Events II, ValueChangeListener

ActionListener was used *for* buttons. Form was automatically submitted when clicked, unless immediate attribute is true

ValueChangeListener applies to enythig that can take value:
combobox, listbox, radio button, checkbox, textfield, etc.

Difference between ValueChangeListener and ActionListener

- Form not automatically submitted
- Need to add JavaScript to submit the form onclick="submit()" or onchange="submit()"
- If you take out the onchange or onclick attribute, the form will not be submitted when let's say selected menu item is changed

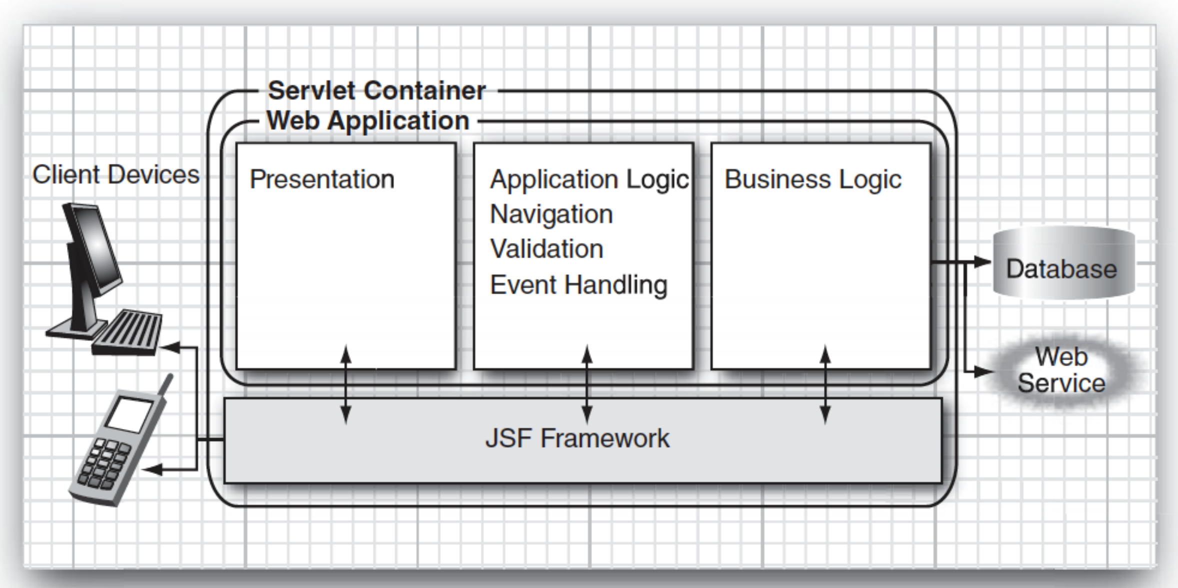
Useful ValueChangeEvent methods

getComponent
getOldValue
getNewValue
– Needed since bean has probably not been populated
– Value for checkbox is of type Boolean
– Value for radio button or textfield corresponds to request parameter

Example

```
public void someMethod(ValueChangeEvent event) {
   Boolean flag = (Boolean)event.getNewValue();
   takeActionBasedOn(flag);
}
```

Immediate Components



Figure| 1–11 High-level overview of the JSF framework

Model-view-controller architecture (MVC)

Presentation is JSF pages (View)

Application Logic

1. Managed Beans interacting with Presentation doing navigation and validation via EVENTS.
2. Interacts with the Business Logic via PLACE holders

Model

The data in the database. The Business Logic manipulates the data of the database .

All software applications let users manipulate certain data, such as shopping carts, travel itineraries, etc. . This data is called the model, in a database.

Web Services (run in background) ALLOW other applications (Android programs, IPhones, any program) to interact with the database.

