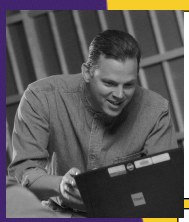
Servlet and JSP Review



Agenda

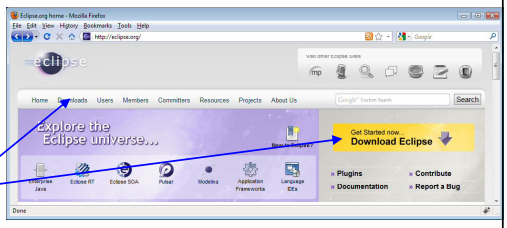
- Eclipse and Tomcat setup
- Deploying apps from Eclipse to Tomcat
- Making new apps in Eclipse
- Servlet basics
- Creating forms and reading form data
- JSP scripting
- Using XML syntax for JSP pages
- JSP file inclusion
- MVC

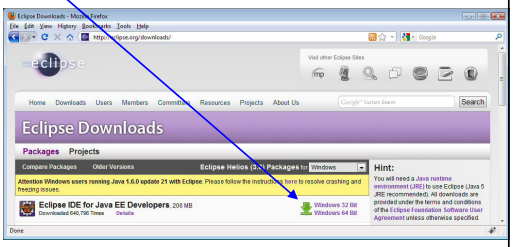
Installing Eclipse

For even more detailed step-by-step instructions, see tutorials on using Eclipse with Tomcat 6 or Tomcat 7 at http://www.coreservlets.com/Apache-Tomcat-Tutorial/

Installing Eclipse

- **Overview**
  - Eclipse is a a free open source IDE for Java. Support for Java, HTML, CSS, JavaScript, C++, PHP, and more.
    - http://eclipse.org/downloads/
    - Choose "Eclipse IDE for Java EE Developers"
      - Need version 3.6 (Helios) for Tomcat 7



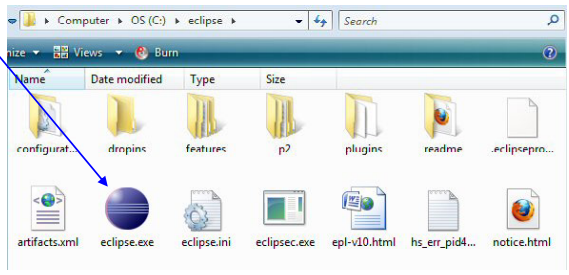


- **Features**
  - Checks your syntax as you type
  - Automatically compiles every time you save file
  - Many tools: refactoring, debugging, server integration, templates for common tasks, etc.
    - Low learning curve: beginners can use Eclipse without knowing these tools

Note: step-by-step Eclipse/Tomcat integration guide at http://www.coreservlets.com/ (click "Apache Tomcat 7" in top left).

Running Eclipse

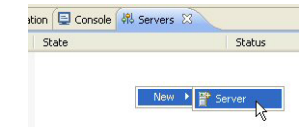
- **Unzip the downloaded file**
  - Call the folder you unzip into "installDir"
- **Double click eclipse.exe**
  - From *installDir*/bin
- **Click on "Workbench" icon**
  - Next time you bring up Eclipse, it will come up in workbench automatically

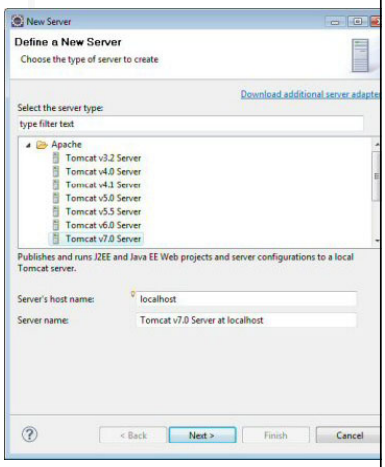


- **Shortcut**
  - Many developers put Eclipse link on their desktop
    - R-click eclipse.exe, Copy, then go to desktop, R-click, and Paste Shortcut (not just Paste!)

Configuring Eclipse

- **Tell Eclipse about Java version**
  - Window → Preferences → Java → Installed JREs → Press "Add", choose "Standard VM", navigate to JDK folder (not "bin" subdirectory)
    - E.g., C:\Program Files\Java\jdk1.6.0_21
- **Tell Eclipse about Tomcat**
  - Click on Servers tab at bottom. R-click in window.
  - New, Server, Apache, Tomcat v7.0, Next, navigate to folder, Finish.
- **Suppress serializable warnings**
  - Window → Preferences → Java → Compiler → Errors/Warnings
    - Change "Serializable class without ..." to "Ignore"



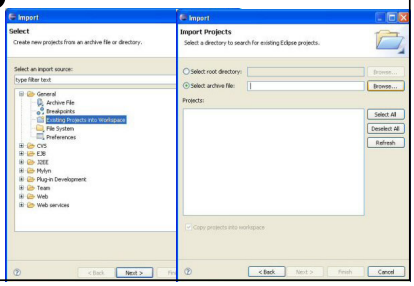


Tomcat v7.0 is choice only in Eclipse 3.6 (Helios). If you prefer Tomcat 6, choose Tomcat v6.0 above instead. If you lose the "Servers" tab at the bottom of Eclipse, use Window, Show View, and hunt for "Servers".

