Chapter 8 Event Handling

1. Introduction

➔ Web applications often needs to respond to user events such as selecting item from the menu or clicking a button.

➔ For that typically event handlers are registers with components. For example

```
<h:selectOneMenu valueChangeListener="#{form.countryChanged}"...>
...
</h:selectOneMenu>
```

Here when user makes selection from menu the jsf implementation invokes the method '`countryChanged`' in bean '`form`' and this event will be listened by the listener.

➔ JSF supports four kinds of events

i. Value Change Events

This events are fired by 'editable value holders' ,such as `h:inputText`, `h:selectOneRadio` and `h:selectManyMenu` etc, when the component's value changes.

ii. Action Events

Action events are fired by action sources such as `h:commandLink` and `h:commandButton` when the button or link is clicked..

iii. Phase Events

This events are routinely fired by JSF life cycle.

iv. System Events

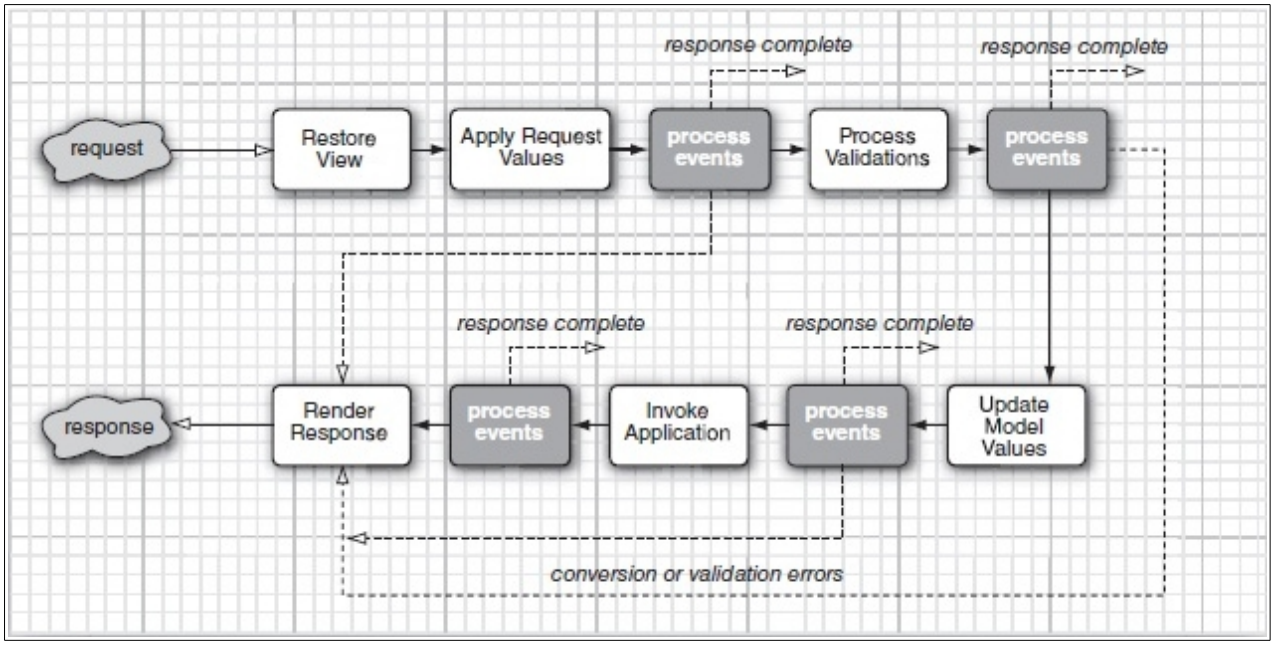
JSF 2.0 added system events. Like its possible to carry out an action before a view or component is rendered.

➔ All JSF events are executed on the server. When an Event Handler is provided in the JSF page, developer tells JSF implementation that he want the event to be handled in the appropriate place in the life cycle when server processes the user input from the page.

2. Events and JSF lifecycle

➔ Requests in JSF are processed by controller servlet which inturn executes JSF lifecycle.

➔ Event handling in JSF life cycle is shown below



➔ As shown in figure above we can see starting from 'Apply Request Values' phase JSF implementation may create events and add them to the event queue during each life cycle phase.

➔ After those phases the JSF implementation broadcasts queued events to registered listeners.

➔ Event listeners can affects JSF lifecycle in three ways

i. Let the life cycle proceed normally.

ii. Call the '`renderResponse`' method of the '`FacesContext`' class to skip the rest of the life cycle upto '`Render Response`' phase.

iii. Call the '`responseComplete`' method of the '`FacesContext`' class to skip the rest of the life cycle.

3. Value Change Events

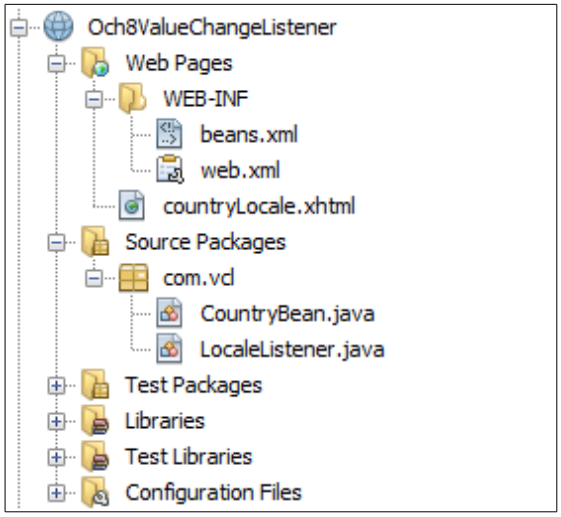
➔ '`ValueChangeEvent`' can be handled in two ways.

a. Method Binding

b. By implementing ValueChangeEvent interface

➔ Example

Lets create 'Och8ValueChangeListener' directory as shown below



CountryBean.java

```
package com.vcl;

import java.io.Serializable;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.Map;
import javax.inject.Named;
import javax.enterprise.context.SessionScoped;
import javax.faces.event.ValueChangeEvent;

@Named(value = "country")
@SessionScoped
public class CountryBean implements Serializable{

    public CountryBean() {
```

```
    }

    private static Map<String,String> countries;
    private String locale="Hi";

    public Map<String,String> getCountryInMap() {
            return this.countries;
      }

    public String getLocale() {
        return locale;
    }

    public void setLocale(String locale) {
        this.locale = locale;
    }

    static {
            countries = new LinkedHashMap<String,String>();

            countries.put("India", "Hi");
            countries.put("Nepal", "Ne");
            countries.put("China", "Ma");
            countries.put("Bhutan", "Bh");
    }
    public void localeForCountry(ValueChangeEvent e)
    {
        locale = e.getNewValue().toString();
    }

}
```

LocaleListener.java

```
package com.vcl;


import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ValueChangeEvent;
import javax.faces.event.ValueChangeListener;

public class LocaleListener implements ValueChangeListener{

    @Override
    public void processValueChange(ValueChangeEvent event)
                            throws AbortProcessingException
    {
        CountryBean country = (CountryBean) FacesContext.getCurrentInstance()
                              .getExternalContext().getSessionMap().get("country");

        country.setLocale(event.getNewValue().toString());

    }
}
```

countryLocale.xhtml

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
```

```
        <title>Country Locale</title>
    </h:head>
    <h:body>
        <h:form>

