

CNT 4714: Enterprise Computing Summer 2014

Introduction to JavaServer Pages (JSP) – Part 1

Instructor : Dr. Mark Llewellyn
 markl@cs.ucf.edu
 HEC 236, 407-823-2790
 <http://www.cs.ucf.edu/courses/cnt4714/sum2014>

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida



Introduction to JavaServer Pages (JSP)

- **JavaServer Pages (JSP)** is an extension of servlet technology.
- Like servlets, JSPs simplify the delivery of dynamic web content. They allow web programmers to create dynamic content by reusing predefined components and by interacting with components using server-side scripting.
- JSPs can reuse JavaBeans and create custom tag libraries that encapsulate complex, dynamic functionality.
- JSP classes and interfaces can be found in packages `javax.servlet.jsp` and `javax.servlet.jsp.tagext`.



Introduction to JSP (cont.)

- There are four key components to JSPs
 1. **Directives:** messages to the JSP container (server component executing the JSP) that enable the programmer to specify page settings, include content from other resources and specify custom tag libraries to use in a JSP.
 2. **Actions:** encapsulate functionality based on the information sent to the server as part of a specific client request. They can also create Java objects for use in JSP scriptlets.
 3. **Scripting elements:** enable the programmer to insert Java code that interacts with components in a JSP to perform request processing.
 4. **Tag libraries:** are part of the **tag extension mechanism** that enables programmers to create custom tags. Typically, most useful for web page designers with little knowledge of Java.



Introduction to JSP (cont.)

- In some ways, JSPs look like standard HTML or XML documents.
- JSPs normally include HTML or XML markup. Such markup is known as **fixed-template data** or **fixed-template text**.
- Fixed-template data/text often helps a programmer decide whether to use a servlet or a JSP. Recall that JSPs are most often used when most of the content sent to the client is fixed-template data and little or none of the content is generated dynamically with Java code. Servlets are more commonly used when only a small amount of the content returned to the client is fixed-template data.



Introduction to JSP (cont.)

- When a JSP-enabled server receives the first request for a JSP, the JSP container translates the JSP into a Java servlet that handles the current request as well as all future requests to the JSP.
- Literal text in the JSP becomes string literals in the servlet that represents the translated JSP.
- Any errors that occur in compiling the new servlet result in **translation-time errors**.
- The JSP container places the Java statements that implement the JSP's response in method `_jspService` at translation time.
- If the new servlet compiles properly, the JSP container invokes method `_jspService` to process the request.
- The JSP may respond directly or may invoke other web application components to assist in processing the request. Any errors that occur during request processing are known as **request-time errors**.



Introduction to JSP (cont.)

- Overall, the request-response mechanism and the JSP life-cycle are the same as those of a servlet.
- JSPs can override methods `jspInit` and `jspDestroy` (similar to servlet methods `init` and `destroy`), which the JSP container invokes when initializing and terminating a JSP.
- A JSP programmer defines these methods using JSP declarations which are part of the scripting mechanism.



The First JSP Example

- Our first look at a JSP is with a simple clock JSP which displays the current date and time inserted into a web page using a JSP expression.
- To execute this `clock.jsp` from your own system, as with the servlet examples we've been running – copy the `clock.jsp` file into the `webapps` subdirectory you created for your servlet examples.
 - My Tomcat `webapps` subdirectory is named `CNT4714` and I created a subdirectory named `JSP` in this directory to hold all the JSP examples. From the index page I created – the JSPs can be executed directly, otherwise...type <http://localhost:8080/CNT4714/jsp/clock.jsp> to execute this JSP.