The view component can be wired to a bean property of a model object, such as:
<h:inputText value="#{user.name}"/>
So, the JSF implementation operates as the controller that reacts to the user by processing *action* and *value change* events, to the Managed Bean and updates the model or the view or both.
Example, User Login

1. User clicks <h:commandButton value="Login" action="#{user.check}"/>
2. JSF implementation invokes the check method of the Managed Bean user .
3. The method will check the database via a PlaceHolder variable, and it returns the ID of the next page to be displayed.

Rendering Pages in Browser

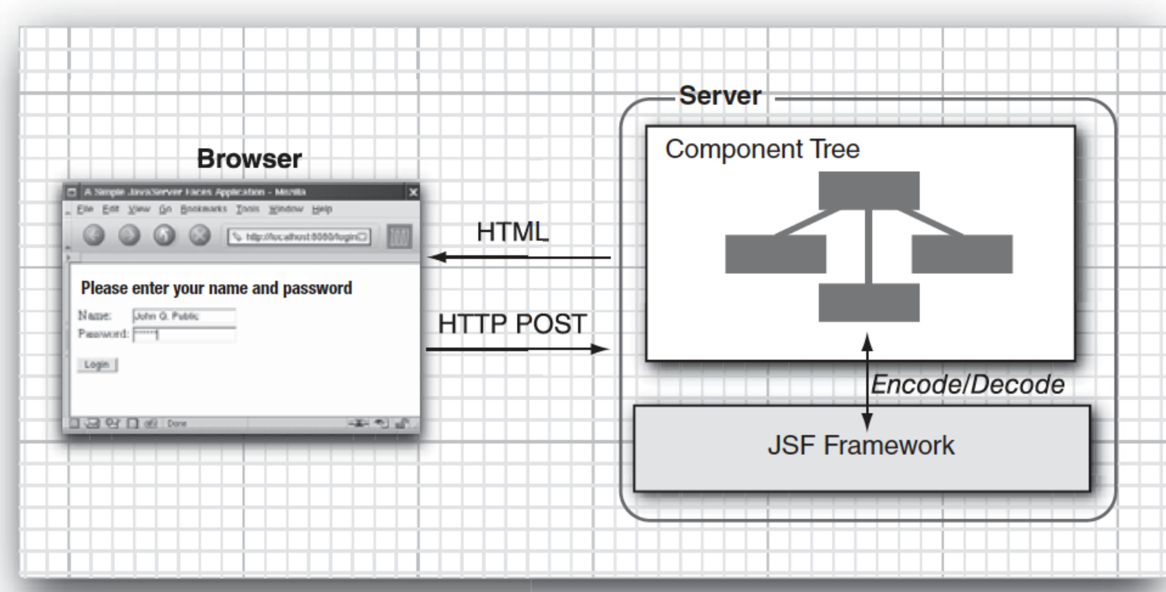


Figure 1–13 Encoding and decoding JSF pages

This process is called encoding.

0. All text that is not a JSF tag is passed through.
1. The h:form, h:inputText, h:inputSecret, and h:commandButton tags are converted to HTML.
2. Each of these tags gives rise to an associated component.
3. Each component has a renderer that produces HTML output, reflecting the component state. For example, the renderer for the component that corresponds to the h:inputText tag produces the following output:
   <input type="text" name="unique ID" value="current value"/>

The IDs can look rather random, such as _id_id12:_id_id21.
The encoded page is sent to the browser, and the browser displays it .

This process is called decoding.

1. The form data is placed in a hash table that all components can access.

2. The JSF implementation gives each component a chance to inspect that hash table. Each component decides on its own how to interpret the form data.

   For a login form has three component objects: two UIInput objects that correspond to the text fields on the form and one UICommand object that corresponds to the submit button.
   - The UIInput components updates the bean properties referenced in the value attributes: they invoke the setter methods with the values that the user supplied.
   - The UICommand component checks whether the button was clicked. If so, it fires an action event to launch the login action referenced in the action attribute. That event tells the navigation handler to look up the successor page, welcome.xhtml .

Now the cycle of encoding/decoding repeats.

