



JSF: Designing Custom Components

Originals of Slides and Source Code for Examples:

<http://www.coreservlets.com/JSF-Tutorial/>

This somewhat old tutorial covers JSF 1, and is left online for those maintaining existing projects. All new projects should use JSF 2, which is both simpler and more powerful. See <http://www.coreservlets.com/JSF-Tutorial/jsf2/>.

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android. Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



For live training on JSF 1 or 2, please see courses at <http://courses.coreservlets.com/>.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
 - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
 - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
 - Courses developed and taught by coreservlets.com experts (edited by Marty)
 - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services
- Contact hall@coreservlets.com for details**

Topics in This Section

- **Simple output-only components**
- **Components that accept attributes**
- **Components that accept input**

6

Main Pieces of Custom Components

- **Java class that extends `UIComponent`**
 - Usually extends `UIComponentBase` or an existing component
- **Custom JSP tag**
 - Associates component and renderer names with tag
- **TLD file that declares custom tag**
 - Normal Tag Library Descriptor file
- **Entry in `faces-config.xml`**
 - Associates class with the name used in the custom tag
- **JSP page that uses custom tag**
 - Imports tag library, uses tag

7

Summary of Process for Tag `<mytags:foo/>`

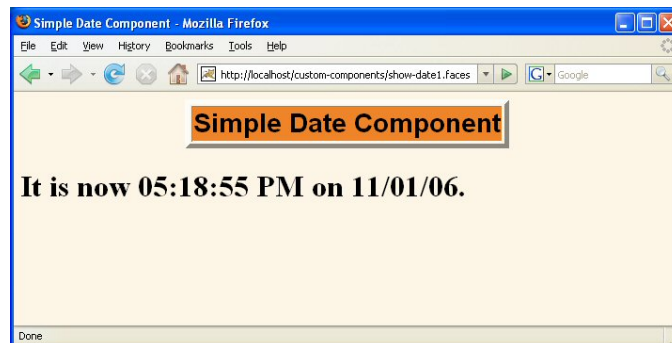
- Go to the TLD file and find the Java class associated with the tag "foo"
 - `<tag><name>foo</name>`
`<tag-class>myPackage.MyFooTag</tag-class>...</tag>`
- Run `getComponentType` from `MyFooTag`
 - ```
public String getComponentType() {
 return("bar");
}
```
- Find the name "bar" in `faces-config.xml`
  - `<component><component-type>bar</component-type>`  
`<component-class>`  
`myPackage.MyComponent`  
`</component-class></component>`
- Run the `encodeBegin` method of `MyComponent`
  - It outputs some HTML

8

## Simple Output-Only Component

- Outputs date in timezone of server
- Sample usage

```
<TABLE...>...
Simple Date Component</TH></TR></TABLE>
<H1><custom:simpleDate/></H1>
```
- Sample output



9

## Java Class that Extends UIComponent

- **You usually extend UIComponentBase**
  - Unless you extend existing component
  - If you output HTML, conventional to name class *HtmlBlahBlah*
- **Override encodeBegin or encodeEnd**
  - For simple components, can do either
  - For components with input, use encodeEnd
  - Obtain Writer via `getResponseWriter`
  - Output desired markup
    - write (use `String.format` to build output)
    - `startElement`, `writeAttribute`, `writeText`, `endElement`
    - Complex tags let separate Renderer create output
- **Override getFamily**
  - If there is no separate renderer, return null

10

## Class that Extends UIComponent: Code

```
package coreservlets;

import javax.faces.component.*;
import javax.faces.context.*;
import java.io.*;
import java.util.*;

public class HtmlSimpleDate extends UIComponentBase {
 public void encodeBegin(FacesContext context)
 throws IOException {
 ResponseWriter out = context.getResponseWriter();
 Calendar currentDateTime = new GregorianCalendar();
 String output =
 String.format("It is now %tr on %tD.",
 currentDateTime, currentDateTime);
 out.write(output);
 }

 public String getFamily() {
 return null;
 }
}
```

11

# Custom JSP Tag

- **Extend UIComponentTag**
  - From javax.faces.webapp package
- **Override getComponentType**
  - Must match name used in faces-config.xml
  - Common to just use the base classname
- **Override getRendererType**
  - For components without separate renderer, simply return null

