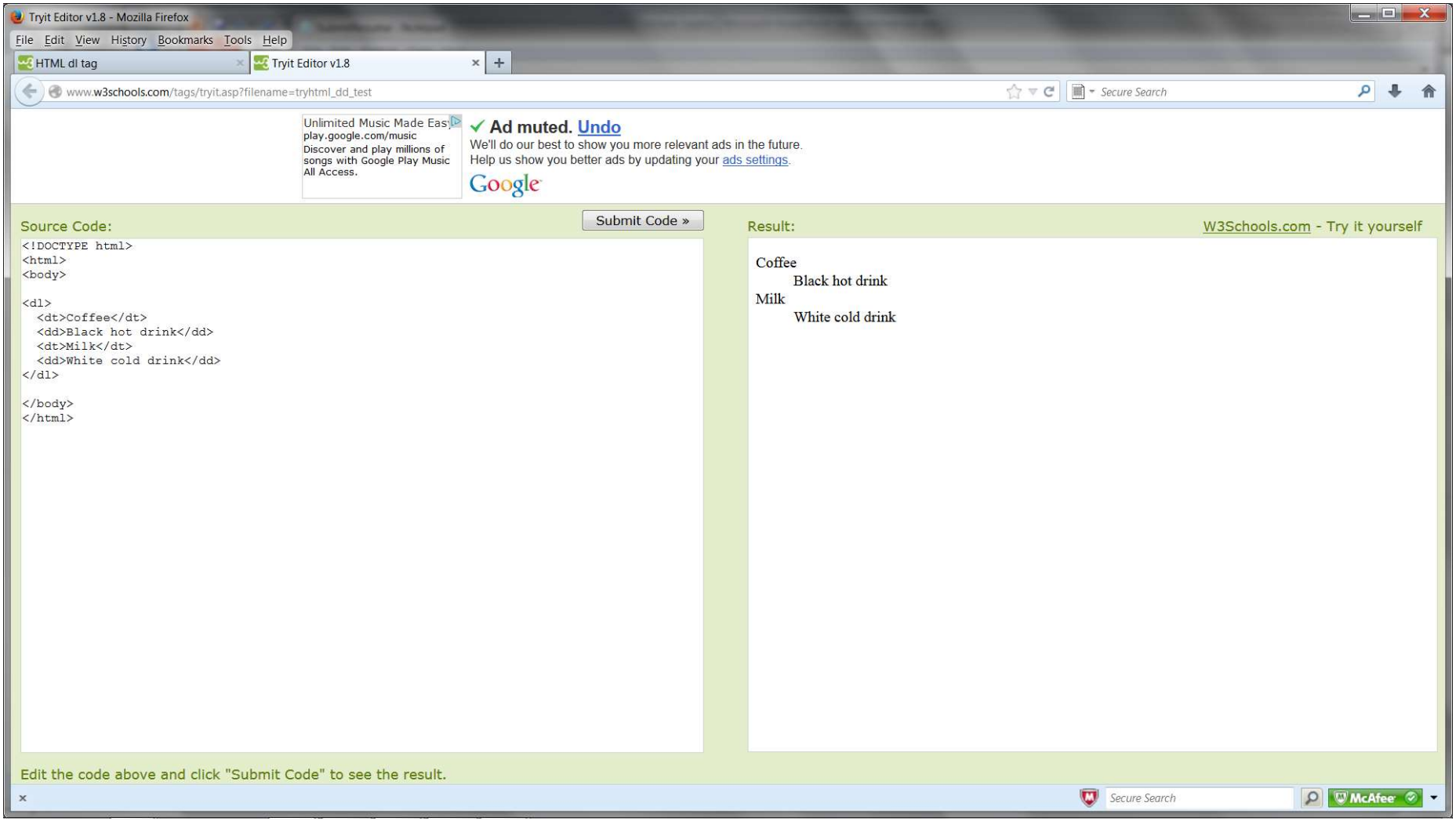
Example: Resume Submission

- http://localhost/csj/SubmitResume.htm
- SubmitResume.java Servlet

