Chapter 1 Getting Started

1. Beans

- A bean must have name and scope.
- Ex

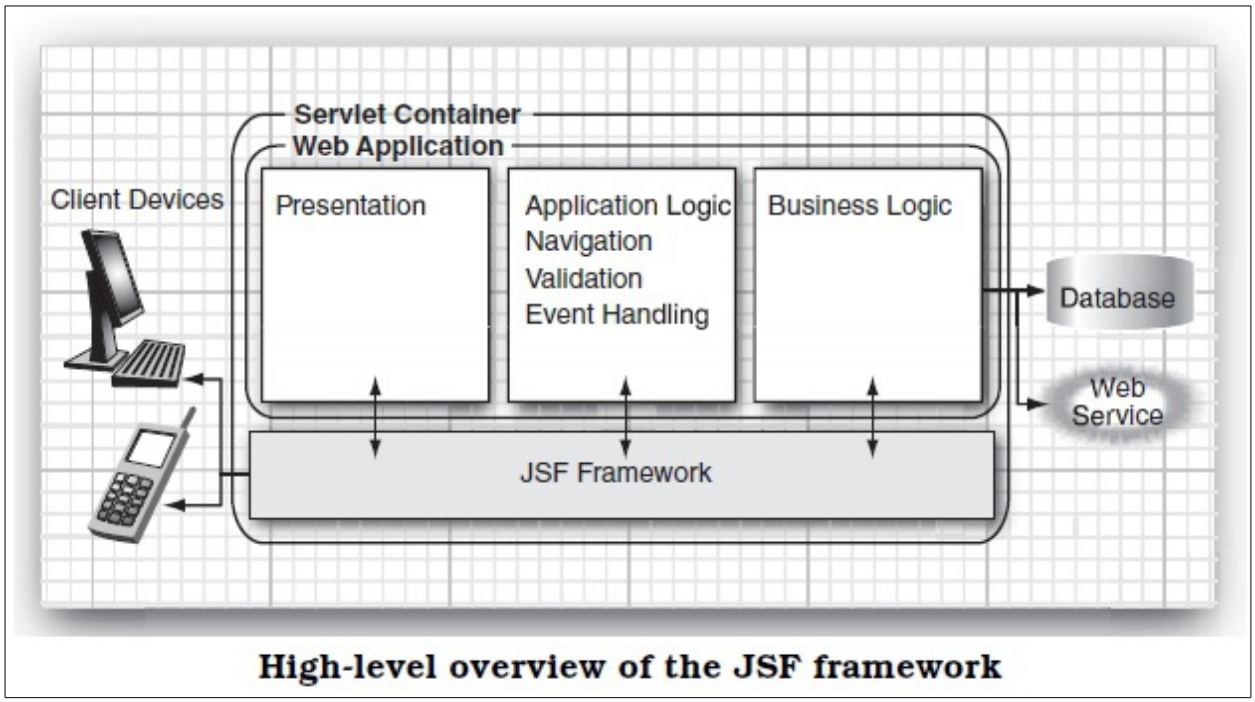
```
@Named('user')
@SessionScoped
```

Here bean name is 'user' and its session scoped.

➔ Beans are managed in the following sense.

When bean name appears in the JSF page then JSF implementation locates the object with that name or constructs one if it doesnt exist in the appropriate scope. For ex, if second user connects for above mentioned bean then another UserBean object will be constructed.

2. JSF Overview



High-level overview of the JSF framework

➔ Scope of JSF is restricted to the presentation tier. Database persistence, web services and other back end connections are outside the scope of JSF.

➔ **Data conversion**—Users enter data into web forms as text. Business objects want data as numbers, dates, or other data types. As explained in Chapter 7, JSF makes it easy to specify and customize conversion rules.

3. JSF Behind the scene

➔ Each tag like h:form and h:inputText have associated tag handler class. When the page is read tag handlers are executed.

➔ Lets take an small application to see how things works with JSF. We have

index.xhtml

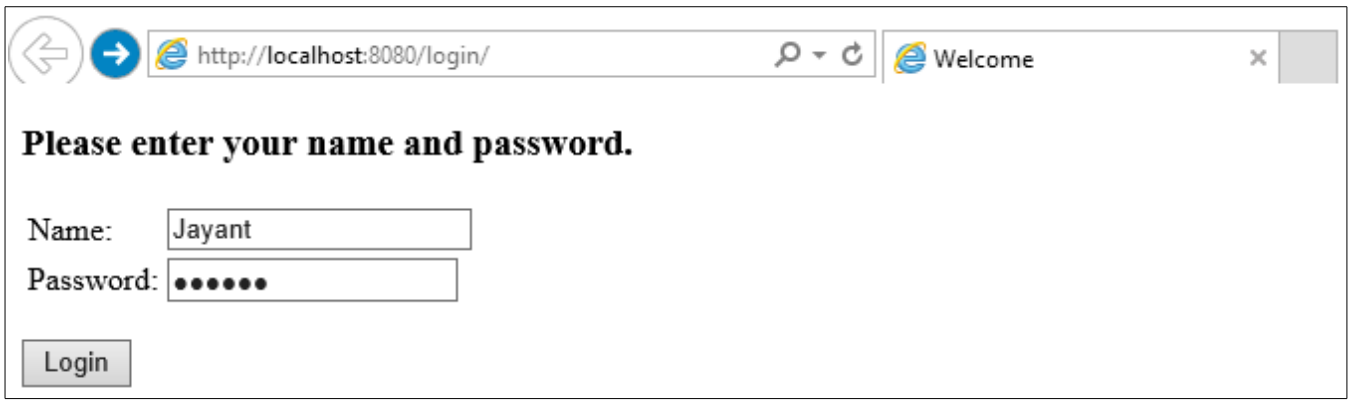
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
   <h:head>
      <title>Welcome</title>
   </h:head>
   <h:body>
      <h:form>
         <h3>Please enter your name and password.</h3>
         <table>
            <tr>
               <td>Name:</td>
               <td><h:inputText value="#{user.name}"/></td>
            </tr>
            <tr>
               <td>Password:</td>
               <td><h:inputSecret value="#{user.password}"/></td>
            </tr>
         </table>
         <p><h:commandButton value="Login" action="welcome"/></p>
      </h:form>
   </h:body>
</html>
```

welcome.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>Welcome</title>
   </h:head>
   <h:body>
      <h3>Welcome to JavaServer Faces, #{user.name}!</h3>
   </h:body>
</html>
```