12

# Custom JSP Tag: Code

```
package coreservlets;

import javax.faces.webapp.*;

public class HtmlSimpleDateTag extends UIComponentTag {
 public String getComponentType() {
 // Associates tag with component in faces-config.xml
 return("HtmlSimpleDate");
 }

 public String getRendererType() {
 // Component renders itself in encodeBegin,
 // so return null.
 return(null);
 }
}
```

13



# TLD File

- **Mostly standard Tag Library Descriptor**
  - Must be named *something.tld*
  - Goes somewhere under WEB-INF
    - E.g., WEB-INF/tlds
- **Conventional to declare uri**
  - The "f" and "h" JSF tags are imported by "fake" address
    - `<%@ taglib uri="http://..." ...>`, not `<%@ taglib uri="/WEB-INF/..." ...>`
  - For consistency, use same approach for your tags
    - This address need not really exist!
- **Parent tag declares common JSF attributes**
  - E.g., `id`, `rendered`
  - Must declare them in TLD file if you want to use them

14

# TLD File: Code

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib ... version="2.0">
 <tlib-version>1.0</tlib-version>
 <short-name>jsf-custom-components</short-name>
 <uri>http://coreservlets.com/jsf/simple</uri>

 <tag>
 <description>
 Outputs date in server's timezone
 </description>
 <name>simpleDate</name>
 <tag-class>coreservlets.HtmlSimpleDateTag</tag-class>
 <body-content>empty</body-content>
 <attribute><name>id</name></attribute>
 <attribute><name>rendered</name></attribute>
 </tag>
</taglib>
```

15

# faces-config.xml

- **Declare with component element**
  - component-name
    - The name as given by getComponentName in the tag file
  - component-class
    - The fully qualified class name

16

# faces-config.xml: Code

```
<?xml version="1.0"?>
<!DOCTYPE ...>
<faces-config>
 <component>
 <component-type>HtmlSimpleDate</component-type>
 <component-class>
 coreservlets.HtmlSimpleDate
 </component-class>
 </component>
 ...
</faces-config>
```

17

# JSP Page

- **Use taglib directive to import custom tag library**
  - For uri attribute, use "fake" address from TLD file
    - `<%@ taglib uri="http://coreservlets.com/jsf/simple" prefix="custom" %>`
- **Still use normal JSF approach**
  - Import f library
  - Import h library
  - Surround entire page in f:view
  - Use h:form to build forms

18

# JSP Page: Code

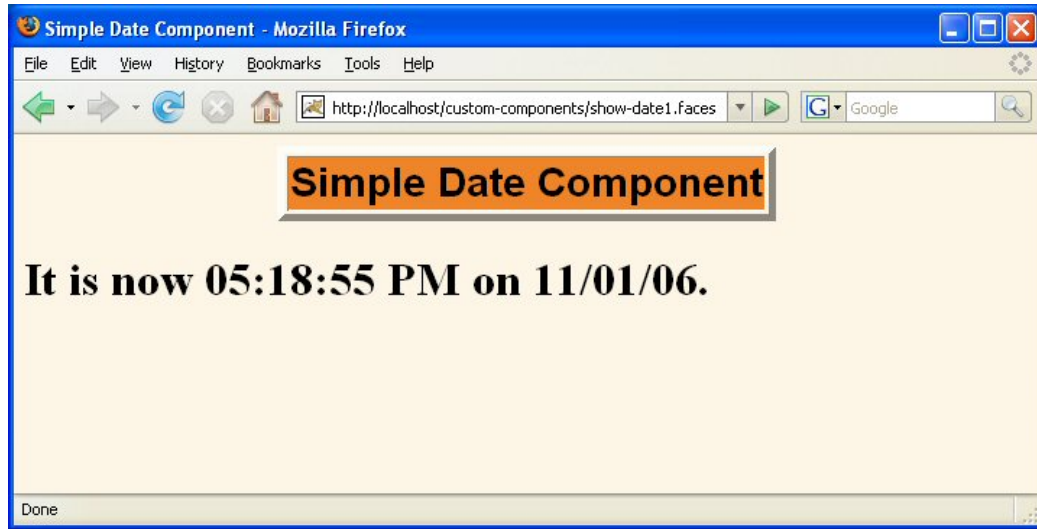
```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://coreservlets.com/jsf/simple"
 prefix="custom" %>

