Introduction to Web Engineering SENG2050/6050

Lecture 3a
JSP

Lecture 3a: JSP

- ➢Good Design - Web Engineering
- ➢Java Server Pages (JSP)
- ➢Scripting Elements
  - ✓Expressions
  - ✓Scriptlets
  - ✓Declarations

Good Design - Web Engineering

Separate the user interface (HTML+CSS) from the "business" logic (XML+Java)

- ✓ Allows you develop each part independently – faster development.
- ✓ Allows graphic designers work on the interface while software engineers work on the logic – better end product.
- ✓ Allows you completely redesign the "look and feel" without changing the business logic – easier maintenance.

Good Design - Web Engineering

Good practices:

- ➢The HTML/CSS separation is a simple example – but HTML is not dynamic.
- ➢Servlets are okay, but you have to "code" the HTML with out.println() – not really a separation.
- ➢A solution: place special tags in the HTML which "access" business logic written in Java.

This is supported using Java Server Pages (JSP)

Java Server Pages – JSP

- ➢In JSPs, the “special” tags contain fragments of Java code
  - ✓The Web server parses the HTML, finds these tags, then compiles and runs the Java code.
- ➢JSPs are built on top of Java Servlets
  - ✓JSPs are compiled into servlets by the Web server.
  - ✓The fragments of Java code have access to the same context, request and response objects as a HttpServlet.

Clarification - Review

- ➢JavaScript
  - ✓Coding in a java-like language in HTML.
  - ✓Generates dynamic HTML on the client.
- ➢Java Servlets
  - ✓Java code with HTML within.
- ➢JSP
  - ✓Generates dynamic Runs on the server.
  - ✓HTML with Java code within.
  - ✓Can be translated into Servlets.

Clarification - Review

- ➢JavaScript vs. JSP
  - ✓They do not overlap.
  - ✓They complement as JSP can generate JavaScript that will be sent to the client.
- ➢Servlets vs. JSP
  - ✓They do not overlap.
  - ✓Highly variable content => Servlets.
  - ✓Large HTML sections + some coding => JSP.

Java Server Pages – JSP

- ➢To avoid the overhead of continually compiling, JSPs are cached by the Web server in a work area
  - ✓They are only recompiled if you change the JSP file.
- ➢To avoid even more compiling
  - ✓Place business logic in separate Java objects that are precompiled and live in the Web server.
  - ✓Java Beans are a specific standard for developing this (later).
  - ✓Also helps in separating presentation from logic.

JSP – Basic Constructs

- ➢Normal HTML tags are passed cleanly through when a JSP is processed
  - ✓Actually, they get converted into out.write().
  - ✓This static HTML is called the template text.
- ➢JSP code compiled and run is identified by
  - ✓`<%` to start a JSP section.
  - ✓`%>` to end JSP section.
  - ✓If you want `%` in the HTML, use `\%`.
  - ✓JSP comments `<%--` … `--%>` are not passed through to the client.

JSP – Scripting Elements: Expressions

1. Expressions

- ✓ **<%=***Java Expression* **%>**
- ✓ **<jsp:expression>** *Java Expression* **</jsp:expression>**
- ✓ They are evaluated and inserted into the servlet's output

```
<h1> A Random Number between 0-10</h1>
<%= Math.random() * 10 %>
```

- ✓ Execute the code and pass its output to the client
- ✓ Just shorthand for

`<% out.println(`*Java expr*`); %>`

JSP – Scripting Elements: Expressions

```
<h2>JSP Expressions</h2>
<ul>
 <li>Current time:
  <%= new java.util.Date() %> </li>
 <li>Your hostname:
  <%= request.getRemoteHost() %> </li>
 <li>Your session ID:
  <%= session.getId() %> </li>
 <li>Your Parameter is:
   <%= request.getParameter("testParam") %> </li>
</ul>
```