XHTML meta-element sets a refresh interval of 60 seconds

```

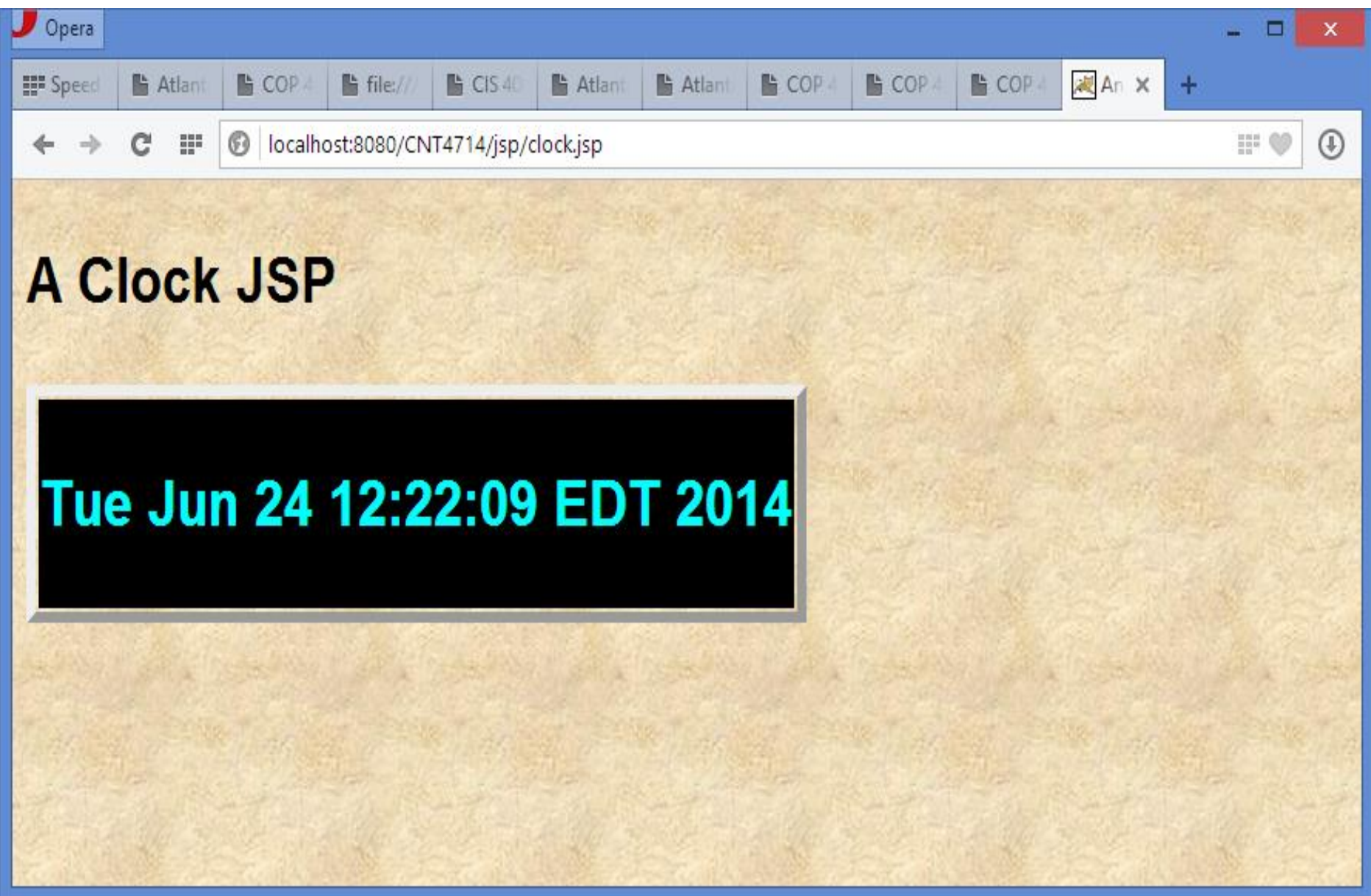
1  <!DOCTYPE html>
2  <!-- A clock.jsp -->
3  <html lang="en">
4  <head>
5      <meta http-equiv = "refresh" content = "60" charset="utf=8" />
6      <title>An Initial JSP Example</title>
7      <style type = "text/css">
8          .big { font-family: helvetica, arial, sans-serif;
9                  font-weight: bold;
10                 font-size: 2em; }
11     </style>
12     <body bgcolor=white background=images/background.jpg lang=EN-US
13         link=blue vlink=blue style='tab-interval:.5in'>
14 </head>
15 <body>
16     <p class = "big">A Clock JSP</p>
17     <table style = "border: 6px outset;">
18         <tr>
19             <td style = "background-color: black;">
20                 <p class = "big" style = "color: cyan;">
21
22                     <!-- JSP expression to insert date/time -->
23                     <%= new java.util.Date() %>
24                 </p>
25             </td>
26         </tr>
27     </table>
28 </body>
29 </html>

```

JSP expressions are delimited by <%= ... %>.

Creates a new instance of class Date (package java.util). When the client requests this JSP, this expression inserts the String representation of the date and time in the response to the client.





Implicit Objects

- Implicit objects provide access to many servlet capabilities in the context of a JSP.
- Implicit objects have four scopes:
 1. **Application:** the JSP container owns objects with application scope. Any JSP can manipulate such objects.
 2. **Page:** objects with page scope can only be manipulated in the page that defines them. Each page has its own instances of the page-scope implicit objects.
 3. **Request:** these objects go out of scope when request processing completes with a response to the client.
 4. **Session:** these objects exist for the client's entire browsing session.



Implicit Objects

Implicit Object	Description
<i>Application Scope</i>	
application	This <code>javax.servlet.ServletContext</code> object represents the container in which the JSP executes.
<i>Page Scope</i>	
config	This <code>javax.servlet.ServletConfig</code> object represents the JSP configuration options. As with servlets, configuration options can be specified in a Web application descriptor.
exception	This <code>java.lang.Throwable</code> object represents the exception that is passed to the JSP error page. This object is available only in a JSP error page.
out	This <code>javax.servlet.jsp.JspWriter</code> object writes text as part of the response to a request. This object is used implicitly with JSP expressions and actions that insert string content in a response.
page	This <code>java.lang.Object</code> object represents the this reference for the current JSP instance.
pageContext	This <code>javax.servlet.jsp.PageContext</code> object hides the implementation details of the underlying servlet and JSP container and provides JSP programmers with Access to the implicit objects listed in this table.



Implicit Objects

Implicit Object	Description
response	This object represents the response to the client. The object normally is an instance of a class that implements HttpServletResponse (package javax.servlet.http). If a protocol other than HTTP is used, this object is an instance of a class that implements javax.servlet.ServletResponse .
<i>Request Scope</i>	
request	This object represents the client request. The object normally is an instance of a class that implements HttpServletRequest (package javax.servlet.http). If a protocol other than HTTP is used, this object is an instance of a subclass of javax.servlet.ServletRequest .
<i>Session Scope</i>	
session	This javax.servlet.http.HttpSession object represents the client session information if such a session has been created. This object is available only in pages that participate in a session.
	.



Scripting

- JSPs often present dynamically generated content as part of an HTML document that is sent to the client in response to a request.
- In some cases, the content is static, but is output only if certain conditions are met during a request (e.g., providing values in a form that submits a request).
- JSP programmers can insert Java code and logic in a JSP using scripting.



Scripting Components

- JSP scripting components include **scriptlets**, **comments**, **expressions**, **declarations**, and **escape sequences**.
- **Scriptlets** are blocks of code delimited by `<%` and `%>`. They contain Java statements that the container places in method `_jspService` at translation time.
- **Comments** come in three flavors in JSPs: JSP comments, XHTML comments, and scripting language comments.
 - JSP comments are delimited by `<%--` and `--%>`. Can be placed throughout the JSP except inside scriptlets.
 - XHTML comments are delimited by `<!--` and `-->`. Can be placed anywhere in the JSP except inside scriptlets.
 - Scripting language comments are Java comments (Java is currently the only JSP scripting language which is allowed). Scriptlets can use either `//` or `/*` and `*/` as in normal Java.