<f:view>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
 Transitional//EN">
<HTML>
<HEAD><TITLE>Simple Date Component</TITLE>
<LINK REL="STYLESHEET"
 HREF="./css/styles.css"
 TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
 <TR><TH CLASS="TITLE">Simple Date
 Component</TH></TR></TABLE>
<H1><custom:simpleDate/></H1>
</BODY></HTML>
</f:view>
```

19



# JSP Page: Result



20

# Component with Attributes

- **Outputs date in city specified in JSP page**
  - Or error message if city is unrecognized

- **Sample usage**

```

 <custom:date city="New York"/>
 <custom:date city="Boston"/>
 <custom:date city="San Francisco"/>
 <custom:date city="#{cityBean.city}"/>
 <custom:date city="Javatown"/>

```

- **Sample output**



21

## Java Class that Extends UIComponent

- **Still follow previous approach**
  - Extend UIComponentBase
  - Override encodeBegin
    - Obtain Writer with getResponseWriter
  - Override getFamily
    - Return null since no separate renderer
- **encodeBegin needs access to "city" attribute from tag**
  - Call getAttributes to obtain a Map of all attributes
  - Call "get" on Map to retrieve individual attribute
  - Separate TimeZone class returns time in the city
    - Code not shown, but online and downloadable

22

## Class that Extends UIComponent: Code

```
...

public class HtmlDate extends UIComponentBase {
 public void encodeBegin(FacesContext context)
 throws IOException {
 ResponseWriter out = context.getResponseWriter();
 Calendar currentDateTime = new GregorianCalendar();
 String city = (String) getAttributes().get("city");
 String output =
 TimeZone.getTimeString(city, currentDateTime);
 out.write(output);
 }

 public String getFamily() {
 return null; }
}
```

23

# Custom JSP Tag

- **Still follow previous approach**
  - Extend `UIComponentTag`
  - Override `getComponentType` to return name given in `faces-config.xml` (usually name of main class)
  - Override `getRendererType` to return null, signifying that there is no separate renderer (component does rendering)
- **Follow standard JSP tag practice**
  - `setBlah` method for each *blah* attribute
- **Override `setProperties`**
  - Call `super.setProperties`
  - Check each attribute to see if it is a value expression (i.e., uses EL ala `{beanName.propertyName}`)
    - If so, create and store a value binding
    - If not, call `getAttributes().put` to store value

24

# Custom JSP Tag: Code

```
public class HtmlDateTag extends UIComponentTag {
 private String city = "Javatown";

 public void setCity(String city) {
 this.city = city;
 }

 public String getComponentType() {
 // Associates tag with component in faces-config.xml
 return("HtmlDate");
 }

 public String getRendererType() {
 // Component renders itself in encodeBegin,
 // so return null.
 return(null);
 }
}
```

25

## Custom JSP Tag: Code (Continued)

```
protected void setProperties(UIComponent component) {
 super.setProperties(component); // Always call this!
 if(isValueReference(city)) {
 FacesContext context =
 FacesContext.getCurrentInstance();
 Application app = context.getApplication();
 ValueBinding binding = app.createValueBinding(city);
 component.setValueBinding("city", binding);
 } else {
 component.getAttributes().put("city", city);
 }
}
}
```

26

## TLD File

- **Still follow previous approach**
  - Use uri attribute to define "fake" address
  - Standard taglib approach with tag-name and tag-class
  - Declare standard JSF attributes defined by parent class (especially id and rendered)
- **Declare new attribute**
  - Define city to be required
    - <attribute>
    - <name>city</name>
    - <required>true</required>
    - </attribute>

27

## TLD File: Code

```
<uri>http://coreservlets.com/jsf/simple</uri>
...
<tag>
 <description>
 Outputs the date in the specified city
 </description>
 <name>date</name>
 <tag-class>coreservlets.HtmlDateTag</tag-class>
 <body-content>empty</body-content>
 <attribute>
 <name>city</name>
 <required>true</required>
 </attribute>
 <attribute>
 <name>id</name>
 </attribute>
 <attribute>
 <name>rendered</name>
 </attribute>
</tag>
```

28

## faces-config.xml

- **Still follows previous approach**
  - Uses component element
    - component-name matches value returned by `getComponentType` in tag file
    - component-class matches fully qualified component class
- **CityBean declared normal way**
  - Using managed-beans entry

29

## faces-config.xml: Code

```
<faces-config>
 ...
 <component>
 <component-type>HtmlDate</component-type>
 <component-class>
 coreservlets.HtmlDate
 </component-class>
 </component>
 <managed-bean>
 <managed-bean-name>cityBean</managed-bean-name>
 <managed-bean-class>
 coreservlets.CityBean
 </managed-bean-class>
 <managed-bean-scope>session</managed-bean-scope>
 </managed-bean>
 ...
</faces-config>
```

30

## CityBean: Code