Deploying Apps from Eclipse

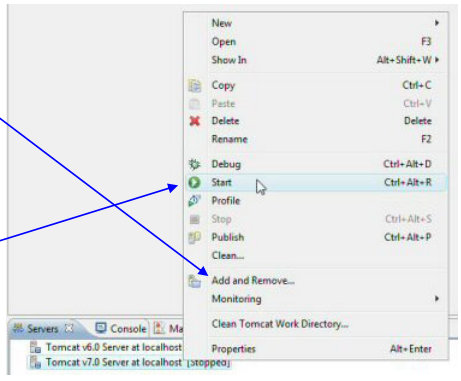
Download and Import Sample Project

- **Get test-app.zip from coreservlets.com**
  - Start at Ajax tutorials
    - http://courses.coreservlets.com/Course-Materials/ajax.html
      - Go to first section (Servlet and JSP Review)
  - Or, start at Apache Tomcat tutorial
    - http://www.coreservlets.com/Apache-Tomcat-Tutorial/
      - Choose Tomcat 7 (recommended) or Tomcat 6 version
- **Then, download test-app.zip**
  - Then, import into Eclipse.
    - File, Import, General, Existing Projects, Select archive file. Then click Browse and navigate to test-app.zip.



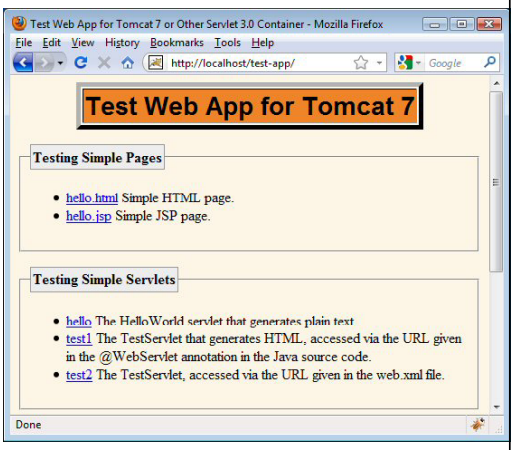
Deploying App in Eclipse

- **Deploy project**
  - Select "Servers" tab at bottom
  - R-click on Tomcat
  - Choose "Add and Remove"
  - Choose project
  - Press "Add"
  - Click "Finish"
- **Start Server**
  - R-click Tomcat at bottom
  - Start (use "Restart" if Tomcat already running)
- **Test URL**
  - http://localhost/test-app/ in any Web browser



Testing Deployed App in Eclipse

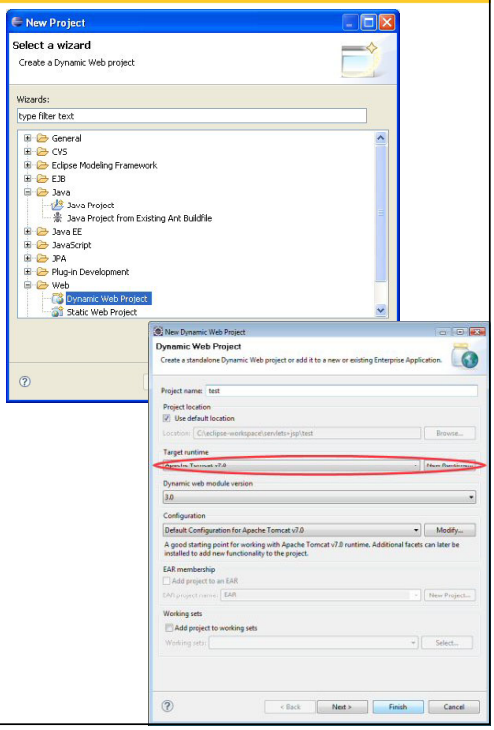
- **Start a browser**
  - Eclipse also has builtin browser, but I prefer to use Firefox or Internet Explorer
- **Test base URL**
  - http://localhost/test-app/
- **Test Web content**
  - http://localhost/test-app/hello.html
  - http://localhost/test-app/hello.jsp
- **Test servlets**
  - http://localhost/test-app/hello
  - http://localhost/test-app/test1
  - http://localhost/test-app/test2