Request and session are predefined objects

➢Parameter passing: Expressions.jsp?Param=Colombia

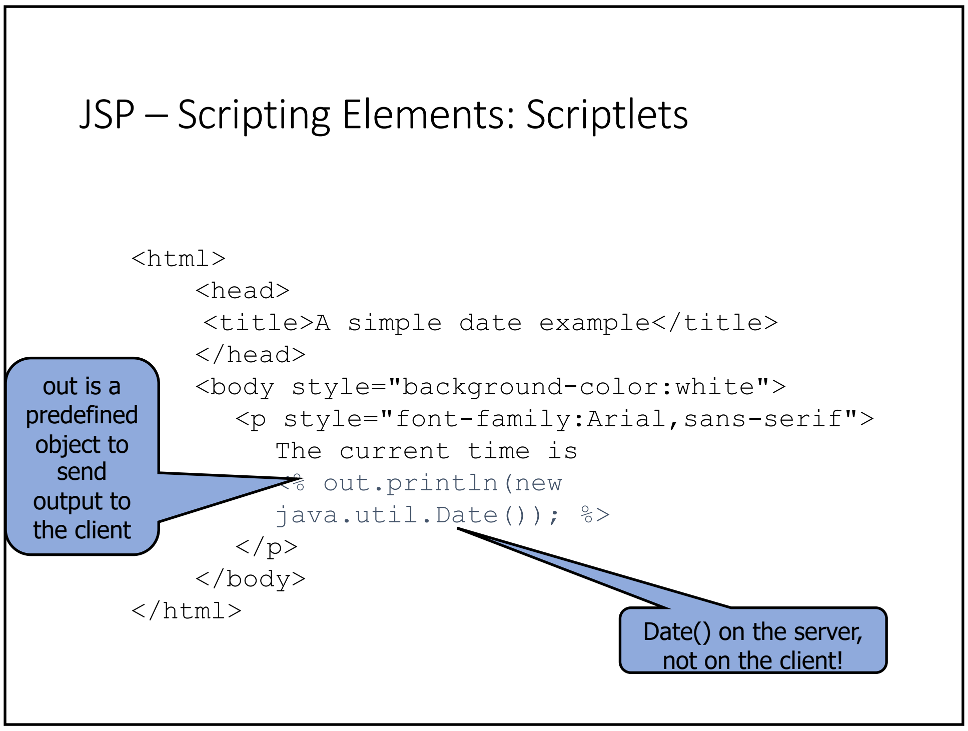
JSP – Scripting Elements: Scriptlets

2. **Scriptlets**

- ✓ `<%` *Java code* `%>`
- ✓ **`<jsp:scriptlet>`** *Java code* **`</jsp:scriptlet>`**

➢Java code that is executed when the request is processed

- ✓ The code is pasted into the `_jspService` method of the resulting servlet, in between any out.write()s for the HTML code
  - It might not produce output – e.g., it could use JDBC to update a database with some <form> input

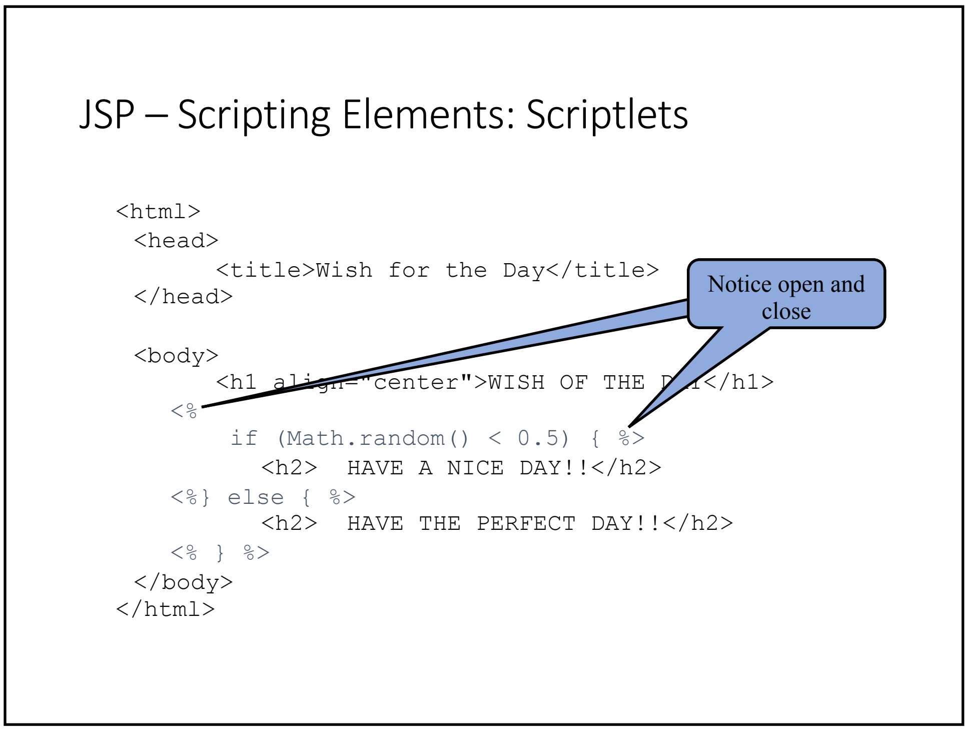


JSP – Scripting Elements: Scriptlets

```
<html>
    <head>
     <title>A simple date example</title>
    </head>
    <body style="background-color:white">
       <p style="font-family:Arial,sans-serif">
          The current time is
          <% out.println(new
          java.util.Date()); %>
       </p>
    </body>
</html>
```

out is a predefined object to send output to the client

Date() on the server, not on the client!



