Web Services, Servlets, and JSPs

Sections 1A and 1B

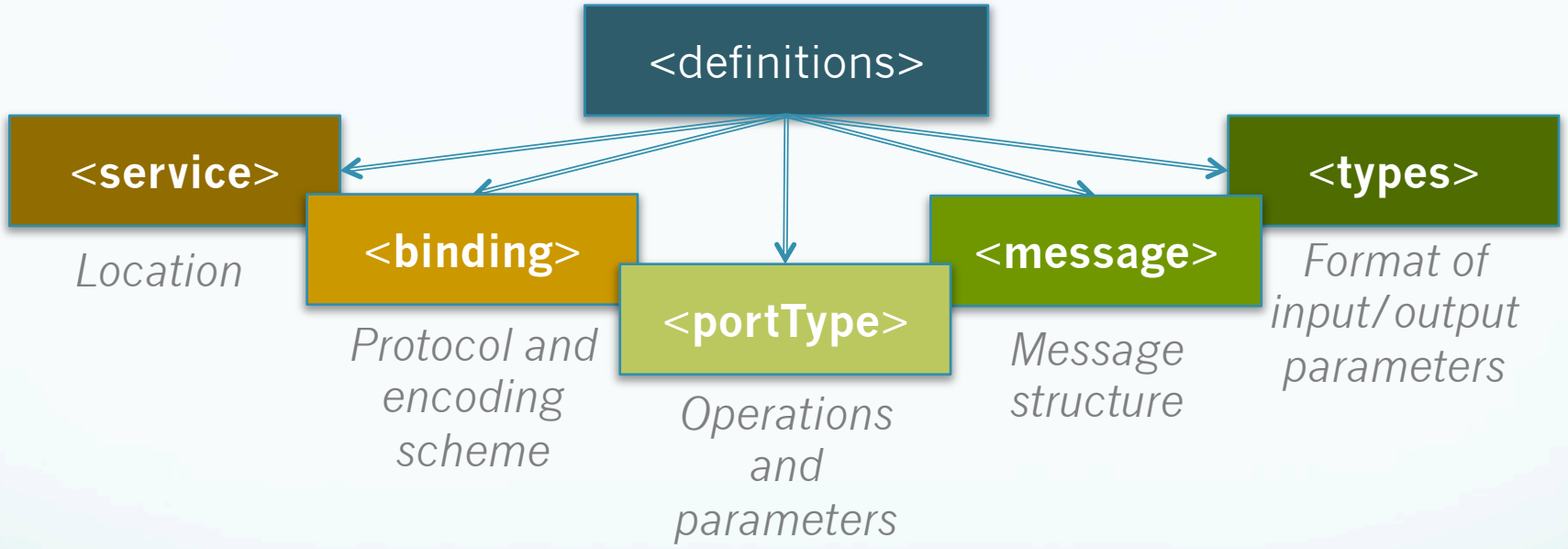
SOAP

- Recall our function **`fahrenheitToCelsius()`**

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
               xmlns:tn="http://oak.cs.ucla.edu/cs144/Converter">
  <soap:Body>
    <tn:fahrenheitToCelsius>
      <tn:fahrenheit>80</tn:fahrenheit>
    </tn:fahrenheitToCelsius>
  </soap:Body>
</soap:Envelope>
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
               xmlns:tn="http://oak.cs.ucla.edu/cs144/Converter">
  <soap:Body>
    <tn:fahrenheitToCelsiusResponse>
      <tn:fahrenheitToCelsiusReturn>26.67</tn:fahrenheitToCelsiusReturn>
    </tn:fahrenheitToCelsiusResponse>
  </soap:Body>
</soap:Envelope>
```

WSDL


<definitions>
<service>
<binding>
<portType>
<message>
<types>
Location
Protocol and encoding scheme
Operations and parameters
Message structure
Format of input/output parameters

WSDL Example

```
<?xml version="1.0"?>
  <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
           xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://oak.cs.ucla.edu/cs144/Converter"
           xmlns:tn="http://oak.cs.ucla.edu/cs144/Converter">