Making New Apps from Eclipse

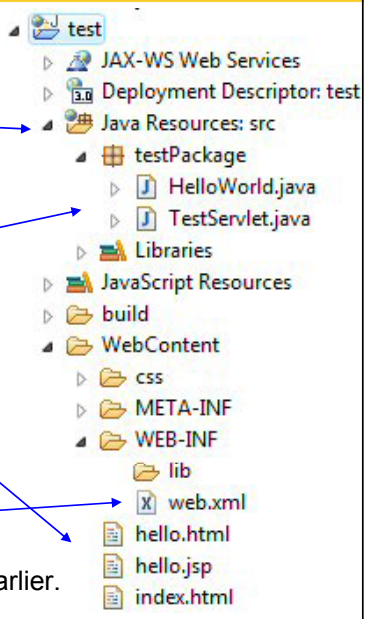
Making Web Apps in Eclipse

- **Make empty project**
  - File → New → Project → Web → Dynamic Web Project
  - For "Target runtime", choose "Apache Tomcat v7.0"
  - Give it a name (e.g., "test")
  - Accept all other defaults
- **Shortcut**
  - If you have made Dynamic Web Project recently in workspace, you can just do File → New → Dynamic Web Project



Adding Code to Eclipse Projects

- **Locations**
  - Java Resources: src
    - R-click and New → Package
    - Never use default package
  - src/testPackage
    - Java code in testPackage package
  - WebContent
    - Web files (HTML, JavaScript, CSS, JSP, images, etc.)
  - WebContent/some-subdirectory
    - Web content in subdirectory
    - R-click on WebContent, New → Folder
  - WebContent/WEB-INF
    - web.xml
      - Optional with servlets 3.0. Required in 2.5 & earlier.
      - Will be discussed later
- **Note**
  - Can cut/paste or drag/drop files into appropriate locations



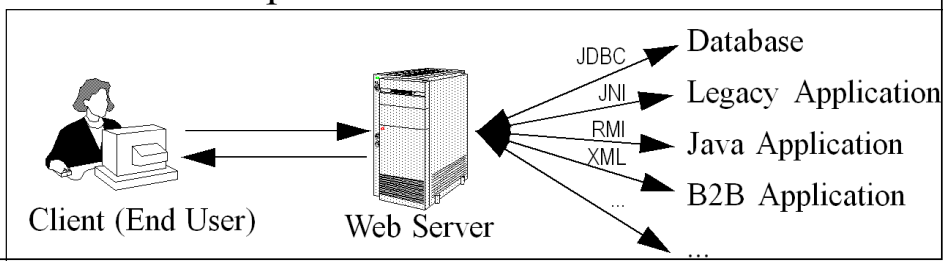
Testing New App

- **Follow same procedure as "deploying app" from previous section**
  - Deploy project
    - Select "Servers" tab at bottom
    - R-click on Tomcat
    - Choose "Add and Remove"
    - Choose project
    - Press "Add"
    - Click "Finish"
  - Start Server
    - R-click Tomcat at bottom
    - Restart (use "Start" if Tomcat not already running)
  - Test URL
    - http://localhost/*appName*/ in any Web browser

Servlet Basics

A Servlet's Job

- **Read explicit data sent by client**
  - Form data
- **Read implicit data sent by client**
  - Request headers
- **Generate the results**
- **Send the explicit data back to client**
  - HTML or XML or JSON or custom data format
- **Send the implicit data to client**
  - Status codes and response headers



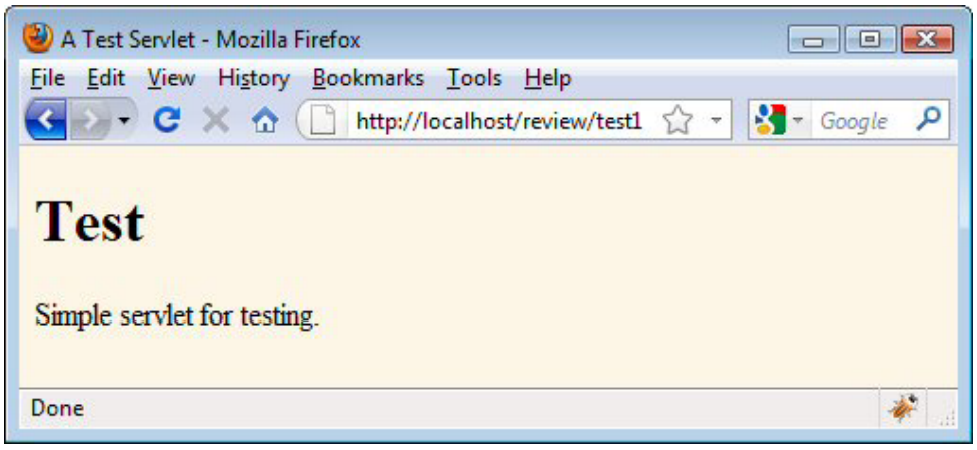
Accessing the Online Documentation

- **Servlets and JSP**
  - http://docs.coreservlets.com/servlet-3.0-api/
    - Servlets 3.0 and JSP 2.2 (Tomcat 7)
  - http://java.sun.com/products/servlet/2.5/docs/servlet-2_5-mr2/
    - Servlets 2.5 (Tomcat 6)
  - http://java.sun.com/products/jsp/2.1/docs/jsp-2_1-pfd2/
    - JSP 2.1 (Tomcat 6)
- **Java 6**
  - http://java.sun.com/javase/6/docs/api/
    - Class uses Java 6 and Tomcat 7
- **Advice**
  - If you have a fast and reliable internet connection, bookmark these addresses
  - If not, download a copy of the APIs onto your local machine and use it