The JSF Life Cycle during the Encoding-Decoding

The JSF specification defines six distinct phases:

1. Restore View

    1. Retrieves the component tree for the requested page if it was displayed previously or constructs a new component tree if it is displayed for the first time.

    2. If there are no request values (no query data) , the JSF implementation skips ahead to the Render Response phase. This happens when a page is displayed for the first time.

2. Apply Request Values

    JSF implementation iterates over the component objects in the component tree.
    Each component object checks which request values belong to it and stores them. (i.e text box Component stores the text of the textbook). They are called "local values".

3. Process Validation

    1. IF you can attach *validators* in tags the JSF perform correctness checks on the local values.
    2. IF a conversion or validation errors occur, we skip to Render Response (6) phase directly, redisplaying the current page so that the user has another chance to provide correct inputs.

4. Update Model Values

    If converters(can be our converters we will see examine them later) and validators had no errors, it is the local values are used to update the managed beans wired to the components.

5. Invoke Application

    Actions for a method with String return type for the button or link component that caused the form submission is executed. The action method returns the string that is passed to JSF-navigation-handler, if not void. The navigation handler looks up the next page.

6. Render Response

    Encodes the response and sends it to the browser.

When a user submits a form, clicks a link, or otherwise generates a new request, the cycle starts anew.