Scripting Components (cont.)

- JSP comments and scripting language comments are ignored and do not appear in the response to a client. When clients view the source code of a JSP response, they will see only the HTML comments in the source code.
 - The different comment styles are useful for separating comments that the user should be able to see from those that document logic processed on the server-side.
- **Expressions** are delimited by `<%=` and `%>` and contain a Java expression that is evaluated when a client requests the JSP containing the expression. The container converts the result of a JSP expression to a `String` object, then outputs the `String` as part of the response to the client.



Scripting Components (cont.)

- **Declarations** are delimited by `<%!` and `%>`. Declarations enable the JSP programmer to define variables and methods for use in a JSP. Variables become instance variables of the servlet class that represents the translated JSP. Similarly, methods become members of the class that represents the translated JSP. Declaration of variables and methods in a JSP use Java syntax such as:

```
<%! int increment = 0; %>
```

- **Escape sequences** are necessary to include special characters or character sequences that the JSP container normally uses to delimit JSP code.
 - Example: literal: `<%`, escape sequence is: `<\%`



Scripting Example – welcome.jsp

```
<!DOCTYPE html>

<!-- welcome.jsp -->
<!-- JSP that processes a "get" request containing data. -->

<html lang="en">

  <!-- head section of document -->
  <head>
    <title>A JSP that processes "get" requests with data</title>
  </head>

  <!-- body section of document -->
  <body>
    <% // begin scriptlet
      String name = request.getParameter( "firstName" );
      if ( name != null )
      {
    %> <%-- end scriptlet to insert fixed template data --%>
```

HTML comments shown
in blue.

Scriptlets shown in green.



```

<h1>
    Hello <%= name %>, <br />
    Welcome to JavaServer Pages Technology!
</h1>

<% // continue scriptlet

    } // end if
    else {

%> <!-- end scriptlet to insert fixed template data --%>

    <form action = "welcome.jsp" method = "get">
        <p>Type your first name and press Submit</p>

        <p><input type = "text" name = "firstName" />
            <input type = "submit" value = "Submit" />
        </p>
    </form>

    <% // continue scriptlet
        } // end else
    %> <!-- end scriptlet --%>
</body>

</html> <!-- end HTML document -->

```





```

17 <body style='tab-interval:.5in'>
18 <font size = 5><br>
19
20 <% // begin scriptlet
21 String name = request.getParameter( "firstName" );
22 if ( name != null )
23 {
24 <%-- end scriptlet to insert fixed template data --%>
25 <h1>
26     Hello <%= name %>, <br />
27     Welcome to JavaServer Pages Technology!
28 </h1>
29 <% // continue scriptlet
30 } // end if
31 else {
32 <%-- end scriptlet to insert fixed template data --%>
33 <form action = "welcome.jsp" method = "get">
34     <p>Type your first name and press Submit</p>
35     <p><input type = "text" name = "firstName" />
36         <input type = "submit" value = "Submit" />
37     </p>
38 </form>
39
40 <% // continue scriptlet
41 } // end else
42 <%-- end scriptlet --%>
43 </body>
44
45 </html> <!-- end XHTML document -->

```

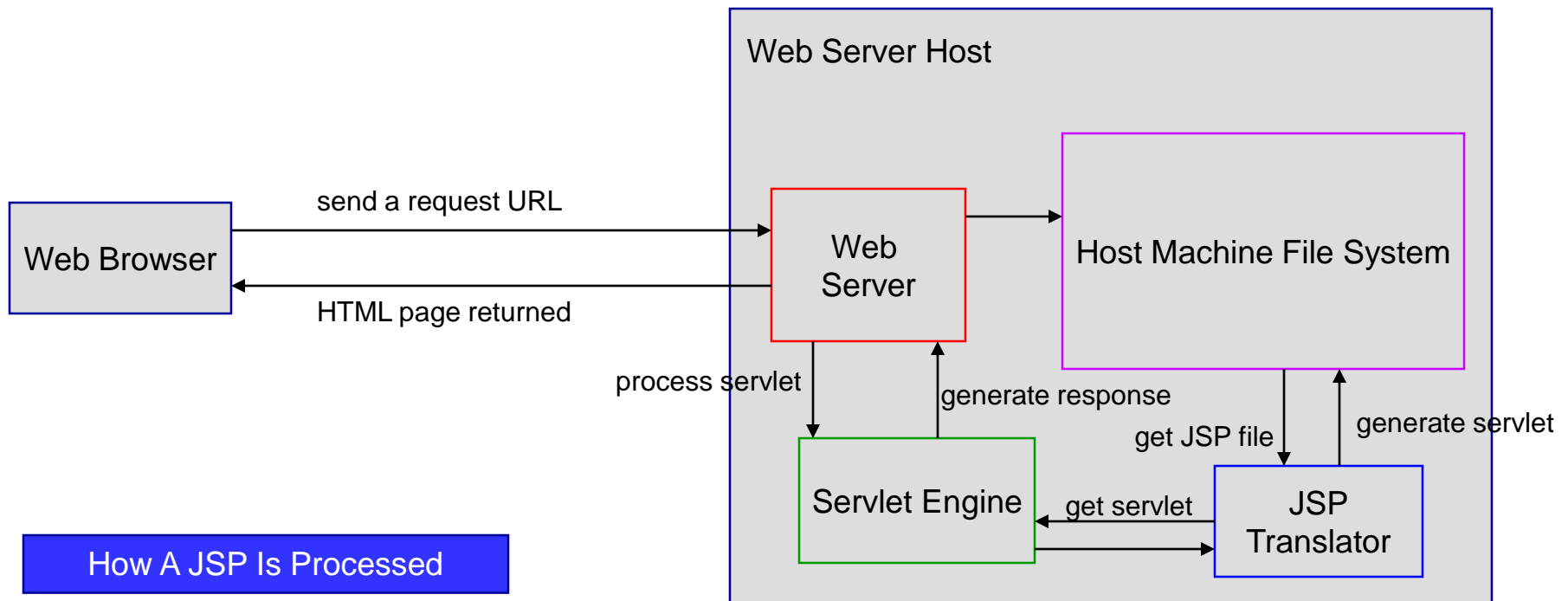
Editors like Notepad++ will delineate your scriptlets and allow for easy differentiation of the various scripting elements that make up more complex JSP files.