A Sample Servlet (Code)

```
@WebServlet("/test1")
public class TestServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println
      ("<!DOCTYPE html>\n" +
       "<html>\n" +
       "<head><title>A Test Servlet</title></head>\n" +
       "<body bgcolor=\"#fdf5e6\">\n" +
       "<h1>Test</h1>\n" +
       "<p>Simple servlet for testing.</p>\n" +
       "</body></html>");
  }
}
```


A Sample Servlet (Result)



Screenshot assumes project is named "review". Code for this app can be downloaded from the tutorial Web site.

Eclipse users can use the TestServlet code as a basis for their own servlets.
Avoid using "New → Servlet" in Eclipse since it results in ugly code.

Debugging Servlets

- **Use print statements; run server on desktop**
- **Use Apache Log4J**
- **Integrated debugger in IDE**
  - Right-click in left margin in source to set breakpoint (Eclipse)
  - R-click Tomcat and use "Debug" instead of "Start"
- **Look at the HTML source**
- **Return error pages to the client**
  - Plan ahead for missing or malformed data
- **Use the log file**
  - log("message") or log("message", Throwable)
- **Separate the request and response data .**
  - Request: see EchoServer at www.coreservlets.com
  - Response: see WebClient at www.coreservlets.com
- **Make sure browser is not caching**
  - Internet Explorer: use Shift-RELOAD
  - Firefox: use Control-RELOAD
- **Stop and restart the server**

Giving URLs to Servlets

Tomcat 7 or Other Servlet 3.0 Containers

- **Give address with @WebServlet**

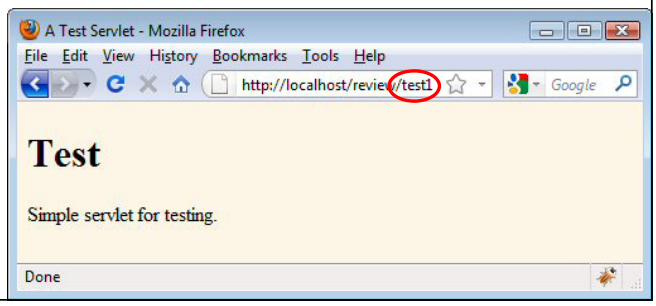
```
@WebServlet("/my-address")
public class MyServlet extends HttpServlet { … }
```

  - Resulting URL
    - http://*hostName*/*appName*/my-address

- **Omit web.xml entirely**
  - You are permitted to use web.xml even when using @WebServlet, but the entire file is completely optional.
    - In earlier versions, you must have a web.xml file even if there were no tags other than the main start and end tags (<web-app …> and </web-app>).

Example: URLs with @WebServlet

```
package coreservlets;
…
@WebServlet("/test1")
public class TestServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println
      ("<!DOCTYPE html>\n" +
       …);
  }
}
```



Defining Custom URLs in web.xml (Servlets 2.5 & Earlier)

- **Java code**

```
package myPackage;  ...
public class MyServlet extends HttpServlet { ... }
```

- **web.xml entry (in <web-app...>...</web-app>)**
  - Give name to servlet

```
<servlet>
  <servlet-name>MyName</servlet-name>
  <servlet-class>myPackage.MyServlet</servlet-class>
</servlet>
```

  - Give address (URL mapping) to servlet

```
<servlet-mapping>
  <servlet-name>MyName</servlet-name>
  <url-pattern>/my-address</url-pattern>
</servlet-mapping>
```

- **Resultant URL**
  - http://*hostname*/*appName*/my-address

Defining Custom URLs: Example

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app  version="2.4"
    ... >