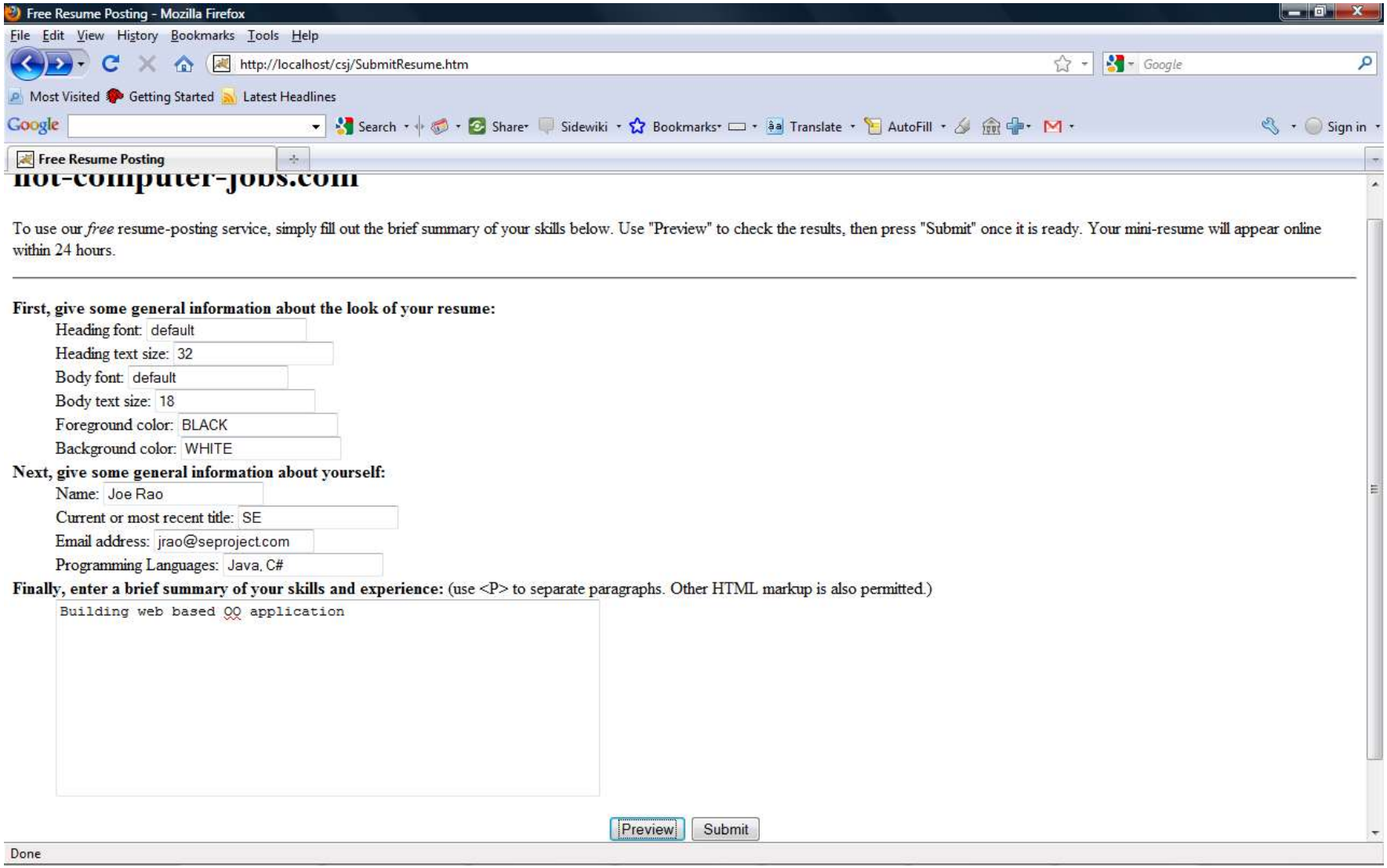
In case you never used DL, DD, DT HTML tags … visit this website to get demo