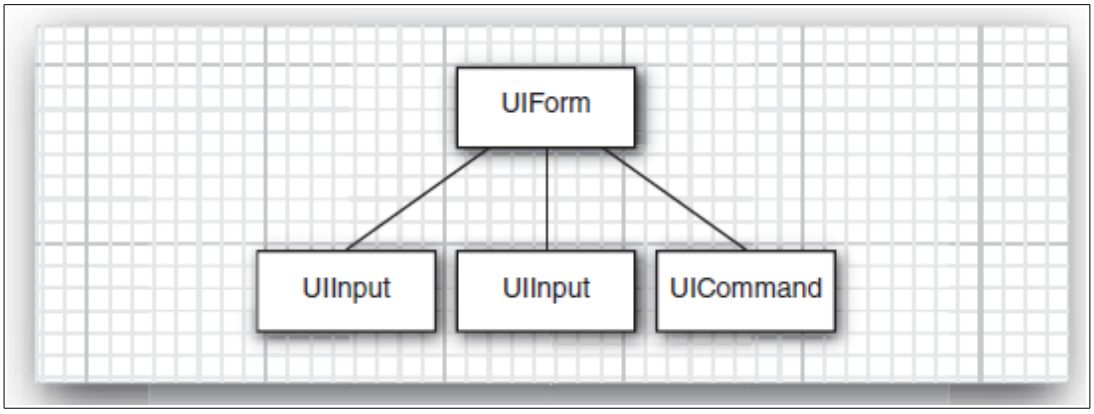
login.xhtml



welcome.xhtml

Lets look at step by step how behind the scene JSF implementation does its job.

1. Component Tree: When the browser first connects through `http://localhost:8080/login/faces/index.xhtml` JSF implementation initializes the `index.xhtml` and JSF tag handlers collabrate with each other to build a `component tree` as shown below. Component tree is a data structure that contains java objects for all user interface elements on JSF page.

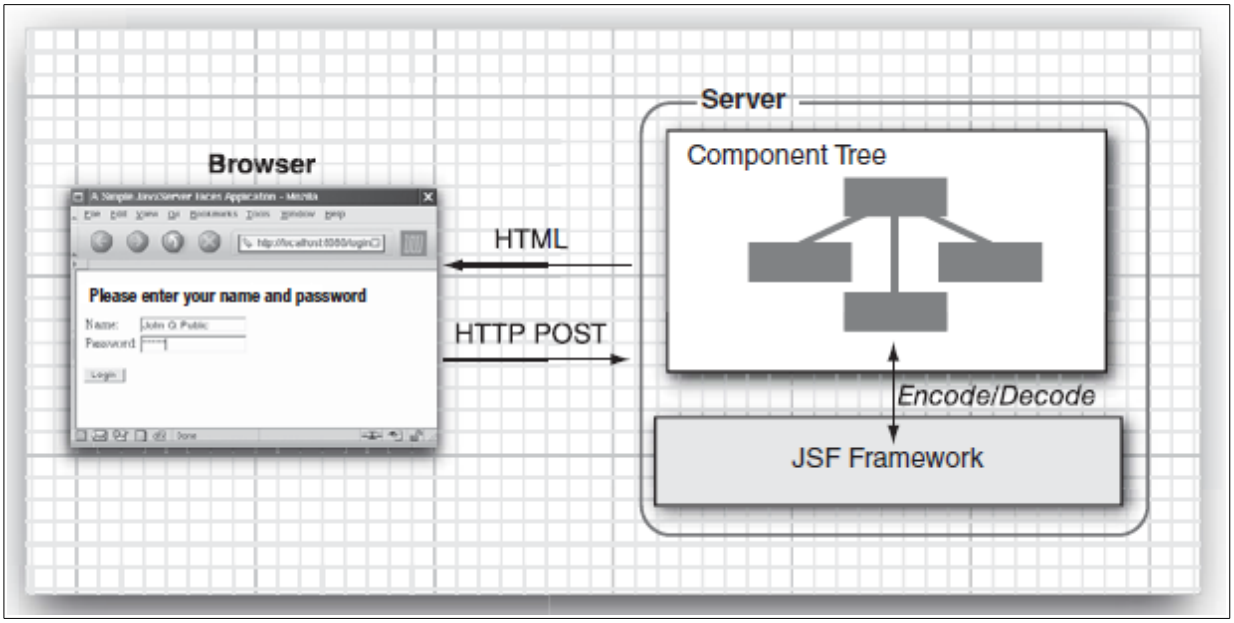


2. Rendering with Encoding: Next, the HTML is rendered. All text which is not JSF tag will pass through. The `h:form`, `h:input`, `h:inputSecret` and `h:commandButton` tags are converted to HTML. For example `h:InputText` component produces following output

`<input type="text" name="unique ID" value="current value"/>`

This process is called encoding. Be default, IDs are assigned by JSF implementation.