  <!-- Use the URL http://hostName/appName/test2 for
       testPackage.TestServlet -->
  <servlet>
    <servlet-name>Test</servlet-name>
    <servlet-class>coreservlets.TestServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Test</servlet-name>
    <url-pattern>/test2</url-pattern>
  </servlet-mapping>
</web-app>
```

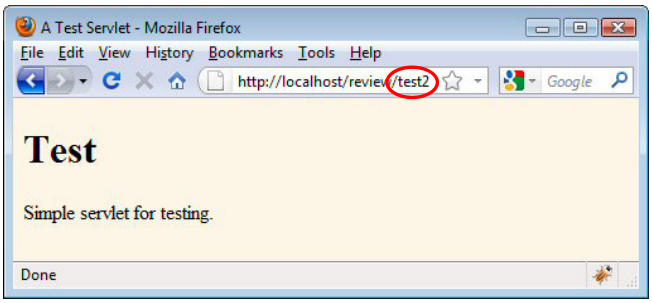
Don't edit this manually. Should match version supported by your server. If your server supports 3.0, can omit web.xml totally and use annotations.

Fully qualified classname.

Any arbitrary name. But must be the same both times.

The part of the URL that comes after the app (project) name. Should start with a slash.

Defining Custom URLs: Result



- **Eclipse details**
  - Name of Eclipse project is "review"
  - Servlet is in src/coreservlets/TestServlet.java
  - Deployed by right-clicking on Tomcat, Add and Remove Projects, Add, choosing review project, Finish, right-clicking again, Start (or Restart)

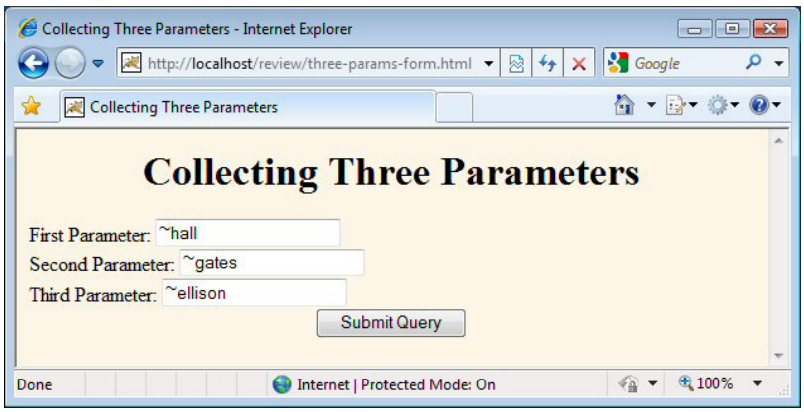
Form Data

Using Form Data

- **HTML form**
  - Should have ACTION referring to servlet
    - Use relative URL
      - ACTION="/webAppName/address"
      - ACTION="./address"
  - Should have input entries with "name" attributes
  - Should be installed under WebContent
- **Servlet**
  - Calls request.getParameter with name as given in HTML
  - Return value is entry as entered by end user
  - Missing values
    - null if no input element of that name was in form
    - Empty string if form submitted with empty textfield

An HTML Form With Three Parameters