In case you never used DL, DD, DT HTML tags … visit this website to get demo



http://localhost/csj/SubmitResume.htm



Once you Click Preview

Joe Rao
SE
jrao@seproject.com

Programming Languages

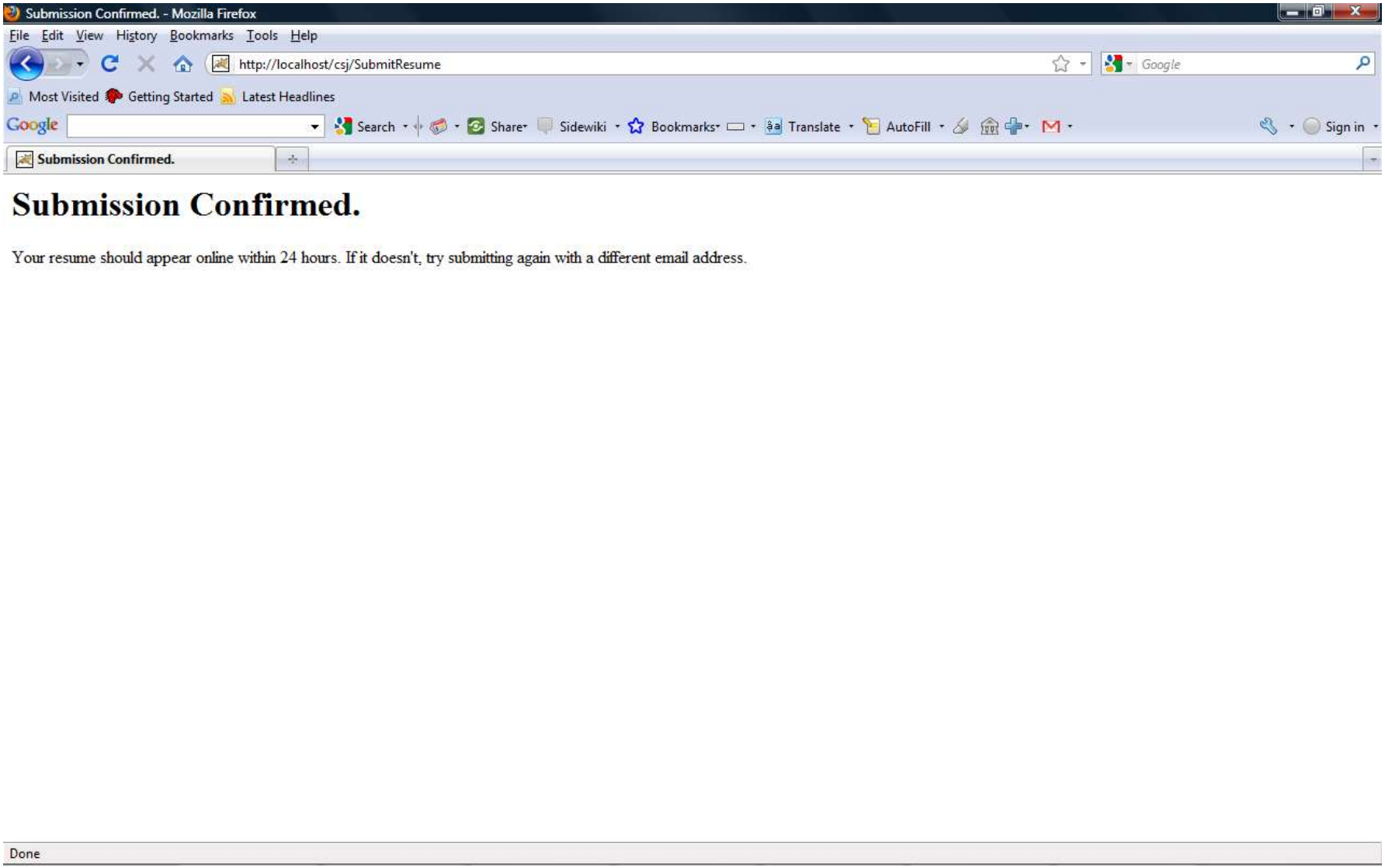
- Java
- C#

Skills and Experience

Building web based OO application

Done

Once you Click Submit

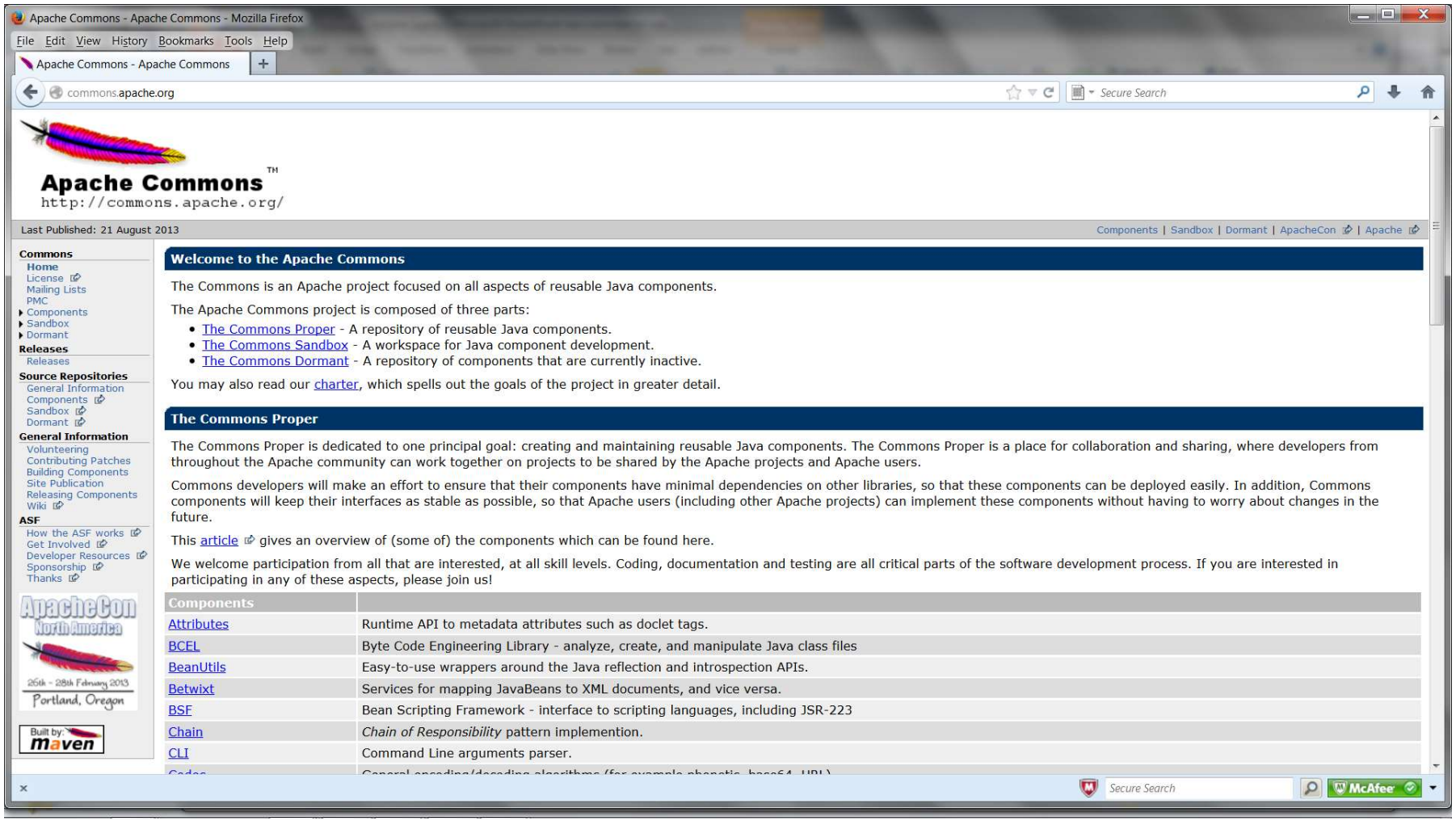


Using BeanUtils to access Request Parameters

- How to automatically populate a bean based on the request parameters?
  - Apache/tomcat has common packages that could be downloaded and automatically populate a bean according to incoming request parameters

- You need to download the following jars from http://commons.apache.org/
  - BeanUtils
  - Collections
  - Logging