This encoded page is sent to the browser and it displays it in a usual way.



3. Decoding:

➔ After page got displayed(`login.xhtml`) the user fills the form data and sends it back to the web server formatted as a POST request.

➔ As part of the normal request processing, the form data is placed in a hash table that all components can access. Next JSF implementation gives chance to each component to inspect that hashtable, a process call `decoding`. Each component

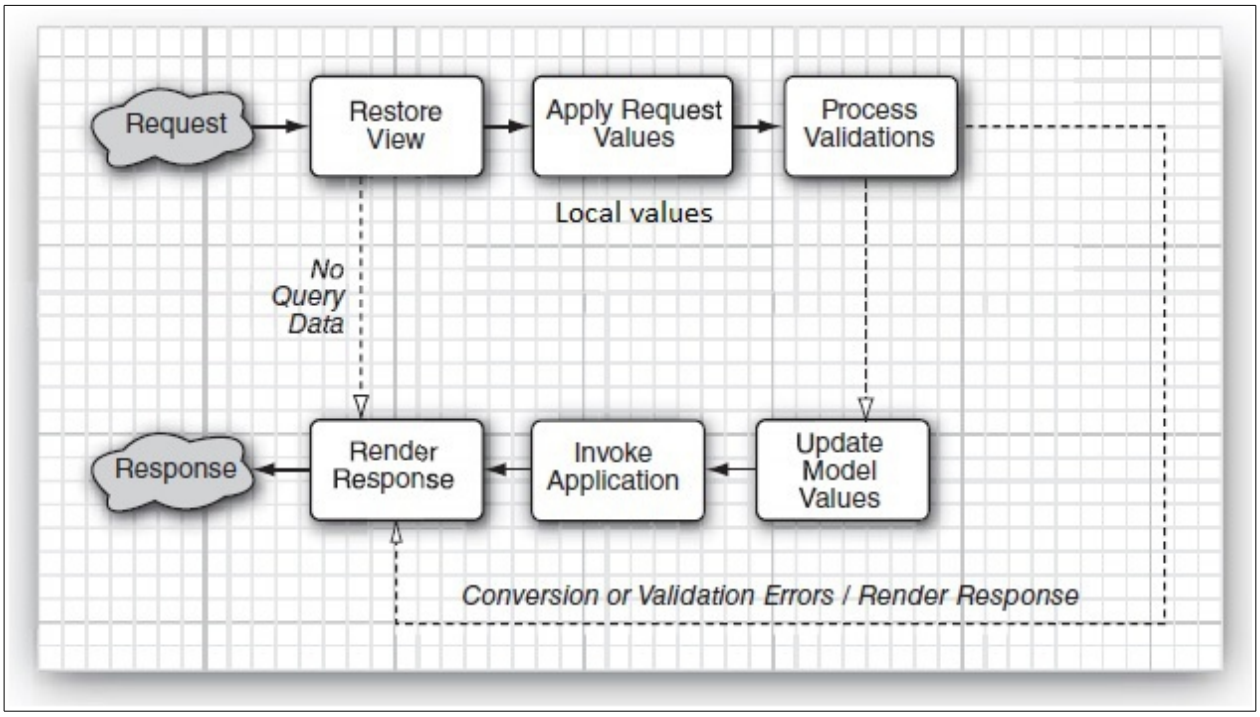
decides on its own how to interpret the form data.

➔ The UIInput components updates the bean properties referenced in the value attribute. They invoke setter methods with the values that the user supplied. UICommand checks whether button has been clicked if so it will fire an action event to launch the login action referenced in the 'action' attribute. That event tells navigation handler to look up the successor page, `welcome.xhtml`.

After this the cycle repeats.

This coding and encoding takes place in 6 phases which is **life cycle** of JSF application.

4. JSF LifeCycle



1. Restore View

Retrieves the component tree for the requested page if it was displayed previously else it will create new component tree. If there are no request values then JSF implementation skips ahead to the 'Render Response' phase. This happens when a page is displayed first time. Otherwise it will go to 'Apply Request Values' phase.

2. Apply Request Values

In this phase each component object in component tree checks which request values belongs to it and stores them. These values stored in components are local values.

3. Process Validations

While creating JSF pages one can attach validators that perform correctness checks on component's local values. These validators are executed in this phase. If validation passes then life cycle proceeds normally however when conversion and validation error occurs JSF implementation skips to the 'Render Response' phase directly, re-displaying the current page so that user has another chance to provide correct inputs.

4. Update Model Values

After converters and validators have completed their work it is assumed that its safe to update the model data. During 'Update Model Values' phase local values are used to update the beans that are wired to the components.

5. Invoke Application

In this phase, the action method of the button or link component that caused the form submission is executed. This method can carry out arbitrary application processing which returns a String that is passed to the navigation handler. The navigation handler looks up the next page.

6. Render Response

Finally this phase encodes the response and send it to the browser.

Chapter 2 Managed Beans

1. Introduction

➔ JSF uses beans to achieve separation between presentation logic and business logic.

➔ Java bean is a removable software component that can be manipulated in a builder tool.

➔ In case of JSF, bean stores the state of web pages.

➔ The JSF implementation does the following.

I. Creates and discards bean as needed.

ii. Read bean properties when displaying a web page.

Iii. Set bean properties when form is posted.

➔ If we have `UserBean` then we need to mention annotations as shown below

```
@ManagedBean(name="user")
@SessionScoped
```

If we remove 'name' attribute then to use bean properties in JSF pages one should use `#{userBean.fname}` i.e. first letter of `UserBean` should be lowercase.