    <!-- types element specifies the "type" of input/output parameters -->
    <types>
      <xs:schema targetNamespace="http://oak.cs.ucla.edu/cs144/Converter">
        <!-- fahrenheightToCelsisus will be the root request element -->
        <xs:element name="fahrenheitToCelsius">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="fahrenheit" type="xs:double"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

        <!-- fahrenheitToCelsiusReponse is the root response element -->
        <xs:element name="fahrenheitToCelsiusResponse">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="farenheitToCelsiusReturn" type="xs:double"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:schema>
    </types>
```

WSDL Example

```
<!-- message element specifies request and response message format
     message element may consist of multiple parts
     like header, body, faultcode, etc-->
<message name="convertRequest">
  <part name="requestBody" element="tn:fahrenheitToCelsius"/>
</message>
<message name="convertResponse">
  <part name="responseBody" element="tn:fahrenheitToCelsiusResponse"/>
</message>

<!-- portType element specifies the available operations (= methods)
     and the associated input/output message format -->
<portType name="ConverterPortType">
  <operation name="fahrenheitToCelsius">
    <input message="tn:convertRequest"/>
    <output message="tn:convertResponse"/>
  </operation>
</portType>
```

WSDL Example

```
  <!-- binding element specifies the transfer protocol
       and the message MIME encoding of the service
       every operation specified here should have a corresponding
       operation in the associated portType -->
  <binding type="tn:ConverterPortType" name="ConverterBinding">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="fahrenheitToCelsius">
      <soap:operation soapAction=""/>
      <input>
          <soap:body use="literal"/>
      </input>
      <output>
          <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <!-- service element specifies the URL where the service is
       available -->
  <service name="ConverterService">
    <port binding="tn:ConverterBinding" name="Converter">
      <soap:address location="http://oak.cs.ucla.edu/cs144/Converter"/>
    </port>
  </service>
</definitions>
```

REST

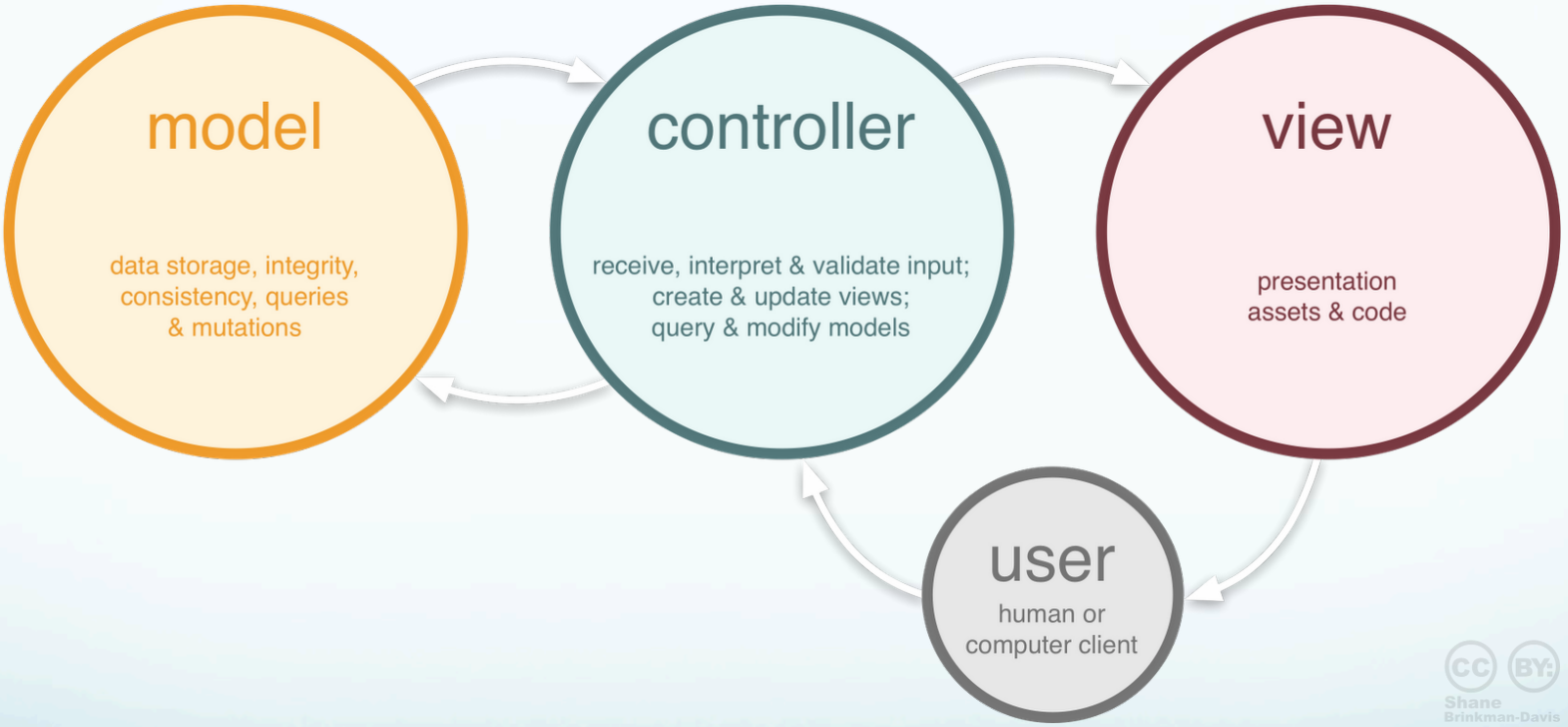
- Less complicated:

```
http://oak.cs.ucla.edu/cs144/Converter?method=fahrenheitToCelsius&fahrenheit=80
```

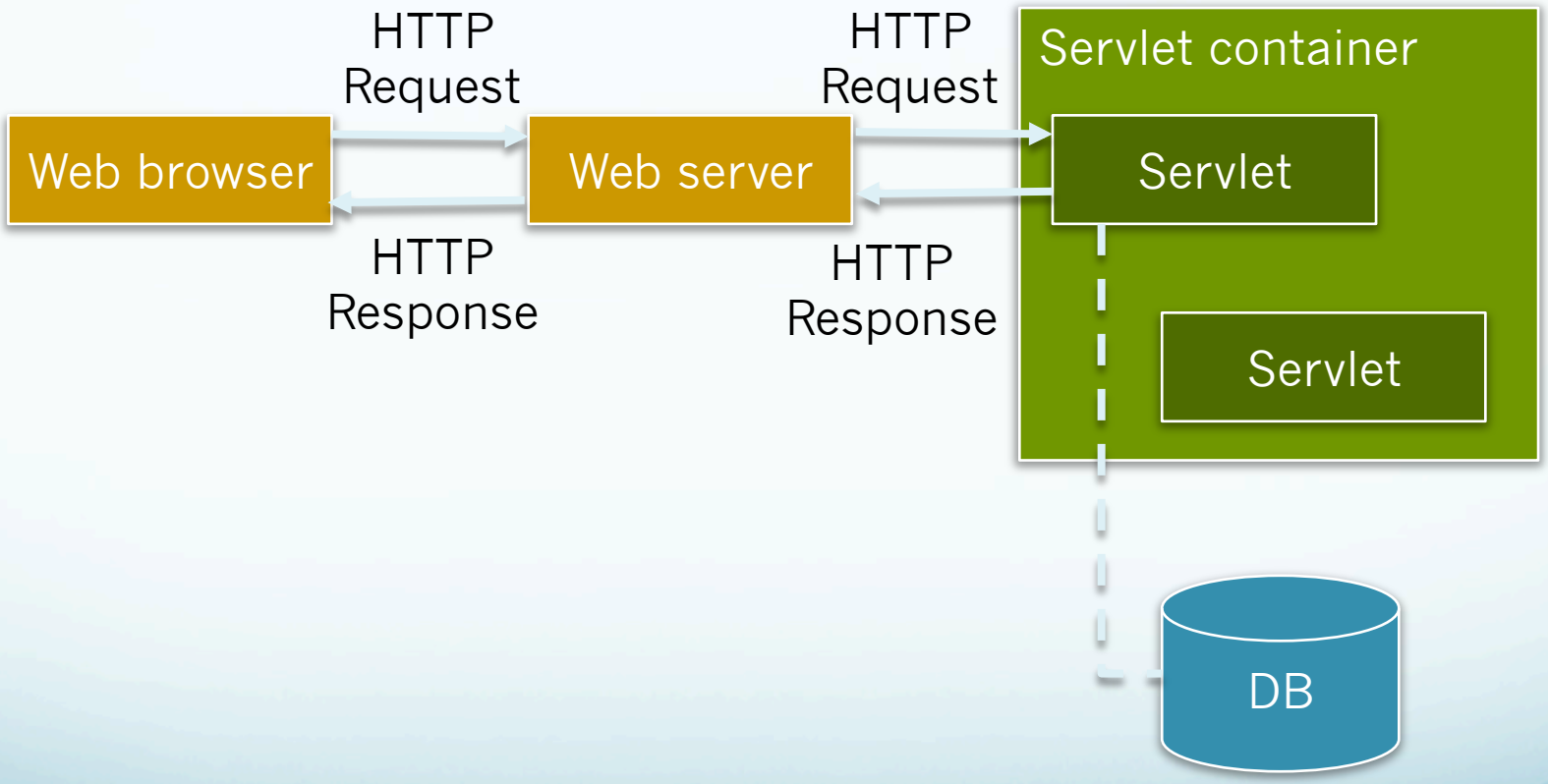
```
<?xml version="1.0"?>
<Celsius>26.67</Celsius>
```

- What are the problems?

MVC



Java Servlets



Servlet functions

- `void init( ServletConfig config )`
  - Executed once in the servlet lifetime.
  - Use it to open your DB connections and statements.
- `void service( ServletRequest request, ServletResponse response )`
  - The Servlet Container calls this method to respond to a client.
  - More appropriate to use doGet and doPost instead.
- `void destroy()`
  - Clean-up process.
  - Use it to close files, DB connections, and statements.