Using BeanUtils to access Request Parameters



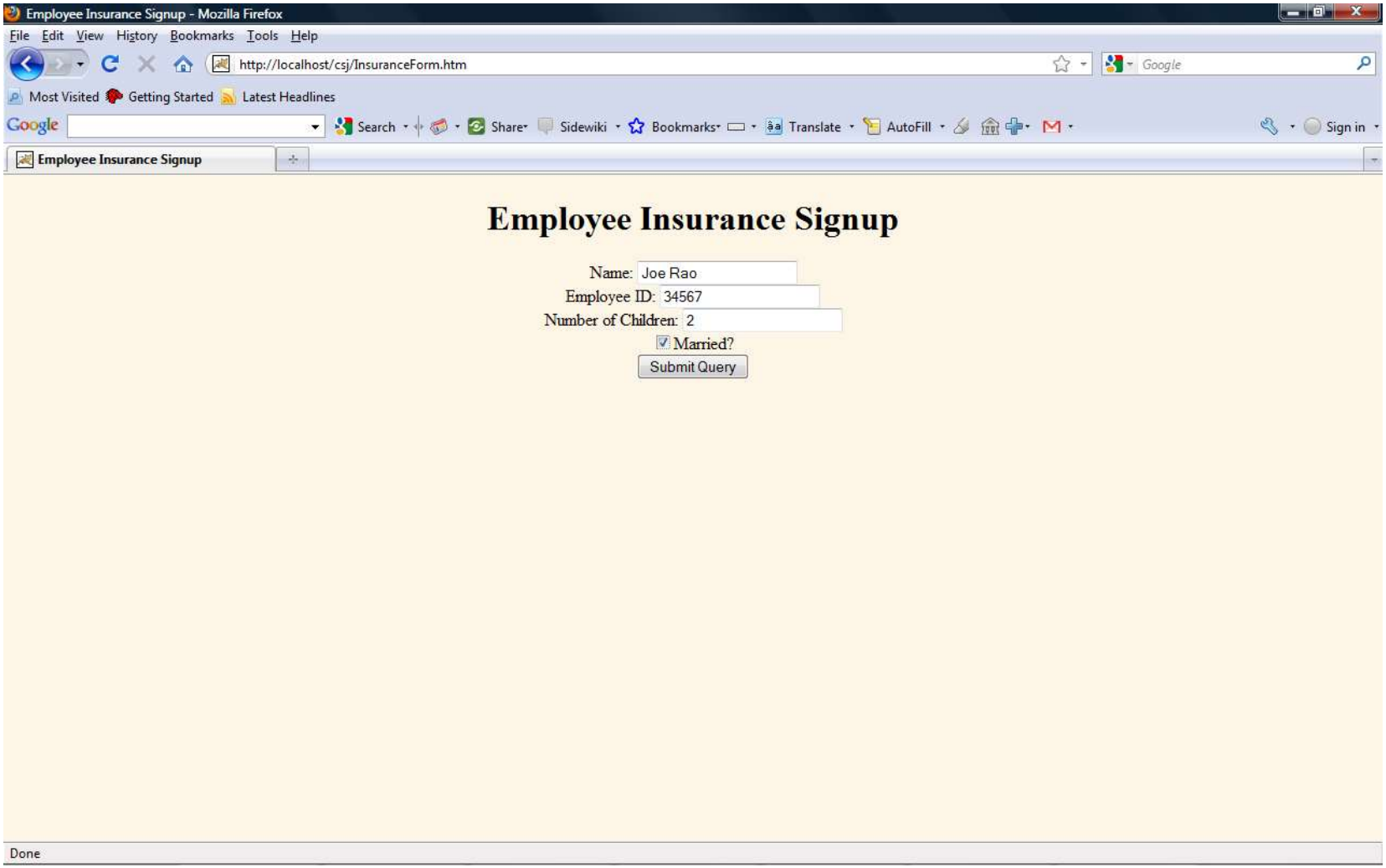
Using BeanUtils to access Request Parameters

- Once you unzip these files you need to copy the following jar files to:
  - C:\apache-tomcat-7.0.34\lib
    - commons-beanutils-1.8.3
    - commons-beanutils-bean-collections-1.8.3
    - commons-beanutils-core-1.8.3
    - commons-collections-3.2.1
    - commons-logging-1.1.1