            <h:panelGrid>
            <ui:remove><!--this method binding code has been put inside ui:remove tag-->
            Select Country Locale(method):
            <h:inputText id="country" value="#{country.locale}" size="20" />
            <h:selectOneMenu value="#{country.locale}" onchange="submit()"
                             valueChangeListener="#{country.localeForCountry}">
                <f:selectItems value="#{country.countryInMap}" />
            </h:selectOneMenu>
            </ui:remove>

            Select Country Locale(listener):
            <h:inputText id="country" value="#{country.locale}"/>
            <h:selectOneMenu value="#{country.locale}" onchange="submit()"
                             valueChangeListener="com.vcl.LocaleListener">
                <f:selectItems value="#{country.countryInMap}"/>
            </h:selectOneMenu>

            </h:panelGrid>

        </h:form>
    </h:body>
</html>
```

Note : In countryLocal.xhtml file, both 'method binding' and 'ValueEventListener implementing Listener' both have been used.

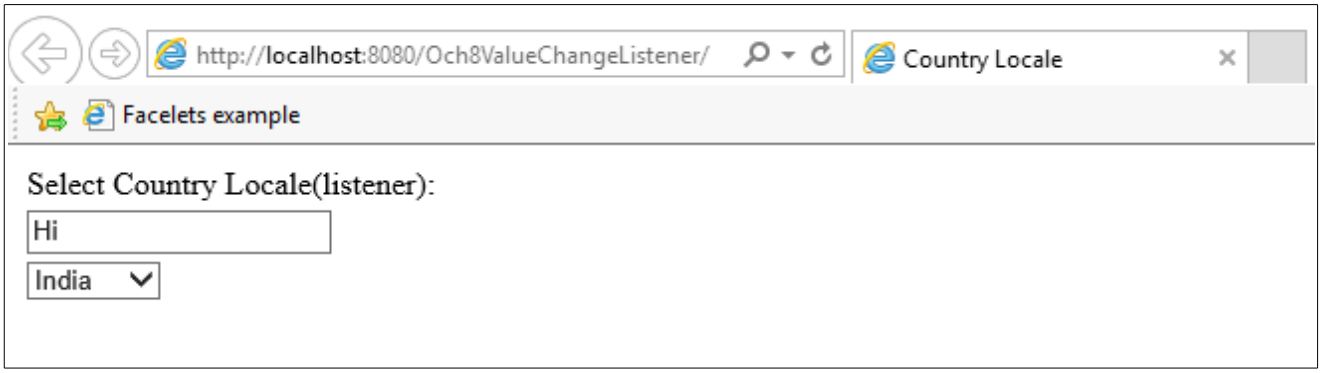
web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/countryLocale.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Output

1. Listener

countryLocale.xhtml

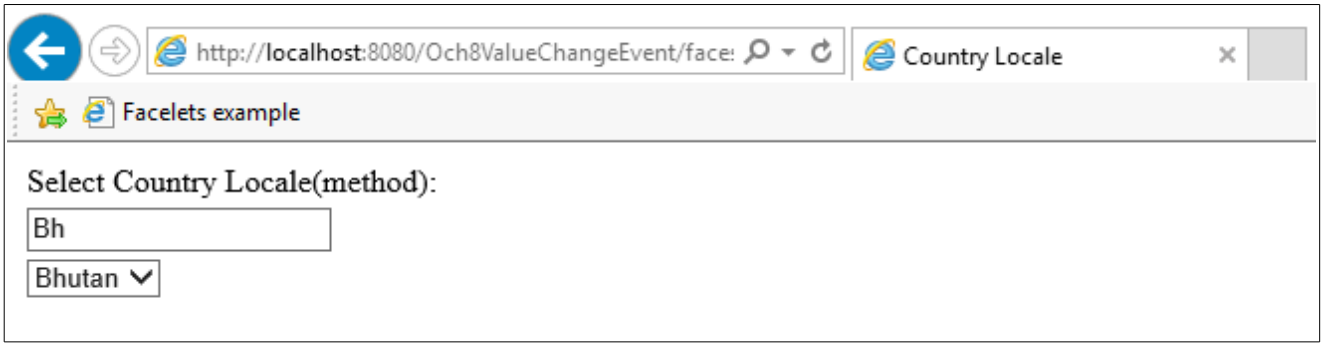


countryLocale.xhtml

2. Methods

countryLocale.xhtml

countryLocale.xhtml



4. Action Events

➔ Action Events are fired by buttons and links.

➔ Action Events are fired during the 'Invoke Application' phase near end of the life cycle.

➔ One can add 'actionListener' to an action source like this

```
<h:commandLink actionListener="#{bean.linkActivated}">
...
</h:commandLink>
```

➔ Command components submit request when they are activated hence no need to use '`onchange`' to force submit as we have done with ValueChangeListener.

➔ There is difference between 'action' and 'actionListener'. 'action' are designed for business logic and participate in navigation handling. On the other hand 'actionListener' typically perform user interface logic and do not participate in navigation handling. JSF implementation always invokes action listeners before actions.

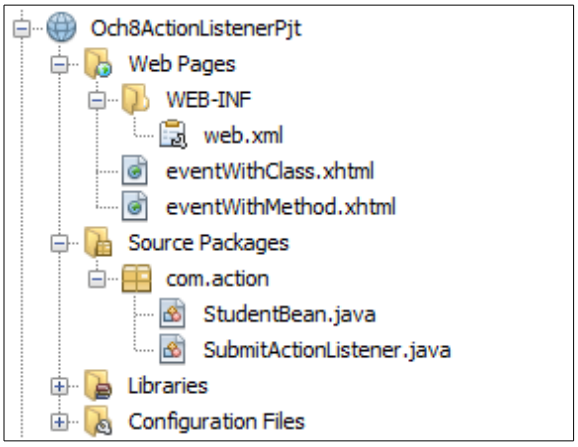
➔ ActionListener too handled in two ways

i. Method Binding

ii. Class implementing ActionListener interface

Below example elobarate both of them.

➔ Lets create 'Och8ActionListenerPjt' directory as shown below



StudentBean.java

```
package com.action;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;

@ManagedBean(name = "studentBean", eager = true)
@RequestScoped
public class StudentBean {
      public void performAction(ActionEvent event)
            throws AbortProcessingException {
            System.out.println("Form Id by ActionListener
```

```
            attribute:"+event.getComponent().getParent().getId());
    }
      public String submitActionForMethod()
            throws AbortProcessingException {
            System.out.println("Action Submitted for ActionListener attribute.");
            return "eventWithMethod";
    }
      public String submitActionForClass()
            throws AbortProcessingException {
            System.out.println("Action Submitted for ActionListener class.");
            return "eventWithClass";
    }
}
```

SubmitActionListener.java