➔ `@ManagedBean` annotation is present in `javax.faces.bean` package. The other one `@ManagedBean` is in `javax.annotation` package which doesn't work for JSF.

➔ When the expression with name '`user`' encounters JSF implementation constructs an object of bean class `UserBean`. The object stays alive during the duration of session(as its `SessionScoped`).

➔ Each session belongs to separate client will have its own `UserBean` object.

2. The Bean Properties

➔ Bean class must have non-arg public constructor.

➔ Bean properties have both setters and getters methods. If it has only setter method then its write only and only getter method then its read only property.

➔ A 'get' method must have no parameters and a 'set' method must have one parameter and no return value.

➔ With get and set the first letter of property name becomes upper case letter example property `fname` becomes `getFname()`. If property's name itself starts with capital letter like `URL` then it will remain same `getURL()`.

➔ For boolean type there are choice for prefix

Ex

public boolean isConnected();

or

public boolean getConnected();

both are same.

3. Value Expression

➔ Expression like `#{user.fname}` are called value expressions.

➔ `<h:inputText value=#{user.fname}/>` calls setter method.

➔ `Welcome #{user.fname}` calls getter method.

4. CDI Beans(Context and Dependency Injection Beans)

➔ These beans are bound to contexts(such as current request, a browser session or even a user-defined life cycle context).

➔ CDI beans are more powerful concepts then managed beans and if one wants to deploy his app in J2EE application server then its make more sense to use CDI beans.

➔ CDI beans are used in the same way as managed beans with 'Named' annotation as shown below

```
@Named(“user”)
@SessionsScoped
public class UserBean implements Serializable
{
```

➔ Here session scope is for `javax.enterprise.context` package.

➔ Note session scoped bean **must** implement `Serializable` interface.

➔ One must also include `WEB-INF/beans.xml` to activate CDI bean processing. This file could be empty or it can optionally contain instruction for configuring the beans.

5. Message Bundles

➔ JSF provides facility to have all your messages at a single place.

➔ All messages are collected in a file time-honored properties format

```
guessNext=Guess the next number in Sequence !!
answer=Your Answer :
```

➔ All this messages will be saved in lets say messages.properties file and kept together with class ex src/java/com/corejsf/messages.properties. One can have any file name or directory path but one must use extension .properties.

➔ Managed bundles can be declared in two ways

i. Using faces-config.xml

```
<application>
    <resource-bundle>
         <base-name>com.msgs.messages</base-name>
         <var>msgs</var>
    </resource-bundle>
</application>
```

This is more efficient as it is created once for whole application.

ii. One can add <F:LoadBundle> element to JSF page where message will be used as shown below

```
<f:LoadBundle basename=”com.corejsf.messages” var=”msgs”/>
```

➔ In either case, the messages are accessible through a map variable '`msgs`' and used in expression as

```
#{msgs.guessNext}
```

➔ For local(not international) bundles one should add local suffix to the file name : an underscore followed by lowercase, two letter ISO-369 language code.

EX. For German strings we will have messages_de.properties.

6. Messages With Variable Parts

➔ We can have a part of message as variable I.e generated dynamically.

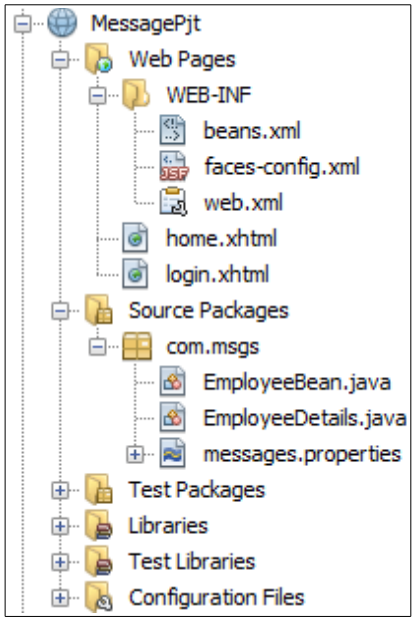
Ex. `email=Email ID is : {0}`

➔ The above message will be stored in messages.properties file. It will be used inside JSF with `<h:OutputFormat>` and `<f:param>` tag as shown below

```
<h:outputFormat value="#{msgs.email}">
      <f:param value="#{empBean.email}"/>
</h:outputFormat>
```

➔ Example

Our MessagePjt directory is as shown below.



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/login.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
              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-facesconfig_2_2.xsd">
<application>
    <resource-bundle>
         <base-name>com.msgs.messages</base-name>
         <var>msgs</var>
    </resource-bundle>
</application>
</faces-config>
```

EmployeeDetails.java

```
package com.msgs;


public class EmployeeDetails {

    String name, email;
    int empid;
}
```

EmployeeBean.java

```
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.TreeSet;
import javax.inject.Named;
import javax.enterprise.context.SessionScoped;

@Named(value = "empBean")
@SessionScoped
public class EmployeeBean implements Serializable{

    public EmployeeBean() {
    }

    String name, email;

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
    int empid;

    public String getName() {
        return name;
    }

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

    public int getEmpid() {
        return empid;
    }

    public void setEmpid(int empid) {
        this.empid = empid;
    }
```

```
    public String checkDet()
    {
       EmployeeDetails ed1 = new EmployeeDetails();
       EmployeeDetails ed2 = new EmployeeDetails();


       ed1.name="Jayant Joshi";
       ed1.empid=111;
       ed1.email="jayantjoshi0209@gmail.com";

       ed2.name="Gunagya Joshi";
       ed2.empid=222;
       ed2.email="GunagyaJoshi@gmail.com";