How A JSP Is Processed

- Much like a servlet, a JSP must first be processed by a web server before it can be displayed in a web browser. The web server must support JSPs and the JSP page must be stored in a file with a `.jsp` extension. The web server translates the JSP into a Java servlet, compiles the servlet, and executes it. The result of the execution is sent to the browser for display.



More On JSP Scripting Constructs

- There are three main types of JSP constructs: **scripting constructs**, **directives**, and **actions**.
- **Scripting elements** allow you to specify Java code that will become part of the resultant servlet.
- **Directives** enable you to control the overall structure of the resultant servlet.
- **Actions** enable you to control the behavior of the JSP engine.
- We'll look in more detail at all of these, starting with the scripting constructs.



Scripting Constructs

- There are three main types of JSP scripting constructs that can be used to insert Java code into a resultant servlet: **expressions**, **scriptlets** and **declarations**. Recall that there are also **comments** and **escape sequences**.
- A **JSP expression** is used to insert a Java expression directly into the output. It has the following form:

```
<%= java expression %>
```

- The expression is evaluated, converted into a string, and set to the output stream of the servlet.



Scripting Constructs

- A **JSP scriptlet** enables you to insert a Java statement into the servlet's `jspService` method which is invoked by the service method. A JSP scriptlet has the following form:

```
<% java statement %>
```

- A **JSP declaration** is for declaring methods or fields into the servlet. It has the following form:

```
<%! java declaration %>
```

- HTML comments have the form:

```
<!-- HTML comment -->
```

- If you don't want the comment to appear in the resultant HTML file, use a **JSP comment** which has the form:

```
<%-- JSP comment -->
```





temperature.js x SimpleVariable.html x code.html x clock3.jsp x welcome.jsp x ComputeLoan.html x

ComputeLoan.html

```
1  <!-- ComputeLoan.html -->
2  <html>
3  <head>
4      <title>ComputeLoan</title>
5  </head>
6  <body bgcolor=white background=images/background.jpg lang=EN-US link=blue vlink=blue
7      style='tab-interval:.5in'>
8  <font size = 5><b>Compute Loan Payment</b></font>
9  <form method="get" action="/CNT4714/jsp/ComputeLoan.jsp"><p><p><br>
10     Loan Amount    <input type="text" name="loanAmount"><br><br>
11     Annual Interest Rate <input type="text" name="annualInterestRate"><br><br>
12     Number of Years <input type="text" name="numberOfYears" size="3"><br></p><p>
13     <input type="submit" name="Submit" value="Compute Loan Payment">
14     <input type="reset" value="Reset">
15     </p>
16 </form>
17 </body>
18 </html>
```





ComputeLoan.jsp

```
1 <!-- ComputeLoan.jsp -->
2 <html>
3 <head>
4     <title>ComputeLoan</title>
5 </head><body bgcolor=white background=images/background.jpg lang=EN-US link=blue vlink=blue
6 style='tab-interval:.5in'>
7 <% double loanAmount = Double.parseDouble( request.getParameter("loanAmount"));
8     double annualInterestRate = Double.parseDouble(request.getParameter("annualInterestRate"))
9     double numberOfYears = Integer.parseInt(request.getParameter("numberOfYears"));
10    double monthlyInterestRate = annualInterestRate / 1200;
11    double monthlyPayment = loanAmount * monthlyInterestRate /
12        (1 - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
13    double totalPayment = monthlyPayment * numberOfYears * 12;
14 <%>
15 <b><font size = 7> Loan Details </font></b><br><br>
16 <font size = 5>
17 Loan Amount:
18     <%= loanAmount %> <br><br>
19 Annual Interest Rate:
20     <%= annualInterestRate %> <br><br>
21 Number of Years:
22     <%= numberOfYears %> <br><br>
23 <b>
```