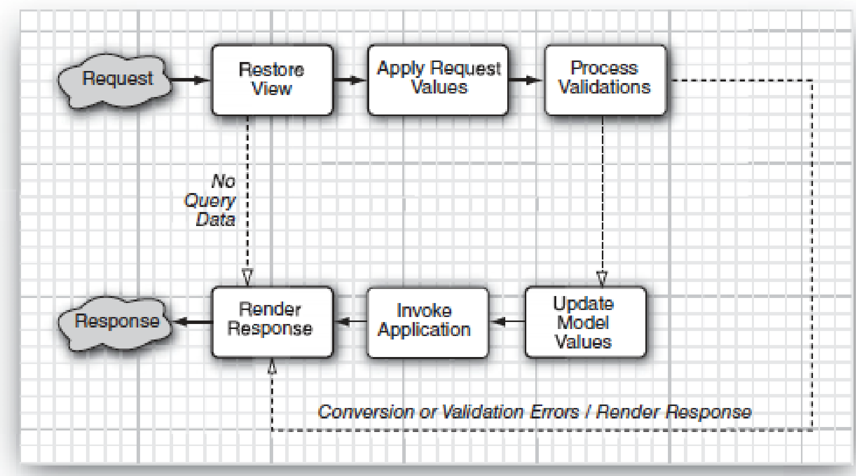


Figure 1–15 The JSF life cycle

Events in the Life Cycle

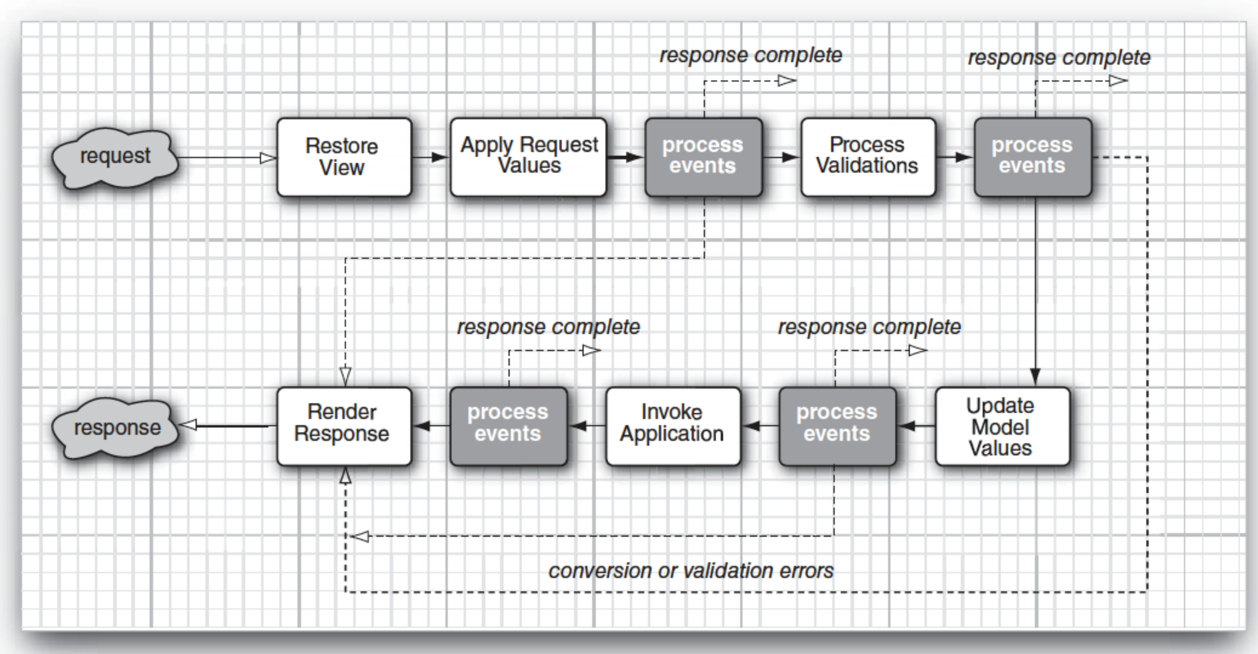


Figure 8–1 Event handling in the JSF life cycle

- **Starting AFTER the Apply Request Values phase, the JSF implementation may create** events and add them to an event queue **DURING EACH phase as shown in Figure 8.1**.
  That is, **After** each phase, the JSF implementation broadcasts queued events to registered listeners.

- Validation occurs only ONCE.

Event listeners can affect the JSF life cycle in one of three ways:

1. Let the life cycle proceed normally.

2. Call the renderResponse method of the FacesContext class to skip the rest of the life cycle up to Render Response. Render Response will render.

3. Call the responseComplete method of the FacesContext class to skip the rest of the life cycle entirely. Render Response will NOT render.

How to Monitor Phases Programmatically via a Phase Listener

Clean and build Eventslab6_Phase_WithFlags *Web App.*

The f:phaseListener (line 14 of JSF page listens to phases the FormBean is called.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4   <html xmlns="http://www.w3.org/1999/xhtml"
5         xmlns:h="http://java.sun.com/jsf/html"
6         xmlns:f="http://java.sun.com/jsf/core">
7       <head>
8           <title>JSF phaseListener Tag</title>
9       </head>
10      <body>
11          <h:form>
12              Enter Text Here : <h:inputText value="#{formBean.name}" />
13              <h:commandButton action="outputPage" value="submit">
14                  <f:phaseListener type="edu.slcc.FormBean"/>
15              </h:commandButton>
16          </h:form>
17      </body>
18  </html>
19
```

```
2
3   import ...5 lines
8   //import javax.faces.event.PhaseListener;
9
10  @Named
11  @RequestScoped
12  public class FormBean extends PhaseListenerASDV
13
14  {
15
16      private static final Logger logger = Logger.getLogger("edu.slcc.formBean");
17      FacesContext context = null;
18      String name = "abc";
19
20      public String getName()
21      {
22          return name;
23      }
24
25      public void setName(String name)
26      {
27          this.name = name;
28      }
29
30  }
31
```

In PhaseListenerASDV

1. line 12 is used to bind the actual JSF phase to the instance var phase.
2. The Logger, writes to the Glassfish server. Good for the debugging, is bettter than souts.
3. We override methods of interface PhaseListener