```
<FORM ACTION="three-params">
  First Parameter:  <INPUT TYPE="TEXT" NAME="param1"><BR>
  Second Parameter: <INPUT TYPE="TEXT" NAME="param2"><BR>
  Third Parameter:  <INPUT TYPE="TEXT" NAME="param3"><BR>
  <CENTER><INPUT TYPE="SUBMIT"></CENTER>
</FORM>
```

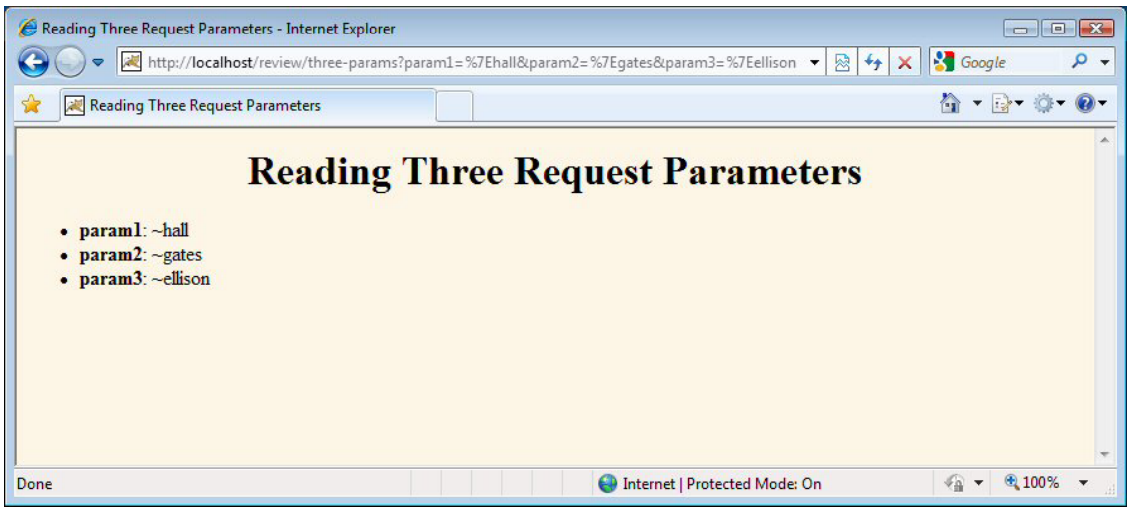


- Project name is "review"
- Form installed in WebContent/three-params-form.html

Reading the Three Parameters

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


Reading Three Parameters: Result



JSP Scripting

Uses of JSP Constructs

Simple Application

Complex Application

- **Scripting elements calling servlet code directly**
- **Scripting elements calling servlet code indirectly (by means of utility classes)**
- **Beans**
- **Servlet/JSP combo (MVC)**
- **MVC with JSP expression language**
- **Custom tags**
- **MVC with beans, custom tags, and a framework like Struts or JSF**

JSP Scripting Design Strategy: Limit Java Code in JSP Pages

- **You have two options**
  - Put 25 lines of Java code directly in the JSP page
  - Put those 25 lines in a separate Java class and put 1 line in the JSP page that invokes it
- **Why is the second option *much* better?**
  - **Development**. You write the separate class in a Java environment (editor or IDE), not an HTML environment
  - **Debugging**. If you have syntax errors, you see them immediately at compile time. Simple print statements can be seen.
  - **Testing**. You can write a test routine with a loop that does 10,000 tests and reapply it after each change.
  - **Reuse**. You can use the same class from multiple pages.

JSP Expressions

- **Format**
  - <%= Java Expression %>
- **Result**
  - Expression evaluated, converted to String, and placed into HTML page at the place it occurred in JSP page
  - That is, expression placed in _jspService inside out.print
- **Examples**
  - Current time: <%= new java.util.Date() %>
  - Your hostname: <%= request.getRemoteHost() %>
- **XML-compatible syntax**
  - <jsp:expression>Java Expression</jsp:expression>
  - You cannot mix versions within a single page. You must use XML for *entire* page if you use jsp:expression.

Predefined Variables

- **request**
  - The HttpServletRequest (1st argument to service/doGet)
- **response**
  - The HttpServletResponse (2nd arg to service/doGet)
- **out**
  - The Writer (a buffered version of type JspWriter) used to send output to the client
- **session**
  - The HttpSession associated with the request (unless disabled with the session attribute of the page directive)
- **application**
  - The ServletContext (for sharing data) as obtained via getServletContext().

JSP Scriptlets

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

JSP Declarations

- **Format**
  - <%! Java Code %>
- **Result**
  - Code is inserted verbatim into servlet's class definition, outside of any existing methods
- **Examples**
  - <%! private int someField = 5; %>
  - <%! private void someMethod(...) {...} %>
- **Design consideration**
  - Fields are clearly useful. For methods, it is usually better to define the method in a separate Java class.
- **XML-compatible syntax**
  - <jsp:declaration>Java Code</jsp:declaration>

JSP Pages with XML Syntax

Why Two Versions?

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

XML Syntax for Generating XHTML Files (somefile.jspx)

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

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

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

Builds DOCTYPE line.

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

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

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

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

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

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

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

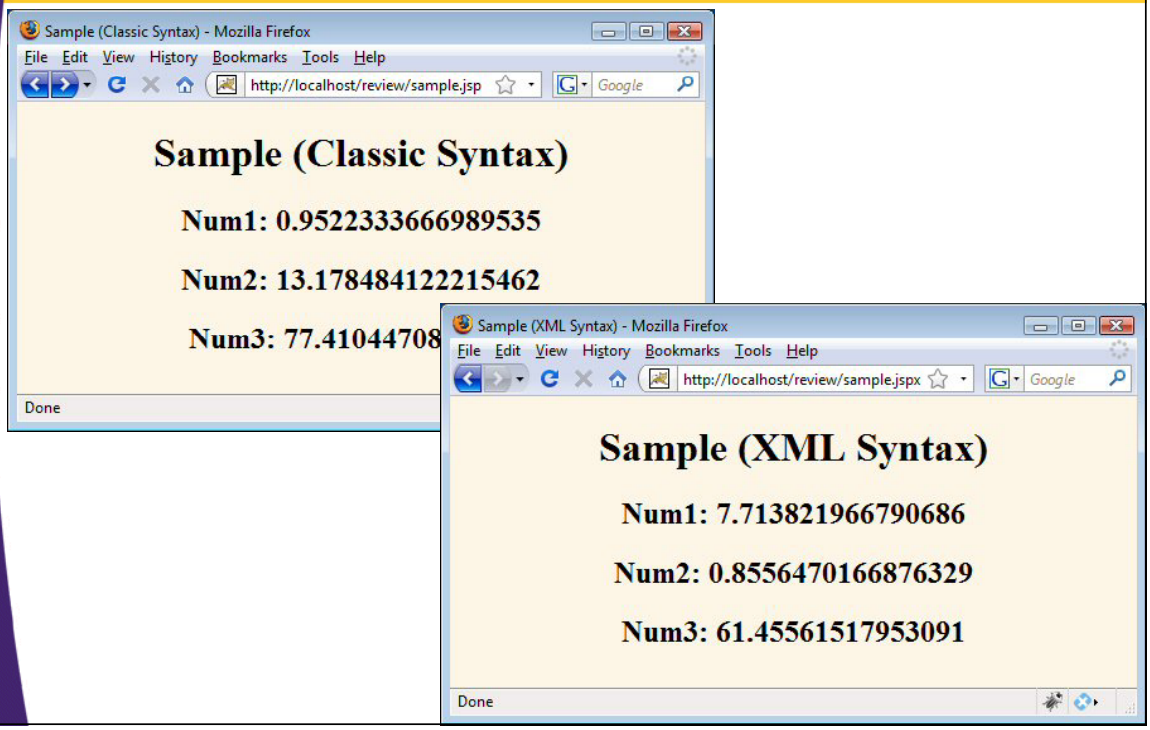
Sample HTML 4 Page: Classic Syntax (sample.jsp)

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

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

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


Sample XHTML Page: XML Syntax (sample.jspx)

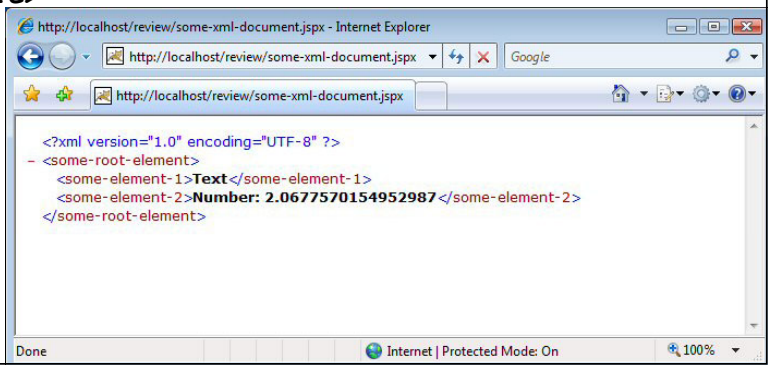
```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns:jsp="http://java.sun.com/JSP/Page">
<jsp:output
     omit-xml-declaration="true"
     doctype-root-element="html"
     doctype-public="-//W3C//DTD ..."
     doctype-system="http://www.w3.org...dtd" />
<jsp:directive.page contentType="text/html"/>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