Java statements

Java expressions



Opera

ComputeLoan

localhost:8080/CNT4714/jsp/ComputeLoan.html

Compute Loan Payment

Loan Amount

Annual Interest Rate

Number of Years



Opera

ComputeLoan

localhost:8080/CNT4714/jsp/ComputeLoan.jsp

Loan Details

Loan Amount: 850000.0

Annual Interest Rate: 6.0

Number of Years: 20.0

Monthly Payment: \$6089.663997064467

Total Payment: \$1461519.3592954723



```
1 package code;
2
3 public class Loan {
4     private double annualInterestRate;
5     private int numOfYears;
6     private double loanAmount;
7     private java.util.Date loanDate;
8
9     /** Default constructor */
10    public Loan() {
11        this(7.5, 30, 100000);
12    }
13
14    /** Construct a loan with specified annual interest rate,
15     * number of years and loan amount
16     */
17    public Loan(double annualInterestRate, int numOfYears,
18        double loanAmount) {
19        this.annualInterestRate = annualInterestRate;
20        this.numOfYears = numOfYears;
21        this.loanAmount = loanAmount;
22        loanDate = new java.util.Date();
23    }
24
25    /** Return annualInterestRate */
26    public double getAnnualInterestRate() {
27        return annualInterestRate;
28    }
29
30    /** Set a new annualInterestRate */
31    public void setAnnualInterestRate(double annualInterestRate) {
32        this.annualInterestRate = annualInterestRate;
33    }
34
35    /** Return numOfYears */
36    public int getNumOfYears() {
```



```
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
hw3dbscript.sql x hw3dbscript-V2.sql x welcome.jsp x ComputeLoan.html x ComputeLoan.jsp x Loan.java x
35 /** Return numOfYears */
36 public int getNumOfYears() {
37     return numOfYears;
38 }
39
40 /** Set a new numOfYears */
41 public void setNumOfYears(int numOfYears) {
42     this.numOfYears = numOfYears;
43 }
44
45 /** Return loanAmount */
46 public double getLoanAmount() {
47     return loanAmount;
48 }
49
50 /** Set a newloanAmount */
51 public void setLoanAmount(double loanAmount) {
52     this.loanAmount = loanAmount;
53 }
54
55 /** Find monthly payment */
56 public double monthlyPayment() {
57     double monthlyInterestRate = annualInterestRate / 1200;
58     return loanAmount * monthlyInterestRate / (1 -
59         (Math.pow(1 / (1 + monthlyInterestRate), numOfYears * 12)));
60 }
61
62 /** Find total payment */
63 public double totalPayment() {
64     return monthlyPayment() * numOfYears * 12;
65 }
66
67 /** Return loan date */
68 public java.util.Date getLoanDate() {
69     return loanDate;
70 }
71 }
```

Java source file

length: 1721 lines: 72

Ln: 17 Col: 2 Sel: 0 | 0

UNIX

ANSI as UTF-8

INS



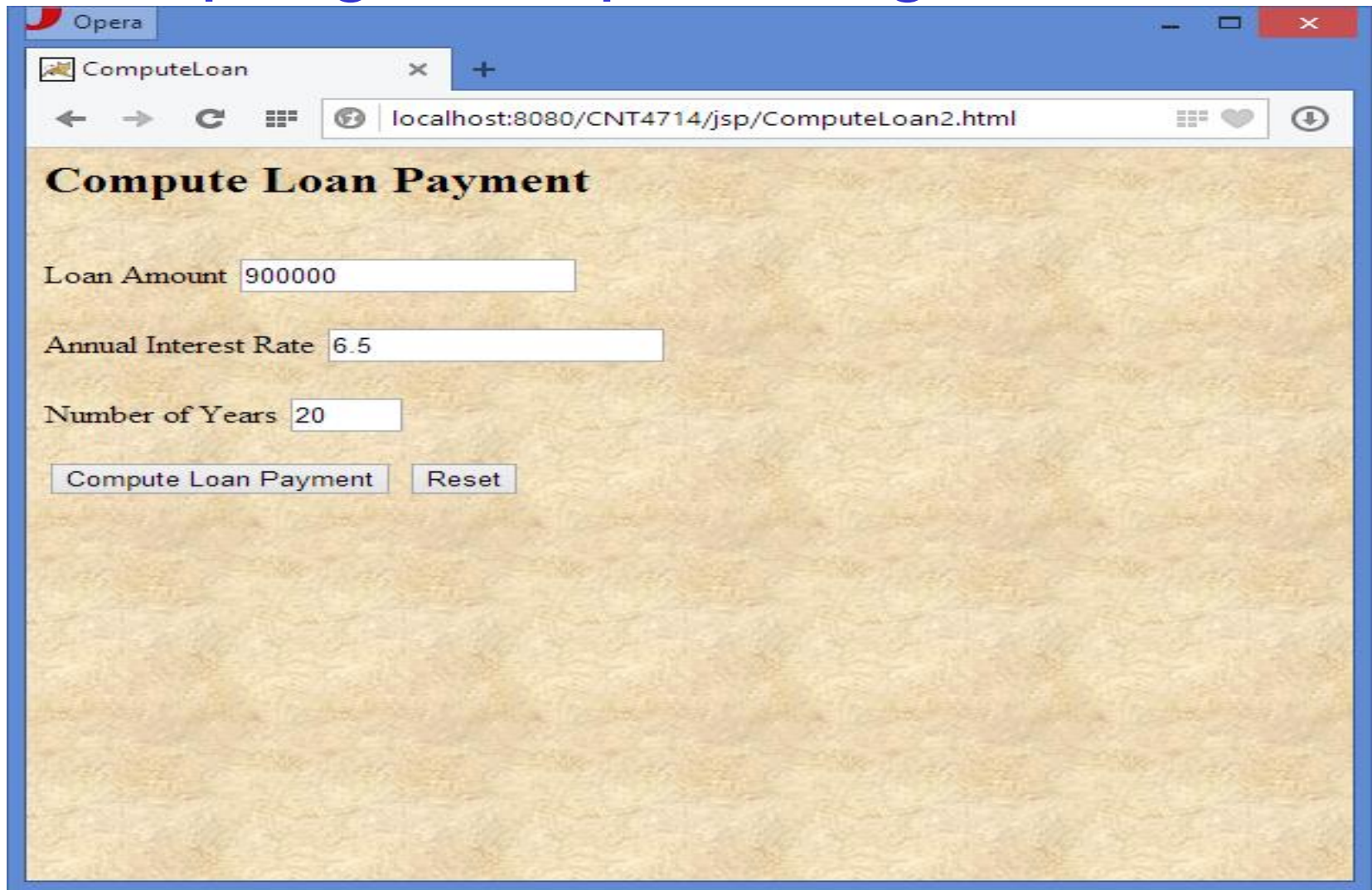


```
1  <!-- ComputeLoan2.jsp -->
2  <html>
3  <head>
4  <title>ComputeLoan</title>
5  </head><body bgcolor=white background=images/background.jpg lang=EN-US link=blue vlink=blue
6  style='tab-interval:.5in'>
7  <%@ page import = "code.Loan" %>
8
9  <% double loanAmount = Double.parseDouble( request.getParameter("loanAmount"));
10     double annualInterestRate = Double.parseDouble(request.getParameter("annualInterestRate"))
11     int numberOfYears = Integer.parseInt(request.getParameter("numberOfYears"));
12
13     Loan loan = new Loan (annualInterestRate, numberOfYears, loanAmount);
14 %>
15
16 <b><font size = 7> Loan Details </b></font><br><br>
17 <font size = 5>
18 Loan Amount:
19 <%= loanAmount %> <br><br>
20 Annual Interest Rate:
21 <%= annualInterestRate %> <br><br>
22 Number of Years:
23 <%= numberOfYears %> <br><br>
```