JSP – Scripting Elements: Scriptlets

```
<html>
 <head>
      <title>Wish for the Day</title>
 </head>

 <body>
      <h1 align="center">WISH OF THE DAY</h1>
    <%
        if (Math.random() < 0.5) { %>
          <h2>  HAVE A NICE DAY!!</h2>
    <%} else { %>
          <h2>  HAVE THE PERFECT DAY!!</h2>
    <% } %>
 </body>
</html>
```

Notice open and close

JSP – Scripting Elements: Declarations

3. Declarations

- ✓ `<%! Field or Method Definition %>`
- ✓ `<jsp:declaration>`

  Java Declaration

  `</jsp:declaration>`
- ✓ The code is pasted into the Java Servlet at top level
  - Allows the addition of new methods, variables, and even subclasses to the servlet
  - Does not produce output to the client

JSP – Scripting Elements: Declarations

Declaration

```
<%! private int accessCount = 0; %>
<h2>Accesses to page since server reboot:
            <%= ++accessCount %></h2>


<%! public java.util.Date PrintDate()
    {
      return (new java.util.Date());
    }
%>
…
<p style="font-family:Arial,sans-serif">
  The current time is <%= PrintDate() %>
</p>
```

Expression

JSP – Implicit Objects

- Objects that exists for use in a JSP
  - ✓ `request`, `response`, `out`
  - ✓ `session`
  - ✓ `config`, `application`
  - ✓ `pageContext`, `page`
  - ✓ `exception`
- These have equivalents in `Servlet` and `HttpServlet`

You don't declare them, you just use them

JSP – Implicit Objects

- `request` – represents the request this JSP is serving.
- `response` – represents the response the JSP is generating for the client.
- `out` – a Writer used to generate output for the client
  - ✓ usually only needed in scriptlets
- `session` – represents the session associated with the request
  - ✓ Created automatically

JSP – Implicit Objects

- `application` – the `ServletContext` object
  - ✓ Shared by all the servlets in the servlet engine
  - ✓ `setAtribute` and `getAtribute` methods
- `config` – the `ServletConfig` object
- `pageContext` – the `PageContext` object
  - ✓ Used for sharing Java Beans between servlets
- `page` = `this`

JSP – Implicit Objects – `request`

- `getProtocol()` – HTTP/1.1, FTP, SMTP, …
- `getServerName()` – the name of the computer running the server
- `getPort()` – the port the server is listening to
- `getRemoteAddr()` – the IP number the request came from
- `getRemoteName()` – the IP name the request came from
- `getParameter(`*name*`)` – the value of a parameter passed in the request

JSP – Implicit Objects – `request`

- `getHeader(`*name*`)` – the value of any header passed in the request
- `getMethod()` – `GET` or `POST` (usually)
- `getPathInfo()` – the path portion of the requesting URI
- `getQueryString()` – the query portion of the requesting URI
- `getRemoteUser()` – the name of the user who sent the request (if it can be determined)
- `getRequestURI()` – full URI of the request

JSP – Implicit Objects – `out`

- `print(`*string*`)`, `println(`*string*`)` – the standard PrintWriter methods
  - ✓ Inside the servlet code (that the JSP engine creates), `out` is a `java.io.PrintWriter` – you get one by calling `ServletResponse.getWriter()`
  - ✓ In JSP, `out` is a `java.servlet.jsp.JspWriter` – you get it automatically through the `pageContext`
  - ✓ For all practical purposes they are interchangeable

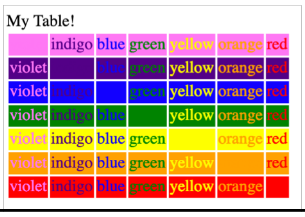
```
Path Info: <%= request.getPathInfo() %>
<% out.print("Path Info: "); %>
<% out.println(request.getPathInfo() ); %>
```

JSP – Examples

```
<p>
  <% java.util.Date now = new
     java.util.Date(); %>
  I think that
  <%= request.getRemoteHost() %>,
  let the dog out at exactly
  <%= now.getHours() %> :
  <%= now.getMinutes() %> :
  <%= now.getSeconds() %> hours.
</p>
```

JSP – Examples

```
<table>
<% int row, col;
   String [] colours = {"violet", "indigo", "blue",
       "green", "yellow", "orange", "red"};
   row = 0;
   while (row < colours.length) { %>
     <tr style="background-color:<%=colours[row] %>">
     <% col = 0;
     while (col < colours.length) { %>
       <td style="color:<%= colours[col] %>">
           <%= colours[col] %></td>
           <% col++;
     } %>
     </tr>
     <% row++;
   } %>
</table>
```


My Table!

	indigo	blue	green	yellow	orange	red
violet		blue	green	yellow	orange	red
violet	indigo		green	yellow	orange	red
violet	indigo	blue		yellow	orange	red
violet	indigo	blue	green		orange	red
violet	indigo	blue	green	yellow		red
violet	indigo	blue	green	yellow	orange	

JSP – Examples

```
<%
  String bg = request.getParameter("bg");
  boolean hasBg;
  if (bg != null) {
   hasBg = true;
  }
  else {
   hasBg = false;
   bg = "white";
  }
 %>
  <body style="background-color: <%= bg %>”
  <p>
   This is a JSP!<br>
   The background color parameter:
          <%= request.getParameter("bg") %>
  </p>
  </body>
```

JSP Resources

- ➢Java Server Pages (JSP)
  - ✓https://www.oracle.com/technetwork/java/javaee/jsp/index.html
  - ✓Training Materials from the textbook
  - ✓http://courses.coreservlets.com/Course-Materials/
- ➢Web
  - ✓http://www.jsptut.com/

THE END

QUESTIONS??

THANKS!!