```
package com.action;

import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;
import javax.faces.event.ActionListener;

public class SubmitActionListener implements ActionListener {
      @Override
      public void processAction(ActionEvent event)
                  throws AbortProcessingException {
            System.out.println("Form Id by ActionListener
                              class:"+event.getComponent().getParent().getId());
      }
}
```

eventWithMethod.xhtml

```
<html lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>ActionListener with Method in JSF 2</title>
    </h:head>
    <h:body>
       <h3>ActionListener with Method in JSF 2</h3>
            <h:form id="studentForm">
           <h:commandButton id="commandButton" action="#{studentBean.submitActionForMethod}"
               value="submit" actionListener="#{studentBean.performAction}"/>
       </h:form>
    </h:body></html>
```

eventWithClass.xhtml

```
<html lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>ActionListener with Class in JSF 2</title>
    </h:head>
    <h:body>
       <h3>ActionListener with Class in JSF 2</h3>
      <h:form id="studentForm">
            <h:commandButton id="commandButton" action="#{studentBean.submitActionForClass}"
             value="submit" actionListener="#{studentBean.performAction}"/>
       </h:form>
```

```
    </h:body>
</html>
```

Note: Here we can see that to use action listener class which implements ActionListener interface we should use tag <f:actionListener>.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/eventWithMethod.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Note : Here we can change <welcome-file> tag value for eventWithMethod.xhtml and eventWithClass.xhtml.

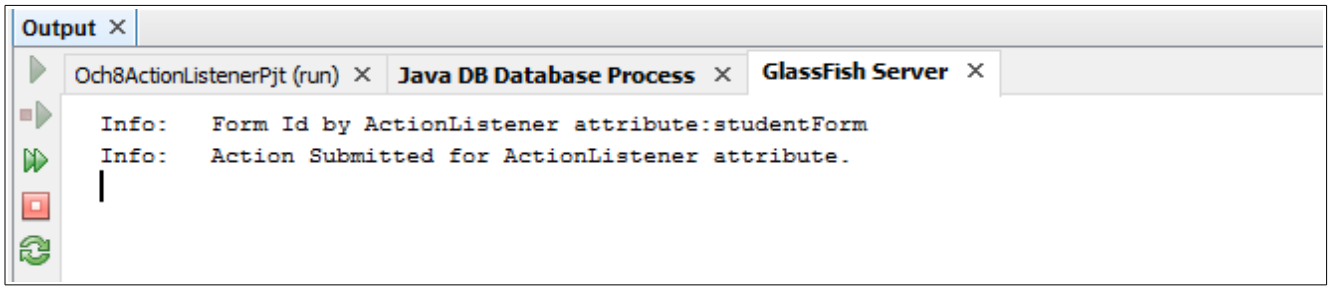
Output

1. Event with method

eventWithMethod.xhtml

eventWithMethod.xhtml(after clicking submit)

Server Console

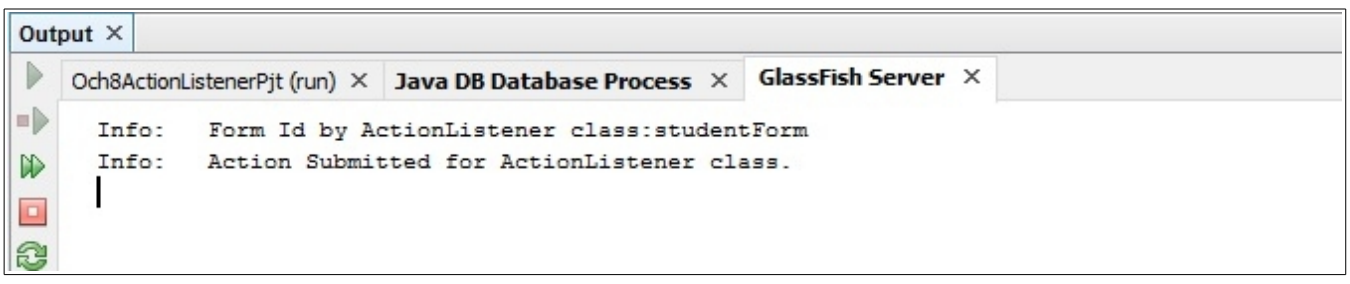


2. Event with Class

eventWithClass.xhtml

eventWithClass.xhtml(after clicking submit button)

Server Console



5. Event Listener Tags

➔ Till now we have used 'valueChangeListener' and 'actionListener' attributes to define listeners. But we can also use <f:valueChangeListener> and <f:actionListener> tags which yields the same results. Example is shown below

```
<h:commandButton id="commandButton" action="#{studentBean.submitActionForClass}"
         actionListener="com.action.SubmitActionListener" value="submit"/>
```

In place of this we could have

```
<h:commandButton id="commandButton" action="#{studentBean.submitActionForClass}"
                              value="submit"/>
   <f:actionListener type="com.action.SubmitActionListener"/>
</h:commandButton>
```

➔ Tags have advantage over attribute. Tags lets multiple listener attached to the single component.

➔ Tags should have only class as 'type' which implements corresponding interface. Method binding is not possible with tags. On the other hand attributes can have both method binding and class.

➔ If there are multiple listeners (defined through both attribute and tags) then they are invoked in the following order.

i. The listener specified by the listener attribute.

ii. Listener specified by the tags in the order they are declared.

6. Immediate Components

➔ We know 'Value Change Events' are fired after 'Process Validation' Phase and 'action events' are fired after 'Invoke Application' Phase. This behavior is preferred because one want to be notified of value change only when they are valid and action events only after all submitted values have been transmitted to the model.

➔ But to bypass validation i.e. when 'immediate' attribute is set to 'true' conversion and validation occurs earlier than usual,

after 'Apply Request Value' phase.

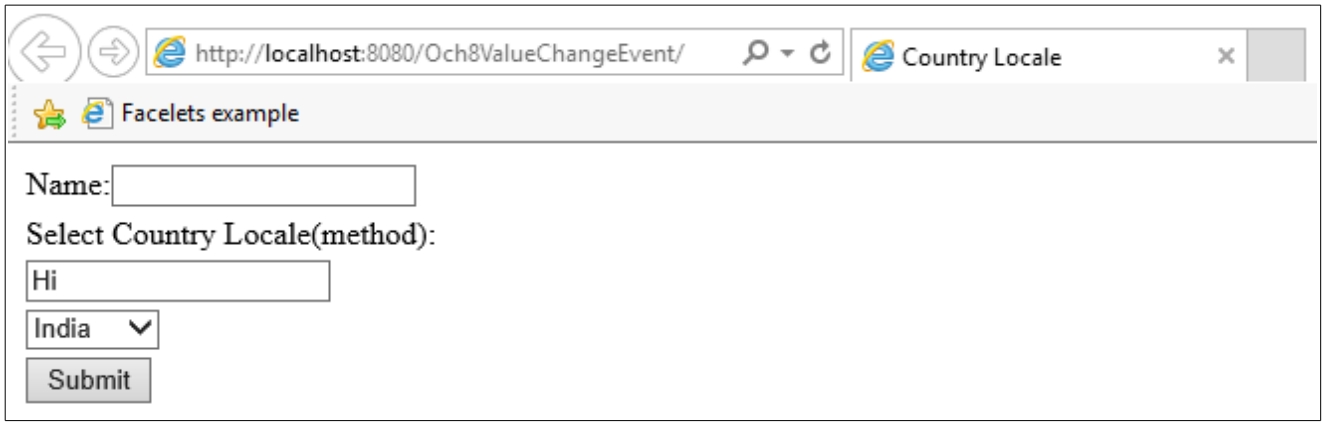
➔ Hence we have Immediate command components which fires events earlier than usual i.e. after 'Apply Request Value' phase. This kicks in the navigation handler which directly goes to 'Render Response' bypassing rest of the life cycle.

➔ Ex In a project `Och8ValueChangeListener` if we change `countryLocal.xhtml` and insert an input field name as shown below