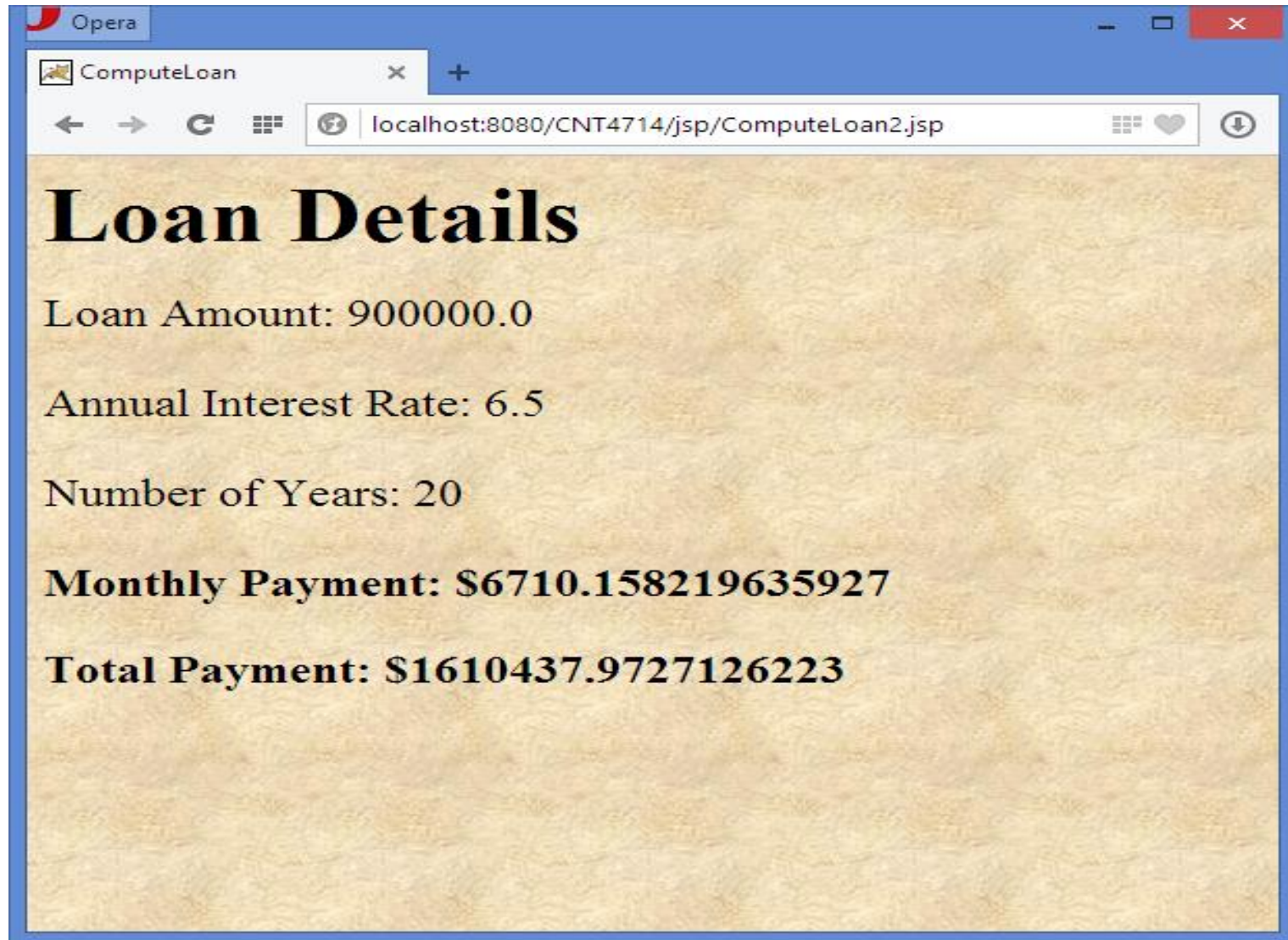
JSP directive to include a Java class.



Scripting Example Using Directives



Scripting Example Using Directives



The screenshot shows a web browser window with the following content:

- Browser: Opera
- Page Title: ComputeLoan
- Address Bar: localhost:8080/CNT4714/jsp/ComputeLoan2.jsp
- Page Content:
 - Loan Details**
 - Loan Amount: 900000.0
 - Annual Interest Rate: 6.5
 - Number of Years: 20
 - Monthly Payment: \$6710.158219635927**
 - Total Payment: \$1610437.9727126223**



JSP Standard Actions

- JSP standard actions provide programmers with access to several of the most common tasks performed in a JSP, such as including content from other resources, forwarding requests to other resources and interacting with JavaBean software components.
- JSP containers process actions at request time.
- Actions are delimited by `<jsp: action>` and `</jsp: action>`, where *action* is the standard action name.
 - In cases where nothing appears between the starting and ending tags, the XML empty element syntax `<jsp: action />` can be used.



JSP Standard Actions

<code><jsp: include></code>	Dynamically includes another resource in a JSP. As the JSP executes, the referenced resource is included and processed.
<code><jsp: forward></code>	Forwards request processing to another JSP, servlet or static page. This action terminates the current JSP's execution.
<code><jsp: plugin></code>	Allows a plug-in component to be added to a page in the form of a browser-specific object or embed HTML element. In the case of a Java applet, this action enables the browser to download and install the Java Plug-in, if it is not already installed on the client computer.
<code><jsp: param></code>	Used with the include, forward and plug-in actions to specify additional name-value pairs of information for use by these actions.