```
package coreservlets;

import java.io.*;

public class CityBean implements Serializable {
 private String city = "Chicago";

 public String getCity() { return(city); }

 public void setCity(String city) {
 this.city = city;
 }
}
```

31



# JSP Page

- **Still follows previous approach**
  - Standard JSF approach
    - Import f and h libraries
    - Surround page with f:view
    - Use h:form for forms
  - Import custom library using "fake" address for uri
- **Supply value for city attribute**

32

# JSP Page: Code

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://coreservlets.com/jsf/simple"
 prefix="custom" %>

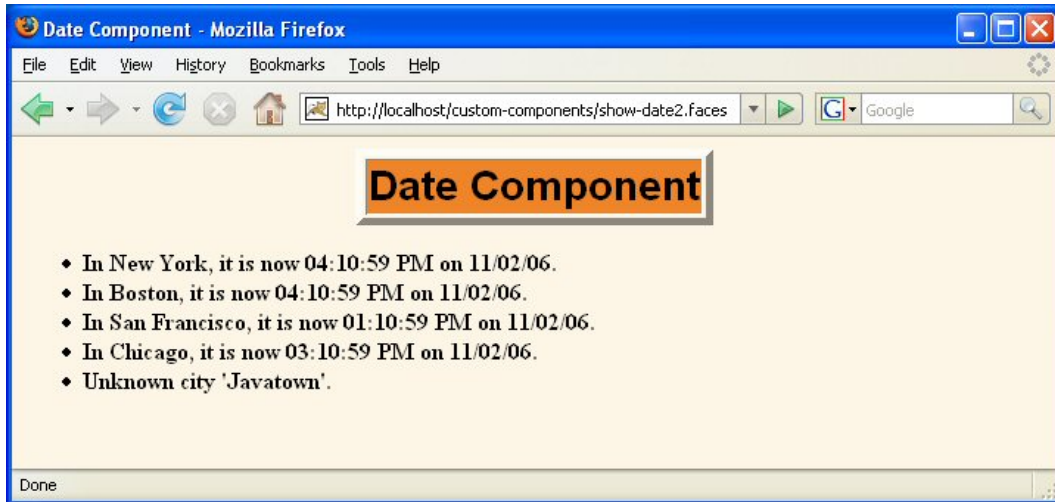
<f:view>
<!DOCTYPE ...>
...
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
 <TR><TH CLASS="TITLE">Date Component</TH></TR></TABLE>

 <custom:date city="New York"/>
 <custom:date city="Boston"/>
 <custom:date city="San Francisco"/>
 <custom:date city="#{cityBean.city}"/>
 <custom:date city="Javatown"/>

</BODY></HTML>
</f:view>
```

33

# JSP Page: Result

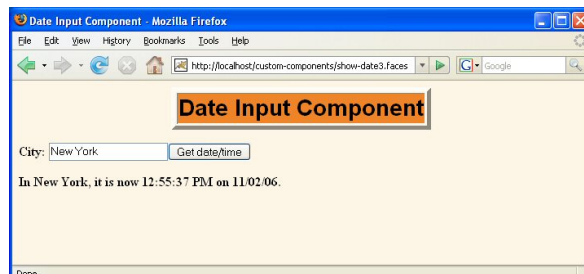


34

# Component that Accepts Input

- **Outputs date in city specified by end user**
  - Combination of four elements
    - Prompt
    - Textfield for city
    - Submit button
    - Output field for time/date
- **Sample usage**

```
<h:form>
 <custom:dateInput/>
</h:form>
```
- **Sample output**



35

## Component Class: New Features

- **Previously**
  - Component had `encodeBegin` or `encodeEnd` to say what output should look like
- **Now**
  - Component has `encodeEnd` to say what output should look like
    - But output might use the value of the component (call `getValue` to get it) in various parts of the output
    - If output contains HTML input elements, the elements must have unique IDs
  - Component has `decode` to say what should happen when a form containing the component is submitted
    - It does raw servlet code to read request parameters
    - It stores final resultant value with `setSubmittedValue`