```
<h:panelGrid>
<h:panelGroup>
      <h:outputLable value="Name:"/>
      <h:inputText value="#{country.name}" required="true" />
</h:panelGroup>
Select Country Locale(method):
<h:inputText id="country" value="#{country.locale}" size="20" />
<h:selectOneMenu value="#{country.locale}" onchange="submit()"
                 valueChangeListener="#{country.localeForCountry}">
    <f:selectItems value="#{country.countryInMap}" />
</h:selectOneMenu>
<h:commandButton value="Submit" action="submit()"/>
</h:panelGrid>
```

Note : Above we can see input field 'name' is a required field. The output for countryLocale.xhtml is shown below

countryLocale.xhtml



When we change the country from 'India' to anything else without filling in the 'Name' field then there will be error. This is happening because 'country.name' is a required field and when country is changed it gets submitted. As shown below

countryLocale.xhtml

➔ We want validation to kicks in when submit button activated, but not when country is changed. So solve this we have Immediate Components.

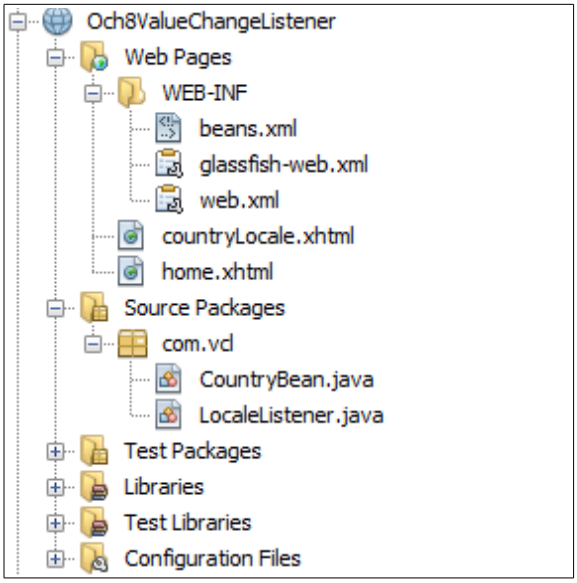
➔ We make Country menu an immediate component by adding attribute `immediate="true"` to `<h:selectOneMenu>` as shown below

```
<h:selectOneMenu value="#{country.locale}" onchange="submit() immediate="true"
                 "valueChangeListener="#{country.localeForCountry}">
<f:selectItems value="#{country.countryInMap}"/>
</h:selectOneMenu>
```

and also call method `renderResponse()` inside `localeForCountry(ValueChangeEvent e)` method of `CountryBean` class as shown below

```
public void localeForCountry(ValueChangeEvent e)
    {
        FacesContext context = FacesContext.getCurrentInstance();
        locale = e.getNewValue().toString();
        context.renderResponse();
    }
```

➔ Changed and new added files of project 'Och8ValueChangeListener' is shown below



CountryBean.java

```
package com.vcl;


import java.io.Serializable;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.Map;
import javax.inject.Named;
import javax.enterprise.context.SessionScoped;
import javax.faces.event.ValueChangeEvent;

@Named(value = "country")
@SessionScoped
public class CountryBean implements Serializable{

    public CountryBean() {
    }

    private static Map<String,String> countries;
    private String locale="Hi";
    private String name;

    public Map<String,String> getCountryInMap() {
            return this.countries;
      }

    public String getLocale() {
        return locale;
    }

    public void setLocale(String locale) {
        this.locale = locale;
    }

    public String getName() {
         return name;
     }

     public void setName(String name) {
         this.locale = name;
```

```
    }

    static {
            countries = new LinkedHashMap<String,String>();
            countries.put("India", "Hi");
            countries.put("Nepal", "Ne");
            countries.put("China", "Ma");
            countries.put("Bhutan", "Bh");
    }
    public void localeForCountry(ValueChangeEvent e)
    {
        FacesContext context = FacesContext.getCurrentInstance();
        locale = e.getNewValue().toString();
        context.renderResponse();
    }
}
```

countryLocale.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Country Locale</title>
    </h:head>
    <h:body>
        <h:form>

            <h:panelGrid>
            <h:panelGroup>
                  <h:outputLable value="Name:"/>
                  <h:inputText value="#{country.name}" required="true" />
            </h:panelGroup>
            Select Country Locale(method):
            <h:inputText id="country" value="#{country.locale}" size="20" />
            <h:selectOneMenu value="#{country.locale}" onchange="submit()" immediate="true"
                             valueChangeListener="#{country.localeForCountry}">
                <f:selectItems value="#{country.countryInMap}" />
            </h:selectOneMenu>
            <h:commandButton value="Submit" action="submit()"/>
            </h:panelGrid>


            <ui:remove>
            Select Country Locale(listener):
            <h:inputText id="country" value="#{country.locale}"/>
            <h:selectOneMenu value="#{country.locale}" onchange="submit()"
                             valueChangeListener="com.vcl.LocaleListener">
                <f:selectItems value="#{country.countryInMap}"/>
            </h:selectOneMenu>
            </ui:remove>

        </h:form>
    </h:body>
</html>
```

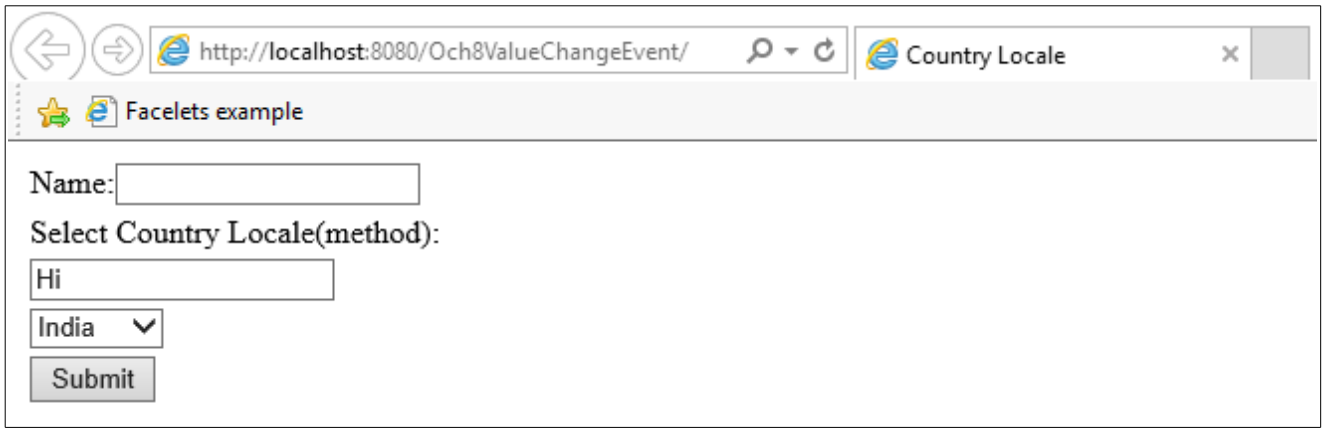
home.xhtml(new file)