       ArrayList<EmployeeDetails> empDet = new ArrayList<>();
       empDet.add(0, ed1);
       empDet.add(1, ed2);

       for(int i=0; i<empDet.size();i++)
       {
           EmployeeDetails ed = empDet.get(i);
           if(ed.empid==empid)
           {
               setEmail(ed.email);
               return "home";
           }
       }
       return "login";
    }
}
```

login.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">

    <h:head>
        <title>#{msgs.title1}</title>
    </h:head>

    <h1>#{msgs.heading1}</h1>
    <h:body>
        <h:form>
            <h:outputLabel value="${msgs.name}"/>
            <h:inputText id="name" value="#{empBean.name}"/><br/>

            <h:outputLabel value="${msgs.empid}"/>
            <h:inputText id="empid" value="#{empBean.empid}"/><br/>

            <h:commandButton id="submit" value="submit" action="#{empBean.checkDet}"/>

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

home.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://java.sun.com/jsf/core">
    <h:head>
        <title>#{msgs.title2}</title>
```

```
    </h:head>

    <h1>#{msgs.heading2}</h1>
    <h:body>
        Hello #{empBean.name}<br/>
        <h:outputFormat value="#{msgs.email}">
            <f:param value="#{empBean.email}"/>
        </h:outputFormat>
    </h:body>
</html>
```

messages.properties

```
title1=login Page
title2=Home Page
heading1=Welcome to Login Page
heading2=Welcome to Home Page
name=Enter Your Name :
empid=Enter Your EmpID :
email=Email ID : {0}
```

Output

1)

login.xhtml

home.xhtml

2)

login.xhtml

home.xhtml

7. Bean Scopes

➔ Most commonly used scopes are

```
@SessionScoped
@RequestScoped
@ApplicationScoped
```

These annotations are in package `javax.faces.bean` for JSF managed beans and in `javax.enterprise.context` for CDI beans.

➔ Only request scope beans are single threaded therefore thread safe. Other scope beans are not single threaded.

➔ Apart from it there are two more scopes

I. Conversation Scope

Conversation scope is easy to use. Follow these rules:

• Use a CDI bean—this is a feature of CDI, not JSF.

• Use the `@ConversationScoped` annotation.

• Add an instance variable: private `@Inject` Conversation conversation; The instance variable will be automatically initialized with a Conversation object when the bean is constructed.

• Call `conversation.begin()` to elevate the scope of the bean from request scope to conversation scope.

• Call `conversation.end()` to remove the bean from conversation scope.

A session could have many conversations.

ii. View Scope

If you have a page that keeps getting redisplayed, then one can use view scope.

8. Cofiguring Beans

i. injecting CDI beans

➔ Sometimes it needed to wire two beans together. Suppose we have `UserBean` that contains information about the current user, and an `EditBean` needs to know about that user. Then one can inject `UserBean` instance into `EditBean` as shown below

```
@Named
@SessionScoped
public class EditBean {
@Inject private UserBean currentUser;
...
}
```

Here when `EditBean` is constructed, an appropriate `UserBean` instance is located in the current session. The

`currentUser` instance variable is then set to `UserBean`.

ii. Injecting Managed Beans

Suppose you have a UserBean with name user that contains information about the current user. Here is how you can inject it into a field of another bean:

```
@ManagedBean
@SessionScoped
public class EditBean implements Serializable {
@ManagedProperty(value="#{user}")
private UserBean currentUser;
public void setCurrentUser(UserBean newValue) { currentUser = newValue; }
. . .
}
```

Note that you annotate the currentUser field, but you *must* supply a setCurrentUser method. When an EditBean instance is constructed, the value expression #{user} is evaluated, and the result is passed to the setCurrentUser method.

9. Bean Life Cycle Annotation

➔ Using `@PostConstruct` and `@PreDestroy` annotations, one can specify bean methods which will be automatically called after bean has been constructed and before bean has been destroyed.

```
public class MyBean {
@PostConstruct
public void initialize() {
// initialization code
}
@PreDestroy
public void shutdown() {
// shutdown code
}
// other bean methods
}
```

The above annotations works for both CDI and JSF managed beans.

➔ Configuring Managed Beans with XML

Before JSF 2.0 all beans needs to be configured in XML. Now a days you have choice between XML and annotations. Bean can be configured in WEB-INF/faces-config.xml file or file ending with name .faces-config.xml file.

I. Defining beans

```
<faces-config>
<managed-bean>
      <managed-bean-name>user</managed-bean-name>
      <managed-bean-class>com.corejsf.UserBean</managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
</faces-config>
```

ii. Set property values

```
<managed-bean>
 <managed-bean-name>user</managed-bean-name>
 <managed-bean-class>com.corejsf.UserBean</managed-bean-class>
 <managed-bean-scope>session</managed-bean-scope>
<managed-property>
 <property-name>name</property-name>
 <value>troosevelt</value>
</managed-property>
<managed-property>
```

```
 <property-name>password</property-name>
 <value>jabberwock</value>
</managed-property>
</managed-bean>
```

When the bean is created, the setters corresponding to properties defined in XML will be called and values are set.

One can also use

```
<property-name>user</property-name>
<value>#{user}</value>
```

iii. Initializing list or map

List:

```
<list-entries>
 <value-class>java.lang.Integer</value.class>
 <value>3</value>
 <value>1</value>
 <value>4</value>
 <value>1</value>
 <value>5</value>
</list-entries>
```

Map:

```
<map-entries>
 <key-class>java.lang.Integer</key-class>
 <map-entry>
  <key>1</key>
  <value>George Washington</value>
 </map-entry>
 <map-entry>
  <key>3</key>
  <value>Thomas Jefferson</value>
 </map-entry>
 <map-entry>
  <key>16</key>
  <value>Abraham Lincoln</value>
 </map-entry>
 <map-entry>
  <key>26</key>
  <value>Theodore Roosevelt</value>
 </map-entry>
</map-entries>
```

<list-entries> and <map-entries> can be used to initialized managed-bean or a managed-property provided that the bean or property type is a List or Map.