HTTP Servlet

- Overrides the class *`Servlet`* – we will use this one:
  - Instead of `service`, we have: `doDelete`, `doHead`, `doOptions`, `doPut`, `doTrace`, **`doGet`**, and **`doPost`**.
- In our VM, the Servlet API is at

  `$CATALINA_HOME/lib/servlet-api.jar`
- We can access parameters passed to the **doGet** and **doPost** methods by using:

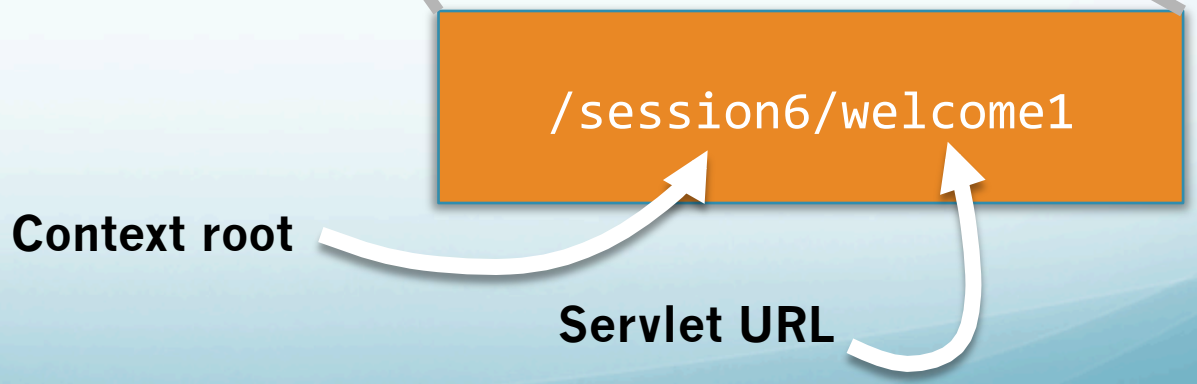
```
protected void doGet( HttpServletRequest request,
        HttpServletResponse response )
        throws ServletException, IOException
        {
                String firstName = request.getParameter( "firstName" );
                ...
```

Servlet Example