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
```

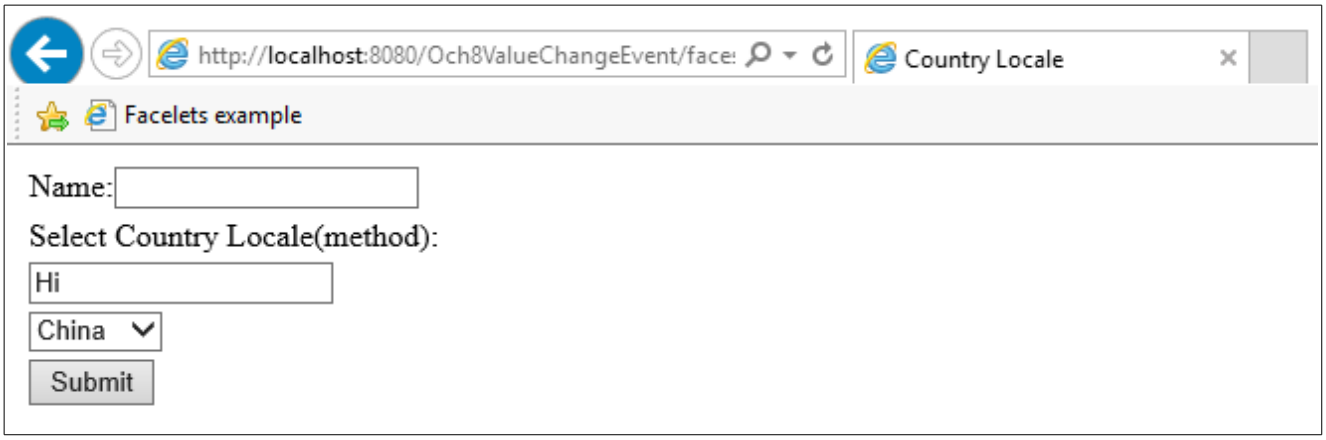
```
    <h:head>
        <title>Home Page</title>
    </h:head>
    <h:body>
        <h:panelGrid>
        <h:outputText value="Hello #{country.name}"/>
        <h:outputText value="Your Country's locale is '#{country.locale}'"/>
        </h:panelGrid>
    </h:body>
</html>
```

Output

countryLocale.xhtml



countryLocale.xhtml(country is changed : No errors)



countryLocale.xhtml(Error when submitted without filling name field)

home.xhtml(When all inputs are valid)

Note: Use of 'immediate' attribute in command components bypass all validation as seen in previous chapter.

7. Phase Events

➔ JSF implementation fires events called phase events, before and after of each life cycle phase. These events are handled by phase listeners.

➔ Unlike Value Change and Action Listeners that are attached to the components, phase listeners are attache to the view root.

➔ One can specify the phase listener for an individual component using tag

<f:phaseListener type="com.phase.PhaseTracker"/>

➔ Alternatively one can specify global phase listener in the faces cofiguration file as shown below

```
<faces-config>
      <lifecycle>
            <phase-listener>com.corejsf.PhaseTracker</phase-listener>
      </lifecycle>
</faces-config>
```

The above code specify only one listener. One can specify as many as one want. Listeners are invoked in the order they have been specified.

➔ One can implement phase listener by implementing interface '`PhaseListener`' from javax.faces.event package. This inteface have three methods

```
PhaseId getPhaseId()
void afterPhase(PhaseEvent)
void beforePhase(PhaseEvent)
```

`getPhaseId()` method tells JSF impementation when to deliever phase events to the listener. For example if `getPhaseId()` returns `PhaseId.APPLY_REQUEST_VALUES`. In that case, `beforePhase()` and `afterPhase()` would be called once per life cycle before and after 'Apply Request Values' phase. One can also specify phase id as PhaseId.ANY_PHASE which really means all phases. In that case `beforePhase()` and `afterPhase()` methods will be called six times per life cycle: once each for each life cycle phase.

➔ Alternatively one can enclose JSF page in an `f:view` tag with `beforePhase` or `afterPhase` attributes. These attributes must point to method with signature void `listener(javax.faces.event.PhaseEvent)`. These are invoked before every phase except for "Restore View" phase.

```
<f:view beforePhase="#{backingBean.beforeListener}">
<h:head>
...
</f:view>
```

➔ Phase listeners are useful for debugging tools. Before JSF 2.0 they offered only mechanism for writing custom components. Its better to prefer using System Events over phase events.

8. System Events

➔ JSF 2.0 introduces fine grained notification system in which individual components as well as JSF implementation notify listeners of many potentially interesting events.

➔ JSF system events are listed below

System Events

Event Class	Description	Source Type
PostConstructApplicationEvent PreDestroyApplicationEvent	Immediately after the application has started; immediately before it is about to be shut down	Application
PostAddToViewEvent PreRemoveFromViewEvent	After a component has been added to the view tree; before it is about to be removed	UIComponent
PostRestoreStateEvent	After the state of a component has been restored	UIComponent
PreValidateEvent PostValidateEvent	Before and after a component is validated	UIComponent
PreRenderViewEvent	Before the view root is about to be rendered	UIViewRoot
PreRenderComponentEvent	Before a component is about to be rendered	UIComponent
PostConstructViewMapEvent PreDestroyViewMapEvent	After the root component has constructed the view scope map; when the view map is cleared[a]	UIViewRoot
PostConstructCustomScopeEvent PreDestroyCustomScopeEvent	After a custom scope has been constructed; before it is about to be destroyed	ScopeContext
ExceptionQueuedEvent	After an exception has been queued	ExceptionQueuedEvent-Context

a. To monitor the life cycle of the application, session, and request maps, use a ServletContextListener, ServletHttpSessionListener, or ServletRequestListener.

8.1. Multiple Component Validation

➔ In chapter 7 we have seen if there are multiple components like Date's Day, Month and Year and we want all of them to

be validated then they could be validated separately. As JSF provide validation for individual compnent not for group of component.

➔ But with System Events its possible to validate all components through single validator. It can be done through 'postValidate' event as shown below