```
10  public class PhaseListenerASDV implements PhaseListener
11  {
12      private static final String PHASE_PARAMETER = "edu.slcc.phaseListenerASDV.phase";
13      private static final Logger logger = Logger.getLogger("edu.slcc.phases");
14      private static String phase = null;
15
16      public void setPhase(String newValue) {phase = newValue;}
17
18      @Override public void beforePhase(PhaseEvent e){
19          logger.info("BEFORE " + e.getPhaseId()
20                  + " on " + new Date().toString());
21      }
22      @Override public void afterPhase(PhaseEvent e){
23          logger.info("AFTER " + e.getPhaseId()
24                  + " on " + new Date().toString());
25      }
26      @Override public PhaseId getPhaseId()
27      {
28          if (phase == null)
29            {
30              FacesContext context = FacesContext.getCurrentInstance();
31              phase = (String) context.getExternalContext().getInitParameter(
32                      PHASE_PARAMETER);
33            }
34          PhaseId phaseId = PhaseId.ANY_PHASE;
35          if (phase != null)
36            {
37              if ("RESTORE_VIEW".equals(phase))              phaseId = PhaseId.RESTORE_VIEW;
38              else if ("APPLY_REQUEST_VALUES".equals(phase))  phaseId = PhaseId.APPLY_REQUEST_VALUES;
39              else if ("PROCESS_VALIDATIONS".equals(phase))   phaseId = PhaseId.PROCESS_VALIDATIONS;
40              else if ("UPDATE_MODEL_VALUES".equals(phase))   phaseId = PhaseId.UPDATE_MODEL_VALUES;
41              else if ("INVOKE_APPLICATION".equals(phase))    phaseId = PhaseId.INVOKE_APPLICATION;
42              else if ("RENDER_RESPONSE".equals(phase))       phaseId = PhaseId.RENDER_RESPONSE;
43              else if ("ANY_PHASE".equals(phase))             phaseId = PhaseId.ANY_PHASE;
44            }
45          return phaseId;
46      }
```

Attribute Immediate = TRUE

Line 21, calls method submit() of bean testImmediate WITHOUT the rest of the form elements to be sent to the bean

```
7      <h:body>
8          <h1>Test Immediate Components</h1>
9          <h:form>
10             <h:panelGrid columns="2">
11                 <h:outputLabel value="#{msgs.txtName}" ></h:outputLabel>
12                 <h:inputText  value="#{testImmediate.name}" size="20"
13                               required="true"
14                               requiredMessage="#{msgs.txtEmptyName}"
15                               />
16
17                 <h:outputLabel value="#{msgs.txtLang}"></h:outputLabel>

19                 <h:selectOneMenu
20                     value="#{testImmediate.language}" onchange="submit()"
21                     immediate="true"
22                     valueChangeListener="#{testImmediate.languageChanged}">
23                     <f:selectItems value="#{testImmediate.languagesMap}"/>
24                 </h:selectOneMenu>
25
26             </h:panelGrid>
27             <h:commandButton value="Submit" />
28
29             <h:outputText escape="false" style="color: red"
30                           value="#{testImmediate.newData()}" />
31
32         </h:form>
33     </h:body>
34
```

The f:attribute of a Button

It allows us to pass data (value) from the f:attribute to a bean's via the action Listener which can retrieve the component (button)

Clean and build Eventslab6_fAttribute *Web App.*

Lines 15, 23 of JSF page has a action-change listeners. The bean's method changeLocale calls getLanguageCode() which retries the value using the name of the commandLink “languageCode” by using the Map of attributes.

```
10        </h:head>
11        <h:body>
12            <h:form>
13                <h:commandLink immediate="true"
14                               actionListener="#{localeChanger.changeLocale}">
15                    <f:attribute name="languageCode" value="en"/>
16                    <h:graphicImage library="images" name="en_flag.gif"
17                                    style="border: 0px; margin-right: 1em;"/>
18                </h:commandLink>
19
20
21                <h:commandLink immediate="true"
22                               actionListener="#{localeChanger.changeLocale}">
23                    <f:attribute name="languageCode" value="de"/>
24                    <h:graphicImage library="images" name="de_flag.gif"
25                                    style="border: 0px; "/>
26                </h:commandLink>
27
28                <p style="font-style: italic; font-size: 1.3em">#{msgs.indexPageTitle}</p>
29                <h:panelGrid columns="2">
30                    #{msgs.namePrompt}
31                    <h:inputText value="#{user.name}"/>
32                    #{msgs.passwordPrompt}
33                    <h:inputSecret value="#{user.password}"/>
34                    #{msgs.tellUsPrompt}
35                    <h:inputTextarea value="#{user.aboutYourself}" rows="5" cols="35"/>
36                    <h:commandButton value="#{msgs.submitPrompt}" action="thankYou"/>
37                </h:panelGrid>
38            </h:form>
39        </h:body>
```