9. The Expression Language Syntax

i. rvalue and lvalue mode

➔ Expression a.b can be rendered in rvalue mode or lvalue mode.

Ex. `<h:inputText value="#{user.name}"/>`

When the text field is rendered, the expression user.name is evaluated in rvalue mode, and the getName method is called. During decoding, the same expression is evaluated in lvalue mode and the setName method is called.

In general, the expression a.b in rvalue mode is evaluated by calling the property getter, whereas a.b in lvalue mode calls the property setter.

ii. Using bracket

➔

```
a.b
a["b"]
a['b']
```

i.e. `user.password`, `user["password"]`, and `user['password']` are equivalent expressions.

➔ Bracket is always a good choice to use in case 'a' evaluates to map or array.

Iii. Map and List Expression

➔ Here too rvalue and lvalue analogy applies.

Ex. For `m.key` or `m["key"]`

In rvalue mode evaluates to

`m.get("key")`

In lvalue mode it evaluates to

`m.put("key", right)`

Evaluating the Value Expression a.b

Type of a	Type of b	lvalue Mode	rvalue Mode
null	any	error	null
any	null	error	null
Map	any	a.put(b, right)	a.get(b)
List	convertible to int	a.set(b, right)	a.get(b)
array	convertible to int	a[b] = right	a[b]
bean	any	call setter of property with name b.toString()	call getter of property with name b.toString()

Table 8–1 Tag Libraries Supported by Facelets

Tag Library	URI	Prefix	Example	Contents
JavaServer Faces Facelets Tag Library	http://xmlns.jcp.org/jsf/facelets	ui:	ui:component ui:insert	Tags for templating
JavaServer Faces HTML Tag Library	http://xmlns.jcp.org/jsf/html	h:	h:head h:body h:outputText h:inputText	JavaServer Faces component tags for all UIComponent objects
JavaServer Faces Core Tag Library	http://xmlns.jcp.org/jsf/core	f:	f:actionListener f:attribute	Tags for JavaServer Faces custom actions that are independent of any particular render kit
Pass-through Elements Tag Library	http://xmlns.jcp.org/jsf	jsf:	jsf:id	Tags to support HTML5-friendly markup
Pass-through Attributes Tag Library	http://xmlns.jcp.org/jsf/passthrough	p:	p:type	Tags to support HTML5-friendly markup
Composite Component Tag Library	http://xmlns.jcp.org/jsf/composite	cc:	cc:interface	Tags to support composite components
JSTL Core Tag Library	http://xmlns.jcp.org/jsp/jstl/core	c:	c:forEach c:catch	JSTL 1.2 Core Tags
JSTL Functions Tag Library	http://xmlns.jcp.org/jsp/jstl/functions	fn:	fn:toUpperCase fn:toLowerCase	JSTL 1.2 Functions Tags

4. Lifecycle of a Facelets Application

I. When a client, such as a browser, makes a new request to a page that is created using Facelets, a new component tree or `javax.faces.component.UIViewRoot` is created and placed in the `FacesContext`.

ii. The `UIViewRoot` is applied to the `Facelets`, and the view is populated with components for rendering.

Iii. The newly built view is rendered back as a response to the client.

iv. On rendering, the state of this view is stored for the next request. The state of input components and form data is stored.

v. The client may interact with the view and request another view or change from the JavaServer Faces application. At this time, the saved view is restored from the stored state.

vi. The restored view is once again passed through the JavaServer Faces lifecycle, which eventually will either generate a new view or re-render the current view if there were no validation problems and no action was triggered.

vii. If the same view is requested, the stored view is rendered once again.

viii. If a new view is requested, then the process described in Step 2 is continued.

ix. The new view is then rendered back as a response to the client.

Chapter 3 Navigation

1. Introduction

➔ Navigation Handler is reponsible for selecting the next JSF page.

2. Types of Navigation

There are two types of Navigation

I. Static Navigation

➔ In a simple web application navigation is static. That is, clicking a particular button always selects a fixed JSF page for response.

➔ Static navigation is achieved through '`action`' attribute provided to each button or link.

```
<h:commandButton value=”Login” action=”welcome”/>
```

➔ The value of action is called 'outcome'. This outcome will be mapped to view-IDs. If they are not mapped then outcome are transformed into view-IDs using following steps

a. If outcome don’t have any extension then extension of current view will be appended.

b. If outcome doesn't start with '/' then path of the current view will be prepended.

Ex. In /index.xhtml, the outcome is 'welcome' and it yields the target view-ID '/welcome.xhtml'.

Note: Since JSF 2.0 mapping of the view-ID is optional. Prior to JSF 2.0 one had to specify explicit navigation rules for every outcome.

ii. Dynamic Navigation

➔ To implement dynamic navigation, the '`action`' attribute must have a method expression.

Ex:

```
<h:commandButton value=”Login” action=#{loginController.verifyUser}/>
```

Here `LoginController` references a bean class and it must have a method 'verifyUser'.

➔ Here is a example of action method

```
String verifyUser()
{
if(...)
return “success”;
else
return “failure”;
}
```

Depends on outcome success or failure next view will be determined.

Note: Action method may return null to indicate current view should be redisplayed.

➔ When '`action`' is a method expression then following steps are carried out

i. The specified bean is retrieved.

ii. The reference method is called and a String is returned.

iii. The outcome string is transformed into view-ID.

iv. The page corresponding to view-ID is displayed.