```
<f:event type="postValidate" listener="#{bb.validateDate}"/>
```

8.2. Making Decisions Before Rendering the View

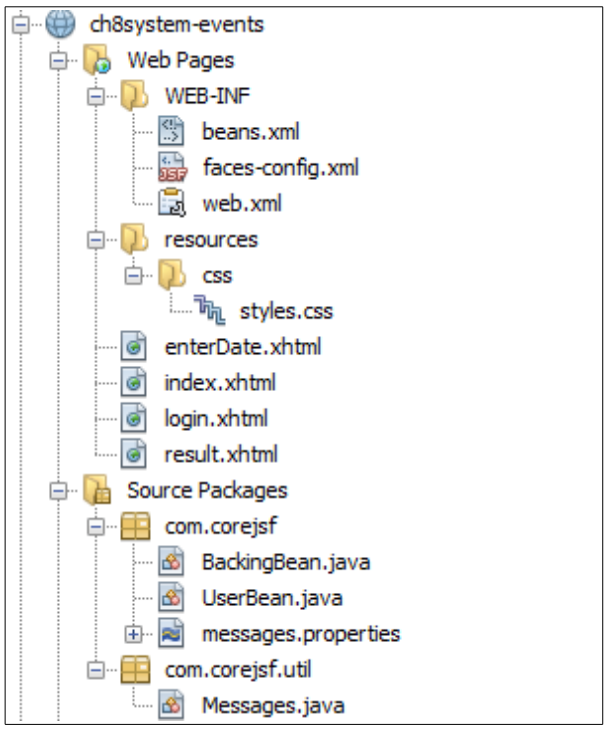
➔ Sometime one wants to be notified before view is rendered. For example, to load data, make changes to the components on the page, or to conditionally navigate to another page.

➔ For example, one wants to be sure that user has been logged in before showing a particular page. For that enclose the view inside <f:view> tag and attach a listener.

```
<f:view>
<f:event type="preRenderView" listener="#{user.checkLogin}"/>
<h:head>
<title>...</title>
</h:head>
<h:body>
...
</h:body>
</f:view>
```

➔ Example

Lets create directory for project 'ch8system-events' as shown below



UserBean.java

```
package com.corejsf;
```

```
import java.io.Serializable;

import javax.faces.application.ConfigurableNavigationHandler;
import javax.inject.Named;
   // or import javax.faces.bean.ManagedBean;
import javax.enterprise.context.SessionScoped;
   // or import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ComponentSystemEvent;

@Named("user") // or @ManagedBean(name="user")
@SessionScoped
public class UserBean implements Serializable {
   private String name = "";
   private String password;
   private boolean loggedIn;

   public String getName() { return name; }
   public void setName(String newValue) { name = newValue; }

   public String getPassword() { return password; }
   public void setPassword(String newValue) { password = newValue; }

   public boolean isLoggedIn() { return loggedIn; }

   public String login()  {
      loggedIn = true;
      return "index";
   }

   public String logout() {
      loggedIn = false;
      return "login";
   }

   public void checkLogin(ComponentSystemEvent event) {
      if (!loggedIn) {
         FacesContext context = FacesContext.getCurrentInstance();
         ConfigurableNavigationHandler handler = (ConfigurableNavigationHandler)
            context.getApplication().getNavigationHandler();
         handler.performNavigation("login");
      }
   }
}
```

BackingBean.java

```
package com.corejsf;


import java.io.Serializable;

import javax.faces.application.FacesMessage;
import javax.inject.Named;
   // or import javax.faces.bean.ManagedBean;
import javax.enterprise.context.SessionScoped;
   // or import javax.faces.bean.SessionScoped;
import javax.faces.component.UIComponent;
import javax.faces.component.UIForm;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.event.ComponentSystemEvent;
import javax.faces.validator.ValidatorException;

@Named("bb") // or @ManagedBean(name="bb")
@SessionScoped
```

```
public class BackingBean implements Serializable {
   private int day;
   private int month;
   private int year;

   public int getDay() { return day; }
   public void setDay(int newValue) { day = newValue; }

   public int getMonth() { return month; }
   public void setMonth(int newValue) { month = newValue; }

   public int getYear() { return year; }
   public void setYear(int newValue) { year = newValue; }

   public void validateDate(ComponentSystemEvent event) {
      UIComponent source = event.getComponent();
      UIInput dayInput = (UIInput) source.findComponent("day");
      UIInput monthInput = (UIInput) source.findComponent("month");
      UIInput yearInput = (UIInput) source.findComponent("year");
      int d = ((Integer) dayInput.getLocalValue()).intValue();
      int m = ((Integer) monthInput.getLocalValue()).intValue();
      int y = ((Integer) yearInput.getLocalValue()).intValue();
      if (!isValidDate(d, m, y)) {
         FacesMessage message = com.corejsf.util.Messages.getMessage(
            "com.corejsf.messages", "invalidDate", null);
         message.setSeverity(FacesMessage.SEVERITY_ERROR);
         FacesContext context = FacesContext.getCurrentInstance();
         context.addMessage(source.getClientId(), message);
         context.renderResponse();
      }
   }

   private static boolean isValidDate(int d, int m, int y) {
      if (d < 1 || m < 1 || m > 12) return false;
      if (m == 2) {
         if (isLeapYear(y)) return d <= 29;
         else return d <= 28;
      }
      else if (m == 4 || m == 6 || m == 9 || m == 11)
         return d <= 30;
      else
         return d <= 31;
   }

   private static boolean isLeapYear(int y) {
      return y % 4 == 0 && (y % 400 == 0 || y % 100 != 0);
   }
}
```

Messages.java

```
package com.corejsf.util;


import java.text.MessageFormat;
import java.util.Locale;
import java.util.MissingResourceException;
import java.util.ResourceBundle;
import javax.faces.application.Application;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIViewRoot;
import javax.faces.context.FacesContext;

public class Messages {
   public static FacesMessage getMessage(String bundleName, String resourceId,
      Object[] params) {
      FacesContext context = FacesContext.getCurrentInstance();
```

```
      Application app = context.getApplication();
      String appBundle = app.getMessageBundle();
      Locale locale = getLocale(context);
      ClassLoader loader = getClassLoader();
      String summary = getString(appBundle, bundleName, resourceId,
         locale, loader, params);
      if (summary == null) summary = "???" + resourceId + "???";
      String detail = getString(appBundle, bundleName, resourceId + "_detail",
         locale, loader, params);
      return new FacesMessage(summary, detail);
   }

   public static String getString(String bundle, String resourceId,
         Object[] params) {
      FacesContext context = FacesContext.getCurrentInstance();
      Application app = context.getApplication();
      String appBundle = app.getMessageBundle();
      Locale locale = getLocale(context);
      ClassLoader loader = getClassLoader();
      return getString(appBundle, bundle, resourceId, locale, loader, params);
   }

   public static String getString(String bundle1, String bundle2,
         String resourceId, Locale locale, ClassLoader loader,
         Object[] params) {
      String resource = null;
      ResourceBundle bundle;

      if (bundle1 != null) {
         bundle = ResourceBundle.getBundle(bundle1, locale, loader);
         if (bundle != null)
            try {
               resource = bundle.getString(resourceId);
            } catch (MissingResourceException ex) {
            }
      }

      if (resource == null) {
         bundle = ResourceBundle.getBundle(bundle2, locale, loader);
         if (bundle != null)
            try {
               resource = bundle.getString(resourceId);
            } catch (MissingResourceException ex) {
            }
      }

      if (resource == null) return null; // no match
      if (params == null) return resource;