36

## Java Class that Extends `UIComponent`

- **Extend `UIInput` instead of `UIComponentBase`**
  - Call `setRendererType(null)` in constructor
  - *Do not override `getFamily`*
- **Override `encodeEnd` instead of `encodeBegin`**
  - To make sure `Converter` is used if available
- **Override `decode`**
  - Tell system how to get data from POST submission
    - Get request params from `ExternalContext`
- **Give unique id's to all form elements**
  - Append string on end of `getClientId`
- **Get value**
  - Overall value: call `getValue`
    - Both initial value and value after submission
  - Submitted value: call `getAttributes().get("value")`
    - Value after submission (null initially)

37

## Class that Extends UIComponent: Code

```
public class HtmlDateInput extends UIInput {

 public HtmlDateInput() {
 setRendererType (null);
 }

 public void encodeEnd (FacesContext context)
 throws IOException {
 encodePrompt (context);
 encodeCityField (context);
 encodeDateButton (context);
 encodeDateOutput (context);
 }
}
```

38

## Class that Extends UIComponent: Code (Continued)

```
 public void encodePrompt (FacesContext context)
 throws IOException {
 ResponseWriter out = context.getResponseWriter();
 String output = String.format("City: ");
 out.write (output);
 }

 public void encodeCityField (FacesContext context)
 throws IOException {
 ResponseWriter out = context.getResponseWriter();
 String value = (String)getValue();
 if (value == null) { // Don't put 'null' in field
 value = "";
 }
 String output = String.format
 ("<input type='text' name='%s' value='%s'>",
 getCityFieldId (context), value);
 out.write (output);
 }
}
```

39

## Class that Extends UIComponent: Code (Continued)

```
public void encodeDateButton(FacesContext context)
 throws IOException {
 ResponseWriter out = context.getResponseWriter();
 String output = String.format
 ("<input type='submit' name='%s' value='%s'>%n",
 getDateButtonId(context), "Get date/time");
 out.write(output);
}

public void encodeDateOutput(FacesContext context)
 throws IOException {
 ResponseWriter out = context.getResponseWriter();
 String city = (String) getAttributes().get("value");
 String timeString = "Date/time will go here.";
 if (city != null) {
 Calendar currentDateTime =
 new GregorianCalendar();
 timeString =
 TimeZone.getTimeString(city, currentDateTime);
 }
 String output =
 String.format("<p>%s</p>%n", timeString);
 out.write(output);
}
```

40

## Class that Extends UIComponent: Code (Continued)

```
@SuppressWarnings("unchecked")

public void decode(FacesContext context) {
 ExternalContext externalContext =
 context.getExternalContext();
 Map<String, String> requestParams =
 externalContext.getRequestParameterMap();
 String clientId = getCityFieldId(context);
 String submittedCity = requestParams.get(clientId);
 setSubmittedValue(submittedCity);
}

private String getCityFieldId(FacesContext context) {
 return(getClientId(context) + ":cityField");
}

private String getDateButtonId(FacesContext context) {
 return(getClientId(context) + ":dateButton");
}
```

41

# Custom JSP Tag

- **Still follow previous approach**
  - Extend `UIComponentTag`
  - Override `getComponentType` to return name given in `faces-config.xml` (usually name of main class)
  - Override `getRendererType` to return null, signifying that there is no separate renderer (component does rendering)

42

# Custom JSP Tag: Code

```
package coreservlets;

import javax.faces.webapp.*;

public class HtmlDateInputTag extends UIComponentTag {
 public String getComponentType() {
 // Associates tag with component in faces-config.xml
 return("HtmlDateInput");
 }

 public String getRendererType() {
 // Component renders itself in encodeEnd,
 // so return null.
 return(null);
 }
}
```

43



# TLD File

- **Still follow previous approach**
  - Use uri attribute to define "fake" address
  - Standard taglib approach with tag-name and tag-class
  - Declare standard JSF attributes defined by parent class (especially id and rendered)

44

# TLD File: Code

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib ...>
 ...
 <uri>http://coreservlets.com/jsf/simple</uri>
 ...
 <tag>
 <description>
 Lets the user enter a city, and shows the date
 for that city.
 </description>
 <name>dateInput</name>
 <tag-class>coreservlets.HtmlDateInputTag</tag-class>
 <body-content>empty</body-content>
 <attribute>
 <name>id</name>
 </attribute>
 <attribute>
 <name>rendered</name>
 </attribute>
 </tag>
</taglib>
```

45

# faces-config.xml

- **Follows exact same approach as before**
  - Uses component element
    - component-name matches value returned by `getComponentType` in tag file
    - component-class matches fully qualified component class

46

# faces-config.xml: Code