3. Mapping outcomes to view-IDs

➔ JSF provides mechanism for mapping logical outcome of methods to actual web pages.

➔ It is achieved by adding `navigation-rule` entries into `faces-config.xml`.

Ex

```
<navigation-rule>
   <from-view-id>/index.xhtml</from-view-id>
      <navigation-case>
            <from-outcome>success</from-outcome>
            <to-view-id>/welcome.xhtml</to-view-id>
      </navigation-case>
</navigation-rule>
```

➔ If one has action method 'logout' the application pages(i.e. many pages have logout button) then one can have all these buttons navigate to the loggedOut.xhtml page as shown below

```
<navigation-rule>
   <navigation-case>
   <from-outcome>logout</from-outcome>
   <to-view-id>/loggedOut.xhtml</to-view-id>
   </navigation-case>
</navigation-rule>
```

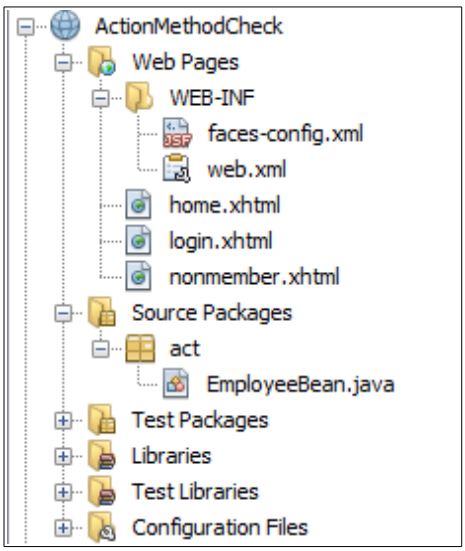
The above rule applies to all the pages because no `<from-view-id>` tag is present.

➔ One can merge navigation rules with same `<from-vieww-id>` as shown below

```
<navigation-rule>
   <from-view-id>/index.xhtml</from-view-id>
      <navigation-case>
            <from-outcome>success</from-outcome>
            <to-view-id>/welcome.xhtml</to-view-id>
      </navigation-case>
      <navigation-case>
            <from-outcome>failure</from-outcome>
            <to-view-id>/newUser.xhtml</to-view-id>
      </navigation-case>
</navigation-rule>
```

➔ Ex

Lets create ActionMethodCheck directory as shown below



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/login.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

faces-config.xml

```
<?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">
   <navigation-rule>
      <navigation-case>
         <from-outcome>response1</from-outcome>
         <to-view-id>/home.xhtml</to-view-id>
      </navigation-case>
   </navigation-rule>
    <navigation-rule>
      <navigation-case>
         <from-outcome>response2</from-outcome>
         <to-view-id>/nonmember.xhtml</to-view-id>
      </navigation-case>
   </navigation-rule>
</faces-config>
```

EmployeeBean.java

```
package act;


import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name="empbean")
@SessionScoped

public class EmployeeBean {

    public EmployeeBean() {
    }

    String name;
    int empid;
    int [] empids = {111,222,333,444,555};

    public String getName() {
        return name;
    }
```

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

    public int getEmpid() {
        return empid;
    }

    public void setEmpid(int empid) {
        this.empid = empid;
    }

    public String valEmpId()
    {
        for(int i=0;i<empids.length;i++)
        {
           while(empid==empids[i])
           {

               return "home";
           }
        }
        return "nonmember";
    }
}
```

login.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">
    <h:head>
        <title>Login Page</title>
    </h:head>
    <h:body>
        <h:form>
            <h:outputLabel value="Name : "/>
            <h:inputText id="name" value="${empbean.name}"/><br/>

            <h:outputLabel value="EmpID : "/>
            <h:inputText id="empid" value="${empbean.empid}"/><br/>

            <h:commandButton id="submit" value="submit" action="#{empbean.valEmpId}"/>
        </h:form>
    </h:body>
</html>
```

home.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">
    <h:head>
        <title>Home Page</title>
    </h:head>
    <h:body>
        Welcome ${empbean.name}
    </h:body>
</html>
```

nonmember.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">
```