      MessageFormat formatter = new MessageFormat(resource, locale);
      return formatter.format(params);
   }

   public static Locale getLocale(FacesContext context) {
      Locale locale = null;
      UIViewRoot viewRoot = context.getViewRoot();
      if (viewRoot != null) locale = viewRoot.getLocale();
      if (locale == null) locale = Locale.getDefault();
      return locale;
   }

   public static ClassLoader getClassLoader() {
      ClassLoader loader = Thread.currentThread().getContextClassLoader();
      if (loader == null) loader = ClassLoader.getSystemClassLoader();
      return loader;
   }
}
```

messages.properties

```
invalidDate=Invalid date.

invalidDate_detail=The entered date is not valid.
title=Validating The Relationship Between Components
enterDate=Please enter a date.
day=Day
month=Month
year=Year
submit=Submit
validDate=You entered a valid date.
back=Back
```

style.css

```
.errorMessage {

   color:red;
}
```

login.xhtml

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
   <h:head>
      <title>Welcome</title>
   </h:head>
   <h:body>
      <h:form>
         <h3>Please enter your name and password.</h3>
         <table>
            <tr>
               <td>Name:</td>
               <td><h:inputText id="name" value="#{user.name}" /></td>
            </tr>
            <tr>
               <td>Password:</td>
               <td><h:inputSecret value="#{user.password}" required="true" /></td>
            </tr>
         </table>
         <p><h:commandButton value="Login" action="#{user.login}" /></p>
      </h:form>
   </h:body>
</html>
```

index.xhtml

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
   <f:view>
      <f:event type="preRenderView" listener="#{user.checkLogin}"/>
      <h:head>
         <title>Welcome</title>
      </h:head>
      <h:body>
         <h3><h:outputText value="Welcome to JavaServer Faces, #{user.name}!" /></h3>
```

```
         <h:form>
            <h:commandButton value="Logout" action="#{user.logout}" />
            <h:commandButton value="Continue" action="enterDate" />
         </h:form>
      </h:body>
   </f:view>
</html>
```

enterDate.xhtml

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
   <h:head>
      <h:outputStylesheet library="css" name="styles.css"/>
      <title>#{msgs.title}</title>
   </h:head>
   <h:body>
      <h:form>
         <h1>#{msgs.enterDate}</h1>
         <h:panelGrid id="date" columns="2">
            <f:event type="postValidate" listener="#{bb.validateDate}"/>
            #{msgs.day}
            <h:inputText id="day" value="#{bb.day}" size="2"
                         required="true"/>

            #{msgs.month}
            <h:inputText id="month" value="#{bb.month}"
                         size="2" required="true"/>

            #{msgs.year}
            <h:inputText id="year" value="#{bb.year}"
                         size="4" required="true"/>
         </h:panelGrid>
         <h:message for="date" styleClass="errorMessage"/>
         <br/>
         <h:commandButton value="#{msgs.submit}" action="result"/>
         <h:commandButton value="#{msgs.back}" action="index" immediate="true"/>
      </h:form>
   </h:body>
</html>
```

result.xhtml

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
   <h:head>
      <title>#{msgs.title}</title>
   </h:head>
   <h:body>
      <h:form>
         <p>#{msgs.validDate}</p>
         <p><h:commandButton value="#{msgs.back}" action="index"/></p>
      </h:form>
   </h:body>
</html>
```

web.xhtml

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   version="2.5">
   <servlet>
      <servlet-name>Faces Servlet</servlet-name>
      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>/faces/*</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
      <welcome-file>faces/index.xhtml</welcome-file>
   </welcome-file-list>
   <context-param>
      <param-name>javax.faces.PROJECT_STAGE</param-name>
      <param-value>Development</param-value>
   </context-param>
</web-app>
```

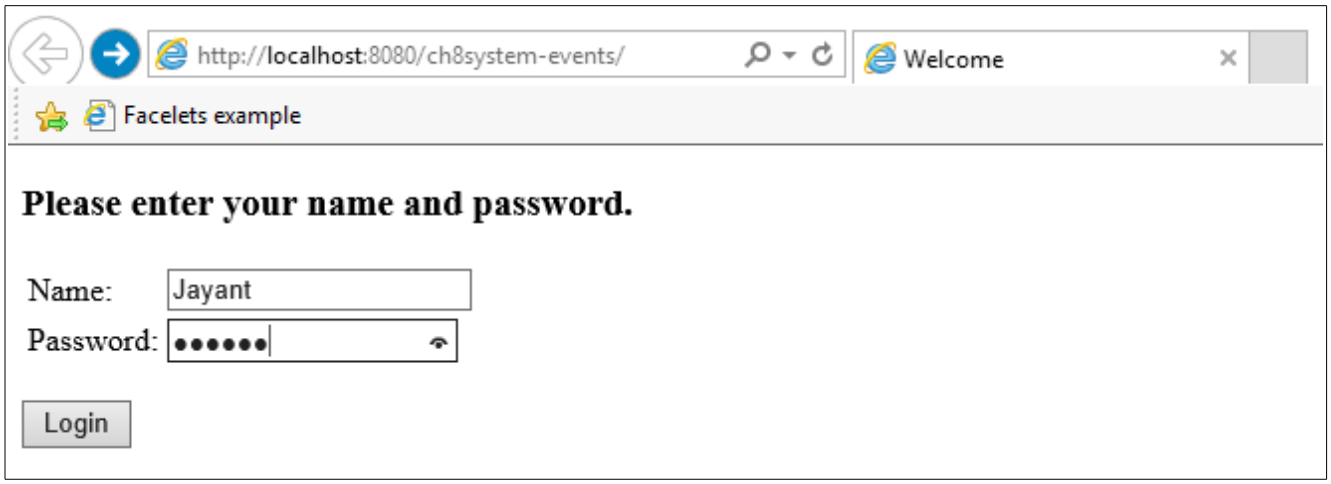
faces-config.xhtml

```
<?xml version="1.0"?>

<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
   version="2.0">
   <application>
      <resource-bundle>
         <base-name>com.corejsf.messages</base-name>
         <var>msgs</var>
      </resource-bundle>
   </application>
</faces-config>
```