```
1   package edu.slcc;
2
3   import ...8 lines
11
12  @Named
13  @RequestScoped
14  public class LocaleChanger
15  {
16
17      public void changeLocale(ActionEvent event)
18      {
19          UIComponent component = event.getComponent();
20          String languageCode = getLanguageCode(component);
21          FacesContext.getCurrentInstance()
22                  .getViewRoot().setLocale(new Locale(languageCode));
23      }
24
25      private String getLanguageCode(UIComponent component)
26      {
27          Map<String, Object> attrs = component.getAttributes();
28          return (String) attrs.get("languageCode");
29      }
30  }
31
```

The f:param of a Button

It allows us to pass data (value) from the f:param to a bean's via the action Listener which can retrieve the component (button)

Clean and build Eventslab6_fPAram *Web App.*

Lines 14, 19 of JSF page has a action-change listeners. The bean's method changeLocale calls getLanguageCode() which retrieves the value using the name of the commandLink “languageCode” by using the Map of parameters

```
6      <h:head>
7          <h:outputStylesheet library="css" name="styles.css"/>
8          <title>#{msgs.indexWindowTitle}</title>
9      </h:head>
10     <h:body>
11         <h:form>
12
13             <h:commandLink action="#{localeChanger.changeLocale('en')}">
14                 <h:graphicImage library="images" name="en_flag.gif"
15                                 style="border: 0px; margin-right: 1em;"/>
16             </h:commandLink>
17
18             <h:commandLink action='#{localeChanger.changeLocale("de")}'>
19                 <h:graphicImage library="images" name="de_flag.gif" style="border: 0px"/>
20             </h:commandLink>
21
22
23             <p style="font-style: italic; font-size: 1.3em">#{msgs.indexPageTitle}</p>
24             <h:panelGrid columns="2">
25                 #{msgs.namePrompt}
26                 <h:inputText value="#{user.name}"/>
27                 #{msgs.passwordPrompt}
28                 <h:inputSecret value="#{user.password}"/>
29                 #{msgs.tellUsPrompt}
30                 <h:inputTextarea value="#{user.aboutYourself}" rows="5" cols="35"/>
31                 <h:commandButton value="#{msgs.submitPrompt}" action="thankYou"/>
32             </h:panelGrid>
33         </h:form>
34     </h:body>
```

```
9      @Named
10     @RequestScoped
11     public class LocaleChanger
12     {
13
14         public void changeLocale()
15         {
16             FacesContext context = FacesContext.getCurrentInstance();
17             String languageCode = getLanguageCode(context);
18             context.getViewRoot().setLocale(new Locale(languageCode));
19
20         }
21
22         private String getLanguageCode(FacesContext context)
23         {
               Map<String, String> params = context.getExternalContext().getRequestParameterMap();
25             return params.get("languageCode");
26         }
27     }
```

Passing values to an action method rom JSF page

It allows us to pass data (value) directly from the link to the action method

Clean and build Eventslab6_method *Web App.*