Sample Pages: Results



XML Document Generated with XML Syntax

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



jsp:include

Including Files at Request Time: jsp:include

- **Format**
  - <jsp:include page="Relative URL" />
- **Purpose**
  - To reuse JSP, HTML, or plain text content
  - To permit updates to the included content without changing the main JSP page(s)
- **Notes**
  - JSP content cannot affect main page: only *output* of included JSP page is used
  - Don't forget that trailing slash
  - Relative URLs that starts with slashes are interpreted relative to the Web app, not relative to the server root.
  - You are permitted to include files from WEB-INF

jsp:include Example: A News Headline Page (Main Page)

```
…
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      What's New at JspNews.com</TABLE>
<P>
Here is a summary of our three
most recent news stories:
<OL>
  <LI><jsp:include page="/WEB-INF/includes/item1.jsp" />
  <LI><jsp:include page="/WEB-INF/includes/item2.jsp" />
  <LI><jsp:include page="/WEB-INF/includes/item3.jsp" />
</OL>
</BODY></HTML>
```


A News Headline Page, Continued (First Included Page)

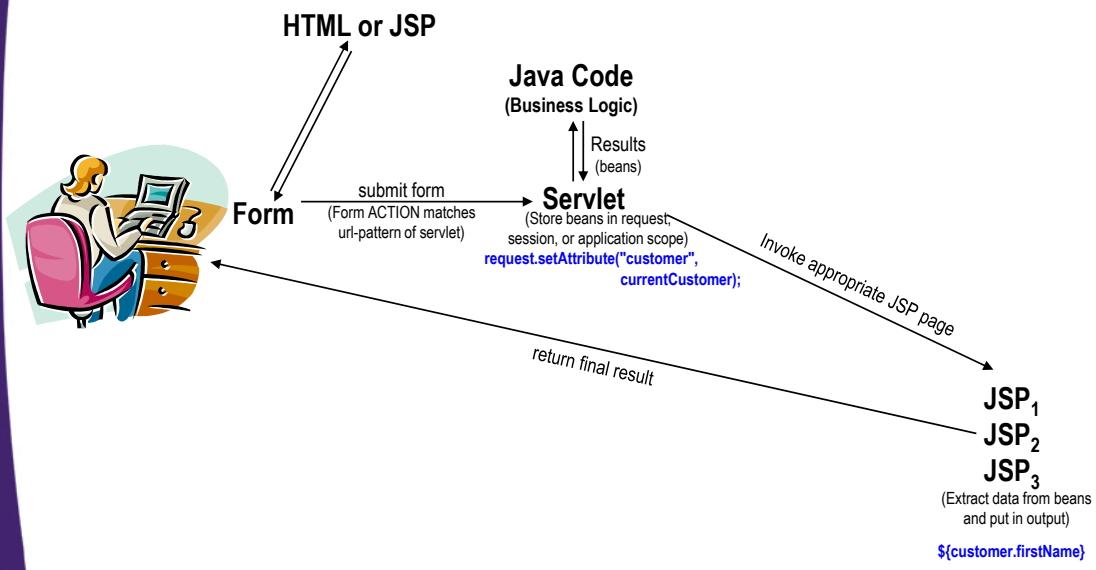
```
<B>Bill Gates acts humble.</B> In a startling
and unexpected development, Microsoft big wig
Bill Gates put on an open act of humility
yesterday.
<A HREF="http://www.microsoft.com/Never.html">
More details...</A>
```

- Note that the page is *not* a complete HTML document; it has only the tags appropriate to the place that it will be inserted.
  - This style of having servlets or JSP pages build only small pieces of HTML (or other data types) is even more widely used in Ajax programming

A News Headline Page: Result

MVC

MVC Flow of Control



Simple MVC Example: Request-Scoped Data

- **Goal**
  - Display a random number to the user
- **Type of sharing**
  - Each request should result in a new number, so request-based sharing is appropriate.