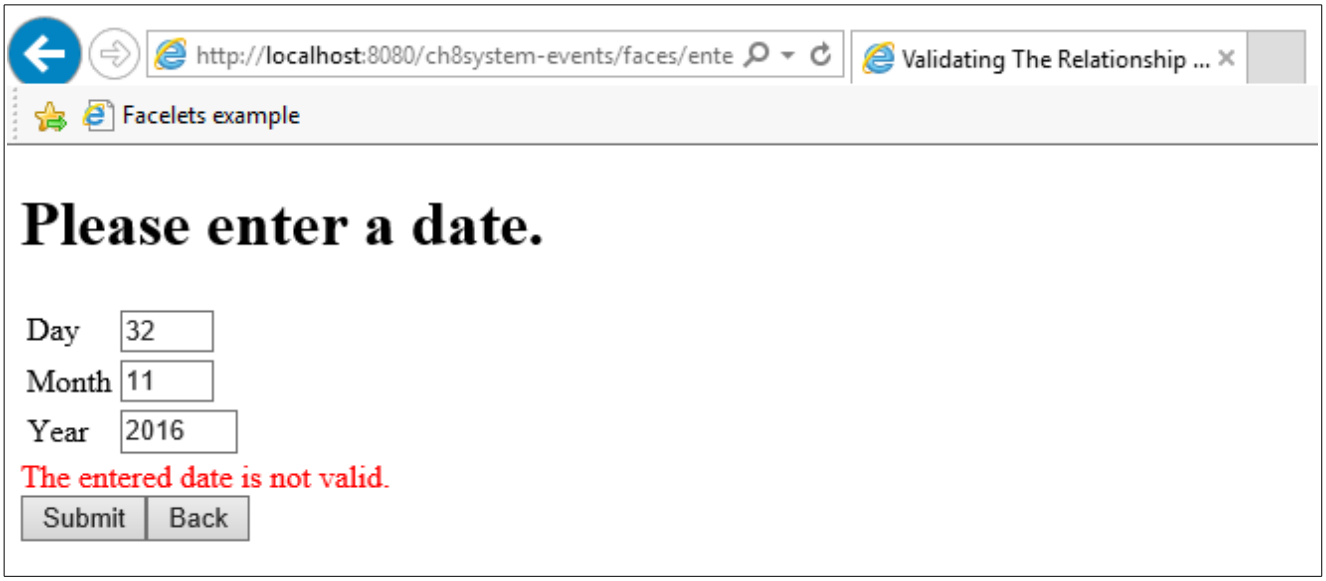
Output

login.xhtml

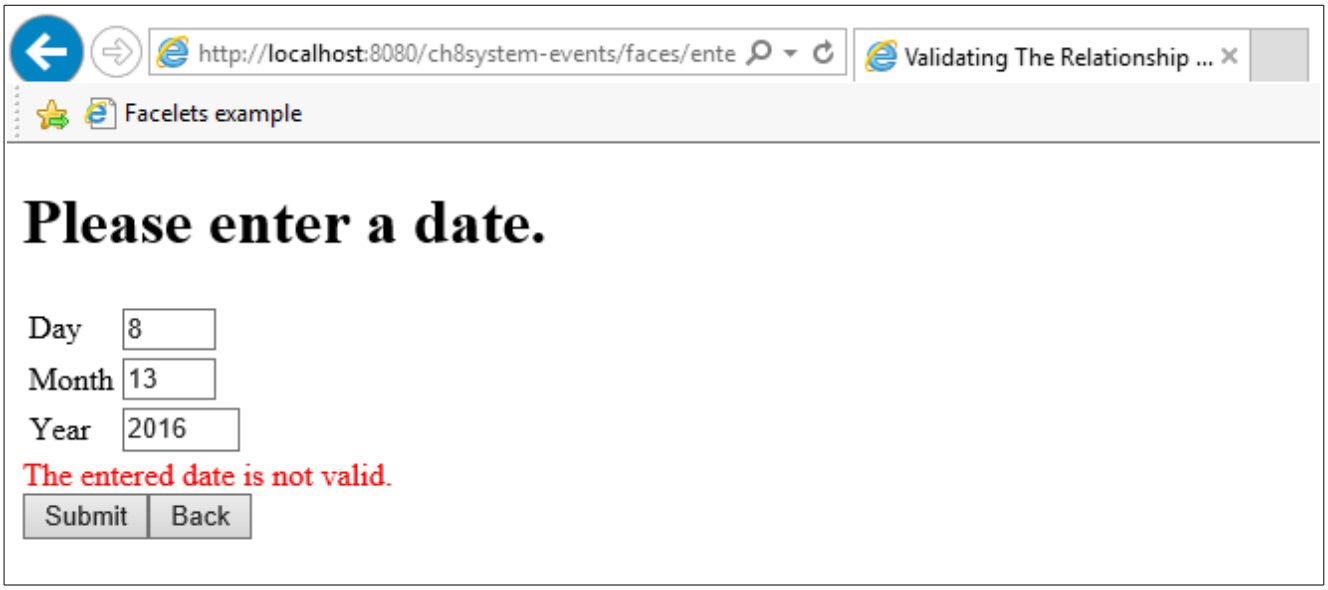


index.xhtml

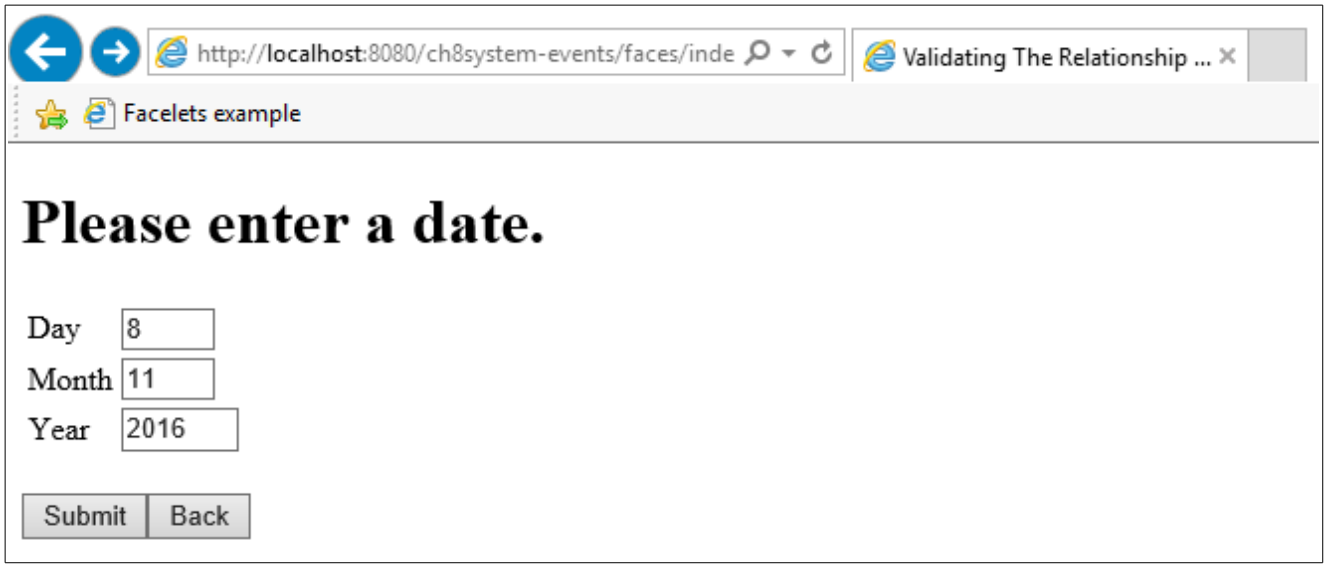
enterDate..xhtml(day is invalid)



enterDate.xhtml(Month is invalid)



enterDate.xhtml(Valid Date)



result.xhtml