JSP Standard Actions

JavaBean Manipulation	
<code><jsp:useBean></code>	Specifies that the JSP uses a JavaBean instance (i.e., an object of the class that declares the JavaBean). This action specifies the scope of the object and assigns it an ID (i.e., a variable name) that scripting components can use to manipulate the bean.
<code><jsp:setProperty></code>	Sets a property in the specified JavaBean instance. A special feature of this action is automatic matching of request parameters to bean properties of the same name.
<code><jsp:getProperty></code>	Gets a property in the specified JavaBean instance and converts the result to a string for output in the response.



<jsp: include> Action

- JSPs support two include mechanisms – the `<jsp: include>` action and the `include` directive.
- Action `<jsp: include>` enables dynamic content to be included in a JSP at request time. If the included resource changes between requests, the next request to the JSP containing the `<jsp: include>` action includes the resource's new content.
- The `include` directive copies the content into the JSP once, at JSP translation time. If the included resource changes, the new content will not be reflected in the JSP that uses the `include` directive, unless the JSP is recompiled, which would normally occur only if a new version of the JSP were installed.



A JSP Using the `<jsp: include>` Action

```
<!DOCTYPE html>

<!-- include.jsp -->

<html lang="en">
  <head>
    <title>Using jsp:include</title>
    <style type = "text/css">
      body {
        font-family: tahoma, helvetica, arial, sans-serif;
      }
      table, tr, td {
        font-size: 1.1em;
        border: 3px groove;
        padding: 5px;
        background-color: #dddddd;
      }
    </style>
  </head>
```



```

<body>
  <table>
    <tr>
      <td style = "width: 250px; text-align: center">
        <img src = "smallucf.gif"
          width = "140" height = "93"
          alt = "pegasus logo" />
      </td>
      <td>
        <!-- include banner.html in this JSP -->
        <jsp:include page = "banner.html"
          flush = "true" />
      </td>
    </tr>
    <tr>
      <td style = "width: 250px">
        <!-- include toc.html in this JSP -->
        <jsp:include page = "toc.html" flush = "true" />
      </td>
      <td style = "vertical-align: top">
        <!-- include clock2.jsp in this JSP -->
        <jsp:include page = "clock2.jsp"
          flush = "true" />
      </td>
    </tr>
  </table>
</body>
</html>

```



Banner.html

```
<!-- banner.html -->
<!-- banner to include in another document -->
<div style = "width: 800px">
  <p>
    CNT 4714 - Enterprise Computing
    <br />
    Summer 2014 Semester - University of Central Florida
  </p>
  <p>
    <a href = "mailto:markl@cs.ucf.edu">markl@cs.ucf.edu</a>
  </p>
</div>
```



Table of Contents (toc.html)

```
<!-- toc.html -->
<!-- contents to include in another document -->
<p><a href = "http://www.cs.ucf.edu/courses/cnt4714/sum2014">
    CNT 4714 Course Webpage
</a></p>
<p><a href = "http://www.cs.ucf.edu/faculty/markl.html">
    Instructor's Webpage
</a></p>
<p><a href =
"http://www.cs.ucf.edu/courses/cnt4714/sum2014/code.html">
    Code Download Page
</a></p>
<p><a href =
"http://www.cs.ucf.edu/courses/cnt4714/sum2014/homework.html">
    Programming Assignments Page
</a></p>
<p>Send questions or comments about this site to
    <a href = "mailto:markl@cs.ucf.edu">
        markl@cs.ucf.edu
    </a><br />
</p>
```



Clock2.jsp

```
<!-- clock2.jsp -->
<!-- date and time to include in another document via redirection -->
<table>
  <tr>
    <td style = "background-color: black;">
      <p class = "big" style = "color: cyan; font-size: 3em;
        font-weight: bold;">
        <%-- script to determine client local and --%>
        <%-- format date accordingly --%>
        <%
          // get client locale
          java.util.Locale locale = request.getLocale();

          // get DateFormat for client's Locale
          java.text.DateFormat dateFormat =
            java.text.DateFormat.getDateTimeInstance(
              java.text.DateFormat.LONG,
              java.text.DateFormat.LONG, locale );
        %> <%-- end script --%>
        <%-- output date --%>
        <%= dateFormat.format( new java.util.Date() ) %>
      </p>
    </td>
  </tr>
</table>
```





CNT 4714 - Enterprise Computing
Summer 2014 Semester - University of Central Florida

markl@cs.ucf.edu

[CNT 4714 Course Webpage](#)

[Instructor's Webpage](#)

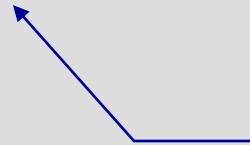
[Code Download Page](#)

[Programming Assignments Page](#)

Send questions or comments about this site to markl@cs.ucf.edu

June 24, 2014 12:25:39 PM EDT

Execution of include.jsp



<jsp: forward> Action

- JSP action <jsp: forward> enables a JSP to forward request processing to a different resource, such as an error page.
- Request processing by the original JSP terminates as soon as the JSP forwards the request.
- In the next example, this action is illustrated by forwarding a welcome request to another welcome page. JSP forward1.jsp forwards the request to JSP forward2.jsp. The forwarding action requests a date and time at which the original request was received that is forwarded.