```
11  <h:form>
12
13      <h:commandLink action="#{localeChanger.changeLocale('en')}">
14          <h:graphicImage library="images" name="en_flag.gif"
15                          style="border: 0px; margin-right: 1em;"/>
16      </h:commandLink>
17
18      <h:commandLink action='#{localeChanger.changeLocale("de")}'>
19          <h:graphicImage library="images" name="de_flag.gif" style="border: 0px"/>
20      </h:commandLink>
21
22
23      <p style="font-style: italic; font-size: 1.3em">#{msgs.indexPageTitle}</p>
24      <h:panelGrid columns="2">
25          #{msgs.namePrompt}
26          <h:inputText value="#{user.name}"/>
27          #{msgs.passwordPrompt}
28          <h:inputSecret value="#{user.password}"/>
29          #{msgs.tellUsPrompt}
30          <h:inputTextarea value="#{user.aboutYourself}" rows="5" cols="35"/>
31          <h:commandButton value="#{msgs.submitPrompt}" action="thankYou"/>
32      </h:panelGrid>
33  </h:form>
```

```
3   import ...5 lines
8
9   @Named
10  @RequestScoped
11  public class LocaleChanger
12  {
13
14      public void changeLocale(String languageCode)
15      {
16          FacesContext context = FacesContext.getCurrentInstance();
17          context.getViewRoot().setLocale(new Locale(languageCode));
18
19      }
20  }
21
```

How to create a Tabbed Pane

1. Clean and build Eventslab6_TabbedPane Web App.
2. Observe lines 45 to 48 in the JSF page. All presidents appear in the same Pane ONE at a TIME, via the navigation of bean. Use the technique TO KEEP THE SAME MENU (links, buttons) , , while the content (US presidents change).

```
14          <h1 >US Presidents </h1>
15          <h:form>
16              <h:panelGrid styleClass="tabbedPane" columnClasses="displayPanel">
17
18                      <h:panelGrid columns="4" styleClass="tabbedPaneHeader">
19                          <h:commandLink tabindex="1" title="#{msgs.washingtonTooltip}"
20                                         styleClass="#{tp.washingtonStyle}"
21                                         actionListener="#{tp.washingtonAction}">
22                              #{msgs.washingtonTabText}
23                          </h:commandLink>
24
25                          <h:commandLink tabindex="2" title="#{msgs.lincolnTooltip}"
26                                         styleClass="#{tp.lincolnStyle}"
27                                         actionListener="#{tp.lincolnAction}">
28                              #{msgs.lincolnTabText}
29                          </h:commandLink>
30
31                          <h:commandLink tabindex="3" title="#{msgs.rooseveltTooltip}"
32                                         styleClass="#{tp.rooseveltStyle}"
33                                         actionListener="#{tp.rooseveltAction}">
34                              #{msgs.rooseveltTabText}
35                          </h:commandLink>
36
37                          <h:commandLink tabindex="4" title="#{msgs.jeffersonTooltip}"
38                                         styleClass="#{tp.jeffersonStyle}"
39                                         actionListener="#{tp.jeffersonAction}">
40                              #{msgs.jeffersonTabText}
41                          </h:commandLink>
                        </h:panelGrid>
43                  <!-- Tabbed pane content -->
44
45                  <ui:include src="washington.xhtml"/>
46                  <ui:include src="roosevelt.xhtml"/>
47                  <ui:include src="lincoln.xhtml"/>
48                  <ui:include src="jefferson.xhtml"/>
49              </h:panelGrid>
50          </h:form>
```

Each president in JSF MUST be in composition tag(line 7, line 15)

```
    <?xml version="1.0" encoding="UTF-8"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4   <html xmlns="http://www.w3.org/1999/xhtml"
5         xmlns:ui="http://java.sun.com/jsf/facelets"
6         xmlns:h="http://java.sun.com/jsf/html">
7       <ui:composition>
8           <h:panelGrid columns="2" columnClasses="presidentDiscussionColumn"
9                        rendered="#{tp.jeffersonCurrent}">
10
11              <h:graphicImage library="images" name="jefferson.jpg"/>
12              <span class="tabbedPaneContent">#{msgs.jeffersonDiscussion}</span>
13
14          </h:panelGrid>
15      </ui:composition>
16  </html>
17
```