Request-Based Sharing: Bean

```
package coreservlets;

public class NumberBean {
  private final double num;

  public NumberBean(double number) {
    this.num = number;
  }

  public double getNumber() {
    return(num);
  }
}
```

The property name in JSP will be "number". The property name is derived from the method name, not from the instance variable name. Also note the lack of a corresponding setter.

Request-Based Sharing: Servlet

```
@WebServlet("/random-number")
public class RandomNumberServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    NumberBean bean =
      RanUtils.randomNum(request.getParameter("range"));
    request.setAttribute("randomNum", bean);
    String address = "/WEB-INF/mvc-sharing/RandomNum.jsp";
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
  }
}
```

Request-Based Sharing: Business Logic

```
public class RanUtils {
  public static NumberBean randomNum(String rangeString) {
    double range;
    try {
      range = Double.parseDouble(rangeString);
    } catch(Exception e) {
      range = 10.0;
    }
    return(new NumberBean(Math.random() * range));
  }

  private RanUtils() {} // Uninstantiable class
}
```


Request-Based Sharing: URL Pattern (web.xml)

```
...
 <servlet>
    <servlet-name>RandomNumberServlet</servlet-name>
    <servlet-class>
      coreservlets.RandomNumberServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>RandomNumberServlet</servlet-name>
    <url-pattern>/random-number</url-pattern>
  </servlet-mapping>
...
```

The web.xml file is not needed with servlets 3.0, and the downloadable "review" project does not have this file. However, for those who are using containers that support only servlets 2.5 or 2.4, a "review2" app is also online. That app uses web.xml instead of @WebServlet for all of the URL patterns.

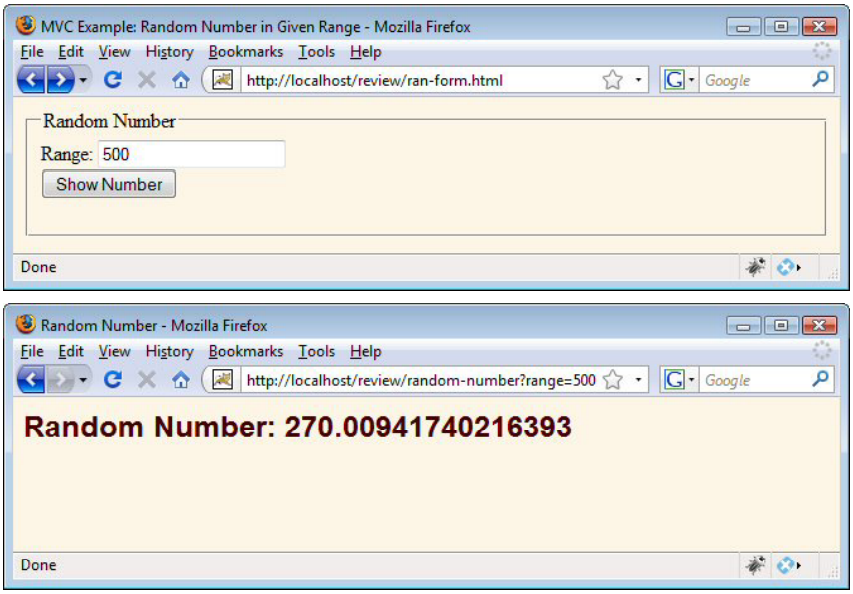
Request-Based Sharing: Input Form

```
...
<fieldset>
  <legend>Random Number</legend>
  <form action="./random-number">
    Range:  <input type="text" name="range"><br/>
    <input type="submit" value="Show Number">
  </form>
</fieldset>
...
```


Request-Based Sharing: Results Page

```
...
<body>
<h2>Random Number: ${randomNum.number}</h2>
</body></html>
```

Request-Based Sharing: Results



Summary

- **Set up Java 6, Tomcat, and Eclipse**
  - See http://www.coreservlets.com/Apache-Tomcat-Tutorial/
- **Give custom URLs to all servlets**
  - Servlets 3.0
    - Use @WebServlet annotation
  - Servlets 2.5 and 2.4
    - Use servlet, servlet-mapping, and url-pattern in web.xml
- **Forms**
  - Use relative URLs for "action".
  - Read parameters with request.getParameter
- **JSP Scripting**
  - If you use scripting, put most Java code in regular classes
- **MVC**
  - Very widely applicable approach.
  - Consider using it in many (most?) applications

Questions?