```
<?xml version="1.0"?>
<!DOCTYPE ...>
<faces-config>
 ...
 <component>
 <component-type>HtmlDateInput</component-type>
 <component-class>
 coreservlets.HtmlDateInput
 </component-class>
 </component>
 ...
</faces-config>
```

47

# JSP Page

- **Still follows previous approach**

- Standard JSF approach
  - Import f and h libraries
  - Surround page with f:view
  - Use h:form for forms
- Import custom library using "fake" address for uri

48

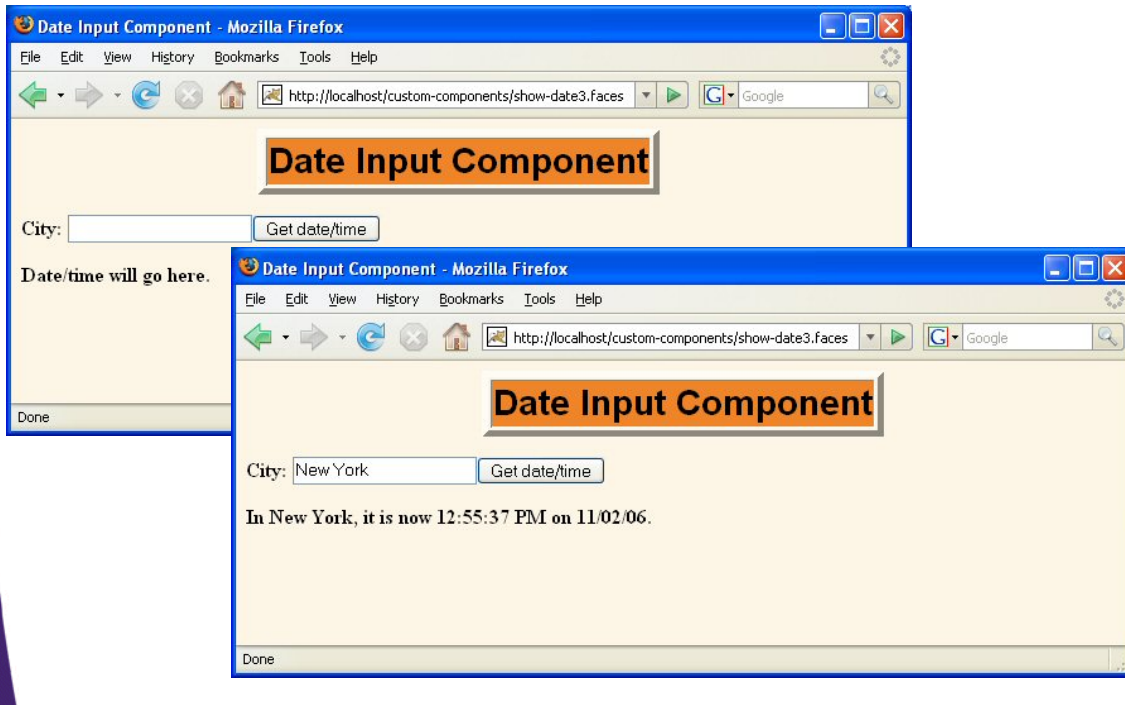
# JSP Page: Code

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://coreservlets.com/jsf/simple"
 prefix="custom" %>

<f:view>
<!DOCTYPE ...>
...
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
 <TR><TH CLASS="TITLE">Date Input
 Component</TH></TR></TABLE>
<P>
<h:form>
 <custom:dateInput/>
</h:form>
</BODY></HTML>
</f:view>
```

49

# JSP Page: Results



50

## Summary

- **Simple output-only components**
  - Component extends `UIComponentBase`
    - Override `encodeBegin` (create output) and `getFamily` (return null)
  - Tag extends `UIComponentTag`
    - Override `getComponentType` (a name) & `getRendererType` (null)
- **Components that accept attributes**
  - Component extends `UIComponentBase`
    - `encodeBegin` calls `getAttributes().get("attribute-name")`
  - Tag extends `UIComponentTag`
    - `setProperties` checks for value binding and stores attribute
- **Components that accept input**
  - Component extends `UIInput`
    - Override `encodeEnd`
      - Overall value: `getValue()`;
      - Submitted value: `getAttributes().get("attribute-name")`
    - Override `decode` and process request parameters

51



# Questions?

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.