```
package angel;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class WelcomeServlet extends HttpServlet
{
          protected void doGet( HttpServletRequest request, HttpServletResponse response )
          throws ServletException, IOException
          {
                    response.setContentType( "text/html" );
                    PrintWriter out = response.getWriter();

                    // Prepare an HTML5 webpage for client.
                    out.println( "<html>" );                              // Head of the HTML page.

                    out.println( "<head>" );                              // Web page header.
                    out.println( "<title>My First Servlet</title>" );
                    out.println( "</head>" );

                    out.println( "<body>" );                              // Webp page content.
                    out.println( "<h1>Welcome to servlets!!!</h1>" );
                    out.println( "</body>" );

                    out.println( "</html>" );

                    // Close writer.
                    out.close();
          }
}
```

Servlet-Invoker Example

```
<!-- WelcomeServlet.html -->
<html>
        <head>
                <title>Handling an HTTP GET request</title>
                <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        </head>

        <body>
                <form action="/session6/welcome1" method="get">
                        <p>Click the button to invoke the servlet</p>
                        <p><input type="submit" value="Send request!" /></p>
                </form>
        </body>

</html>
```



Compiling a Servlet

- Put your servlets in their appropriate package structure:

  `/session6/angel/WelcomeServlet.java`

- When compiling in the VM, we need to add the servlet-api.jar as follows:

```
cd session6
javac -cp .:$CATALINA_HOME/lib/servlet-api.jar angel/*.java
```

Deployment Descriptor **web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app>
        <!-- General description of your web application -->
        <display-name>
                Our discussion session Servlet examples.
        </display-name>
        <description>
                This is the web application where we demonstrate our Servlet.
        </description>

        <!-- Servlet definitions -->
        <servlet>
                <servlet-name>welcome1</servlet-name>
                <description>A simple Servlet.</description>
                <servlet-class>angel.WelcomeServlet</servlet-class>
        </servlet>

        <!-- Servlet mappings -->
        <servlet-mapping>
                <servlet-name>welcome1</servlet-name>
                <url-pattern>/welcome1</url-pattern>
        </servlet-mapping>
</web-app>
```

Steps to Deploy a Servlet

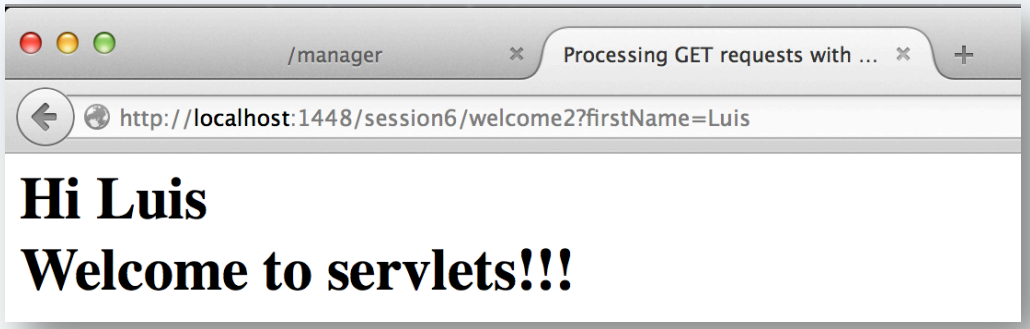
- Structure our web application as follows:

- Copy the entire **/session6** folder to

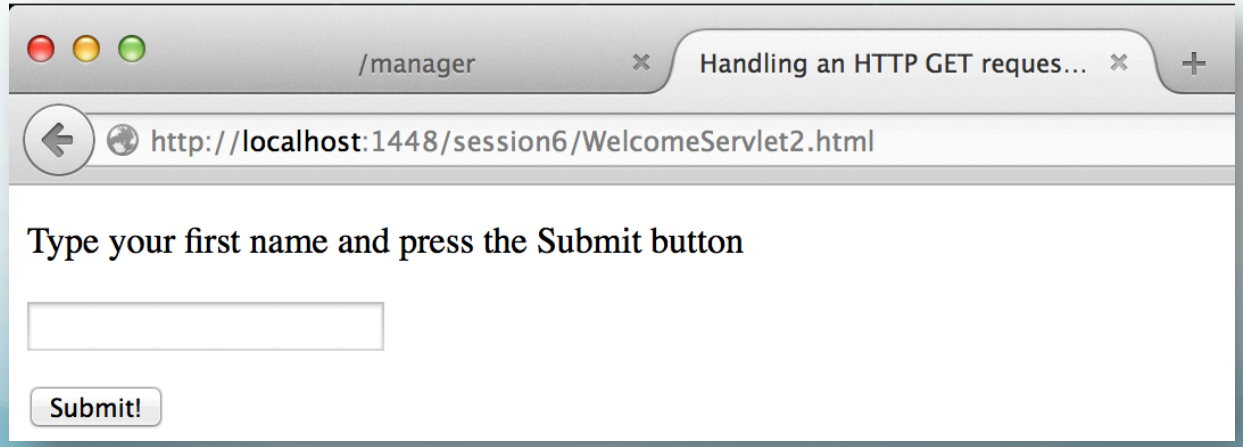
```
$CATALINA_BASE/webapps
```

Exercise

- Write a `WelcomeServlet2.java` that receives the parameter `firstName` and outputs a webpage with a personalized message:



- The **`firstName`** parameter must be read from a **`form`** within a web page `WelcomeServlet2.html`



Java Server Pages (**JSP**s)

- JSPs are **Servlets** in disguised.
- They look to the programmer more like HTML with Java embedded in it.
- The four components of JSPs are *directives*, *actions*, *scriptlets*, and *tag libraries*.
  - **Directives** `<%@___ %>` `page`, `include`, and `taglib`.
  - **Actions** `<jsp:`*action* `/>` `include`, `forward`, `param`, `useBean`, etc.
  - **Scriptlets** `<%` *block*`;` `%>`, `<%=` *expression* `%>`, `<%!` *declaration*`;` `%>`

forward1.jsp

```
<%@page import="java.util.*, java.text.*" %>

<html>
   <head>
      <title>Forward request to another JSP</title>
      <meta http-equiv="content-type" content="text/html; charset=UTF-8">
   </head>
   <body>
      <%  // Begin scriplet.
         String name = request.getParameter( "firstName" );
         if( name != null ) {
            Locale locale = request.getLocale(); // Get client locale.
            DateFormat dateFormat = DateFormat.getDateTimeInstance(
               DateFormat.LONG, DateFormat.LONG, locale );
      %>  <%-- End scriptlet to insert fixed template data --%>

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

      <%  // Continue scriptlet.
         } else {
      %>  <%-- End scriptlet to insert fixed template data --%>

      <jsp:include page="form.jsp" flush="true" /><!-- JSP action -->
      <%  // Continue scriptlet.
         }// End else.
      %>  <%-- End scriptlet --%>
   </body>
</html>
```

form.jsp

```
<form action="forward1.jsp" method="get">
        <p>Type your first name and press the Submit button</p>
        <p><input type="text" name="firstName" /></p>
        <p><input type="submit" value="Submit!" /></p>
</form>
```

forward2.jsp

```
<html>
   <head>
      <title>Processing a forwarded request</title>
      <meta http-equiv="content-type" content="text/html; charset=UTF-8">
      <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" ) %>.
         Your request was received and forwarded at</p>
      <p class="big" style="color:red"><%= request.getParameter( "date" ) %></p>
   </body>
</html>
```

Deploying JSPs

- We use our previous directory structure, and copy it again to `$CATALINA_BASE/webapps`