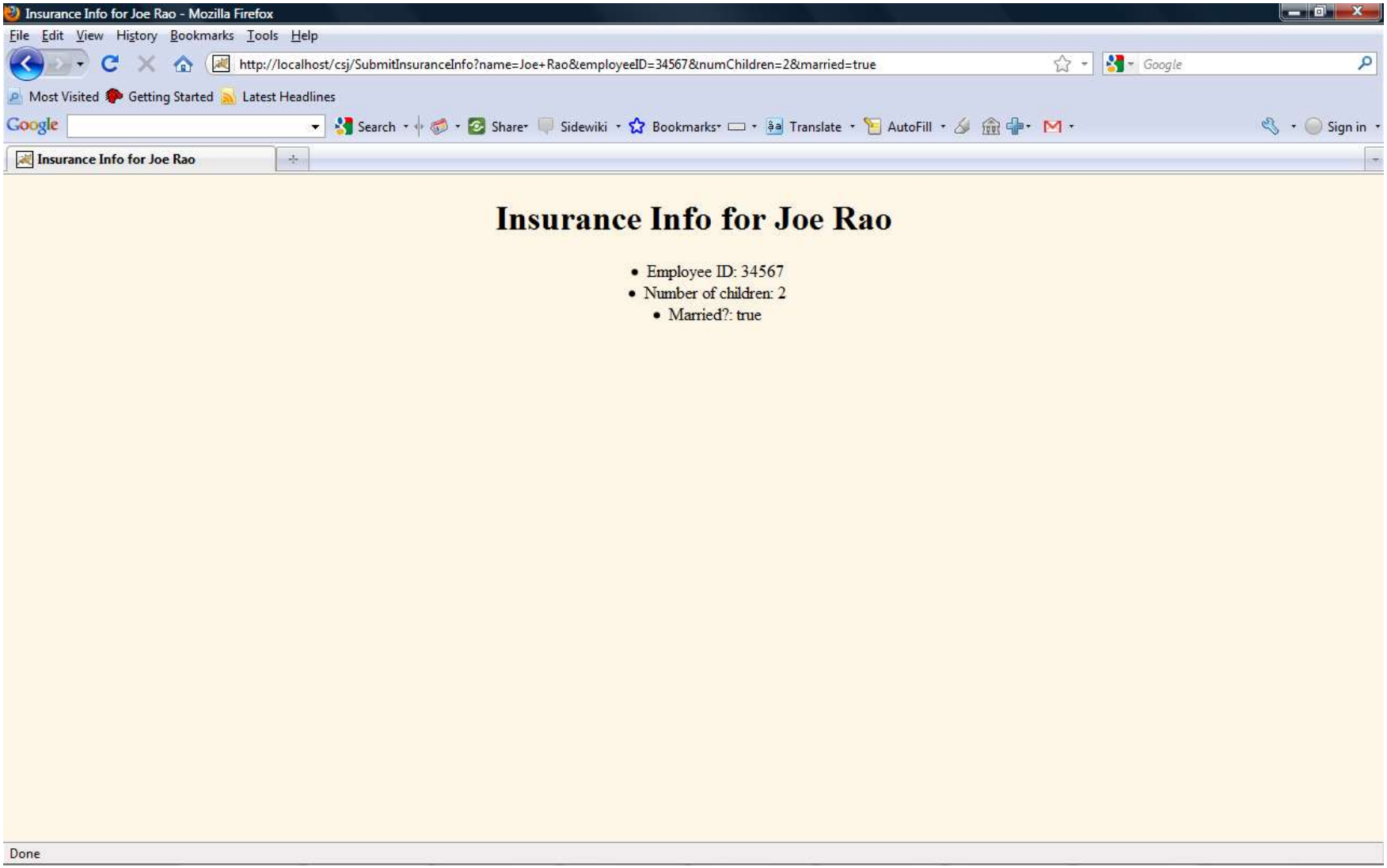
Example: BeanUtils to access Request Parameters

- The following example has the following files:
  - InsuranceForm.htm
  - BeanUtilities.java
    - Has a method BeanUtilities.populateBean that lets you fill in The required information in a single method call.
      - BeanUtilities.populateBean calls BeanUtils.populate from tomcat/apache Common package
  - InsuranceInfo.java (This is your simple Bean that will be populated BeanUtils.populate)
  - SubmitInsuranceInfo.java

Using BeanUtils to access Request Parameters
http://localhost/csj/InsuranceForm.htm