Initial Forward JSP (forward1.jsp)

```
<!DOCTYPE html>
<!-- forward1.jsp -->
<html lang="en">
<head>
  <title>Forward request to another JSP</title>
</head>
<body>
  <% // begin scriptlet
    String name = request.getParameter( "firstName" );
    if ( name != null )
    {
  %> <%-- end scriptlet to insert fixed template data --%>

    <jsp:forward page = "forward2.jsp">
      <jsp:param name = "date"
        value = "<%= new java.util.Date() %>" />
    </jsp:forward>

  <% // continue scriptlet
    } // end if
```



Initial Forward JSP (forward1.jsp) (cont.)

```
else
    {
    %> <%-- end scriptlet to insert fixed template data --%>

        <form action = "forward1.jsp" method = "get">
            <p>Type your first name and press Submit</p>

            <p><input type = "text" name = "firstName" />
                <input type = "submit" value = "Submit" />
            </p>
        </form>

        <% // continue scriptlet

            } // end else

        %> <%-- end scriptlet --%>
    </body>
</html> <!-- end HTML document -->
```



Forward2 JSP (forward2.jsp)

```
<!DOCTYPE html>
<!-- forward2.jsp -->
<html lang="en">
<head>
  <title>Processing a forwarded request</title>
  <style type = "text/css">
    .big
    {
      font-family: tahoma, helvetica, arial, sans-serif;
      font-weight: bold;
      font-size: 2em;
    }
  </style>
</head>
<body>
  <p class = "big">
    Hello <%= request.getParameter( "firstName" ) %>, <br />
    Your redirection request was received <br /> and
    forwarded at
```



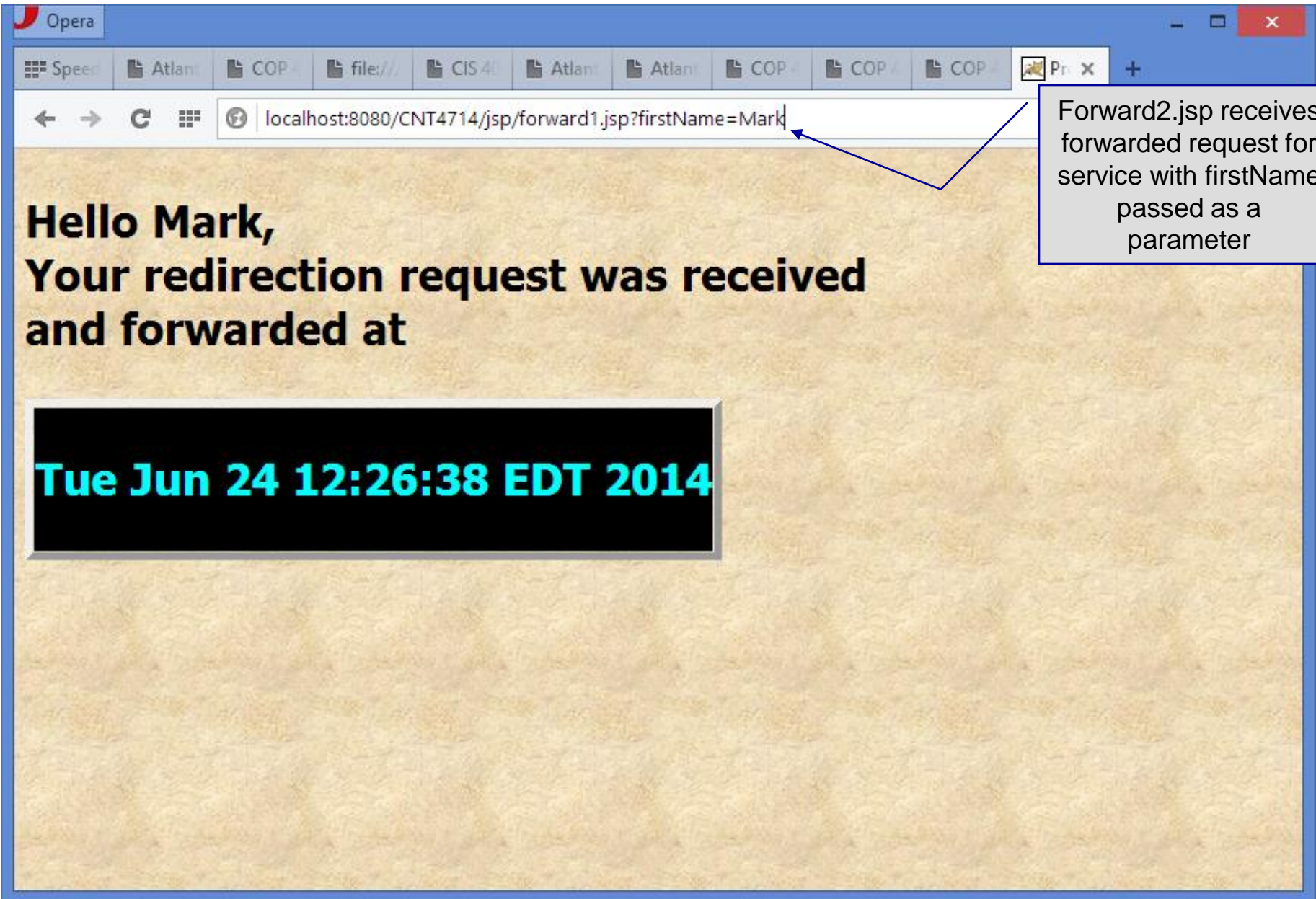
Forward2 JSP (forward2.jsp) (cont.)

```
</p>
  <table style = "border: 6px outset;">
    <tr>
      <td style = "background-color: black;">
        <p class = "big" style = "color: cyan;">
          <%= request.getParameter( "date" ) %>
        </p>
      </td>
    </tr>
  </table>
</body>
</html>
```



The screenshot shows an Opera browser window with several tabs open. The active tab is titled "Forward re". The address bar shows the URL "localhost:8080/CNT4714/jsp/forward1.jsp". The page content consists of the text "Type your first name and press Submit", a text input field containing the name "Mark", and a "Submit" button. A blue arrow points from a callout box to the "Submit" button. The callout box contains the text "Original request is invoked by forward1.jsp".





Forward2.jsp receives forwarded request for service with firstName passed as a parameter