Using BeanUtils to access Request Parameters



HTTP Request Object

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
   Q312461)
Host: localhost
Connection: Keep-Alive
```

HTTP Request Information

- The class `ServletRequest` contains methods that provide information about the current request
  - `int getContentLength()`
    - The length, in bytes, of the request body. Returns -1 if length isn't known.
  - `String getProtocol()`
    - The name and version of the protocol the client uses. In the form: *protocol* **/** *majorVersion* **.** *minorVersion*

    e. g., HTTP/ 1.1 .

HTTP Request Information (cont.)

- `String getRemoteAddr()`
  - The IP address of the client.
- `String getRemoteHost()`
  - The fully qualified host name of the client.
- `boolean isSecure()`
  - Whether this request was made using a secure channel, such as https .

HTTP Request Information (cont.)

- In class HttpServletRequest
  - `String getAuthType()`
    - The name of the authentication scheme used, e. g., BASIC or SSL or null
  - `String getContextPath()`
    - The portion of the request URI that indicates the context of the request.
  - `String getMethod()`
    - The name of the HTTP request method e. g., GET, POST, or PUT.

HTTP Request Information (cont.)

- `String getPathInfo()`
  - Any extra path information associated with the URL the client sent.
- `String getPathTranslated()`
  - Any extra path information after the servlet name but before the query string, and translates it to a real path.
- `String getQueryString()`
  - The query string that is appended to the request URL after the path.

HTTP Request Information (cont.)

- `String getRemoteUser()`
  - The login of the user making this request, if the user has been authenticated, or null .
- `String getRequestURL()`
  - The part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.
- `String getServletPath()`
  - The part of this request's URL that calls the servlet.

Request Information Servlet

```
protected void processRequest(HttpServletRequest request, HttpServletResponse
   response)
   throws ServletException, java.io.IOException {
       String [][] requestInfo = {
           {"Content Length", String.valueOf(request.getContentLength()) },
           {"Content Type", request.getContentType() },
           {"Method" , request.getMethod() },
           {"Authorization Type", request.getAuthType() },
           {"Remote User", request.getRemoteUser() },
           {"Remote Address", request.getRemoteAddr() },
           {"Scheme", request.getScheme() },
           {"Is Secure", String.valueOf(request.isSecure()) },
           {"Protocol", request.getProtocol() },
           {"Context Path", request.getContextPath() },
           {"Path Info", request.getPathInfo() },
           {"Path Translated", request.getPathTranslated() },
           {"Query String", request.getQueryString() },
           {"Servlet Path", request.getServletPath() },
           {"Request URI", request.getRequestURI() }
       };
```

Request Headers

- Most of the above Request information is actually stored in a Request Header.
- Each header is one line of the HTTP request, in a **key:value** format.

Common Request Headers

- `User-Agent`
  - Identifies the browser type and version, e. g.,Mozilla/ 4.72 [en] (X11; U; Linux 2.2.14- 5.0 i686)
- `Host`
  - Indicates the host given in the request URL, e. g.,
  - Required in HTTP 1.1
- `Accept`
  - Indicates MIME types browser can handle, e. g.,image/gif, image/jpeg, image/png, */*
- `Accept-Encoding`
  - Indicates encodings browser can handle, e. g., gzip or compress

Common Request Headers (cont.)

- `Connection`
  - `keep-alive` : browser can handle persistent connection.
- `Authorization`
  - User identification for password- protected pages.
- `Cookie`
  - Cookies previously sent to the client by the same server.
- `If-Modified-Since`
  - Send the page only if it has been changed after specified date.
- `Referer`
  - URL of the referring Web page.

Request Headers API

- `String getHeader( String name)`
  - Returns the value of the specified request header as a `String`.
- `Enumeration getHeaderNames()`
  - Returns an enumeration of all the header names this request contains.
- `Enumeration getHeaders( String name)`
  - Returns all the values of the specified request header as an `Enumeration` of `String` objects.

Retrieving Request Headers

```
Enumeration headers = request.getHeaderNames();
        while(headers.hasMoreElements()) {
            String headerName = (String)headers.nextElement();
            String value = request.getHeader(headerName);
            …
        }
```