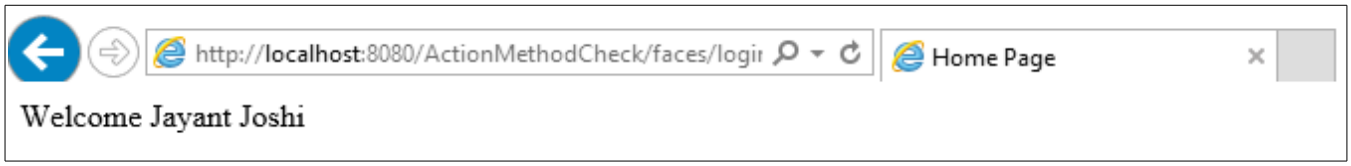
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Non Member Page</title>
    </h:head>
    <h:body>
        Sorry #{empbean.name} you are not allowed !!!
    </h:body>
</html>
```

Output

1)

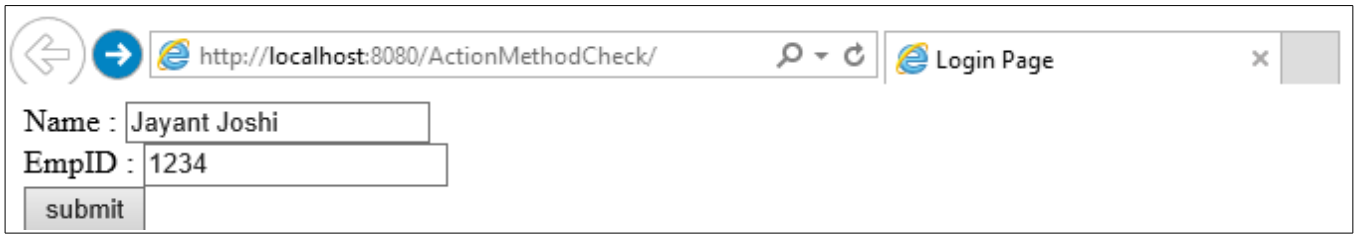
login.xhtml

home.xhtml

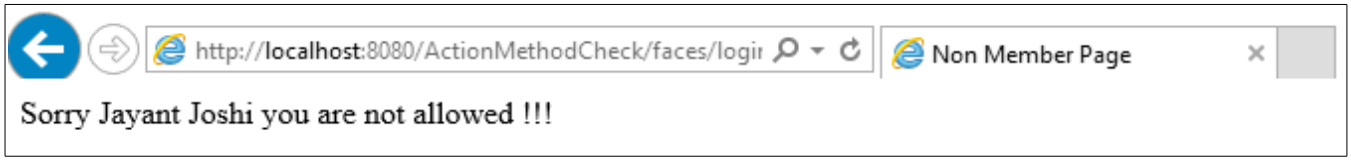


2)

login.xhtml



nonmember.xhtml



4. Redirection

➔ One can ask JSF implementation to redirect to a new view.

➔ When redirect happens address field changes.

Using Redirect

I. Without Navigation Rules

➔ Add the string like '?faces-redirect=true' to the outcome string.

Ex. <h:commandButton value=”Login” action=”welcome?faces-redirect=true”/>

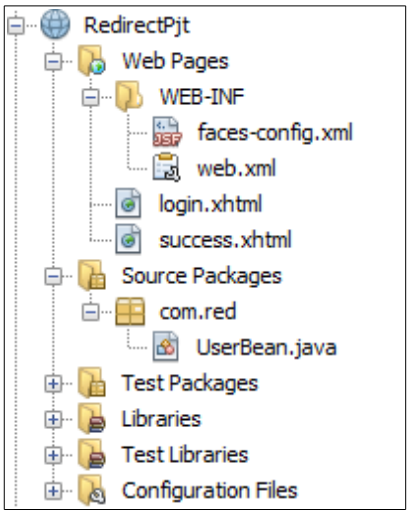
Here when button is clicked it will directly go to welcome.xhtml.

ii. With Navigation Rule(Configuration file)

```
<navigation-case>
   <from-outcome>success</from-outcome>
   <to-view-id>/success.xhtml</to-view-id>
   <redirect/>
</navigation-case>
```

➔ Ex

Lets create RedirectPjt directory as shown below



login.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">
    <h:head>
        <title>Login Page</title>
    </h:head>
    <h:body>
        <h:form>
            <h:outputLabel value="Name :"/>
            <h:inputText value="#{user.name}"/><br/>
            <h:outputLabel value="Emp ID"/>
            <h:inputText value="#{user.empid}"/><br/>
            <h:commandButton id="submit" value="submit" action="success?faces-
            redirect=true"/>
            //For navigation rules
            /*<h:commandButton id="submit" value="submit" action="#{user.verify}"/>*/

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

success.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">
    <h:head>
        <title>Success Page</title>
    </h:head>
    <h:body>
        Hello #{user.name}<br/>
        success redirect
    </h:body>
</html>
```

UserBean.java

```
package com.red;

import java.io.Serializable;
import javax.inject.Named;
import javax.enterprise.context.Dependent;
import javax.enterprise.context.SessionScoped;

@Named(value = "user")
@SessionScoped
public class UserBean implements Serializable{

    public UserBean() {
    }

    public String getName() {
        return name;
    }

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

    public int getEmpid() {
        return empid;
    }

    public void setEmpid(int empid) {
        this.empid = empid;
    }

    String name;
    int empid;

    public String verify()
    {
       if(empid==111)

       return "success";
       else
       return "failure";
    }
}
```

faces-config.xml

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

<faces-config version="2.2"
              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-facesconfig_2_2.xsd">
```

```
    <navigation-rule>
        <navigation-case>
            <from-view-id>/login.xhtml</from-view-id>
            <from-outcome>success</from-outcome>
            <redirect/>
            <to-view-id>/success.xhtml</to-view-id>

        </navigation-case>
        <navigation-case>
            <from-view-id>/login.xhtml</from-view-id>
            <from-outcome>failure</from-outcome>
            <to-view-id>/failure.xhtml</to-view-id>
        </navigation-case>
    </navigation-rule>
</faces-config>
```

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/login.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Output

login.xhtml

success.xhtml

Note : See the URL changes and in both the cases output will remain the same.

5. RESTful Navigation and Bookmarkable URLs

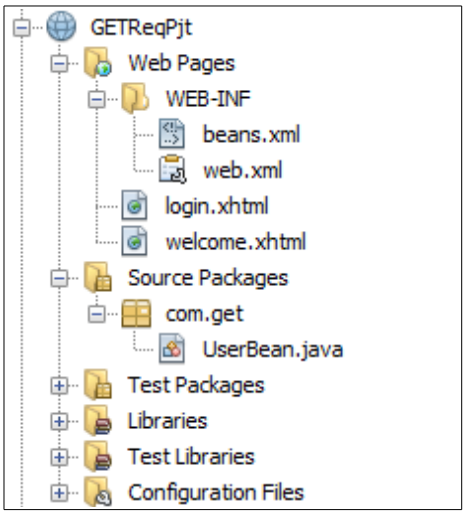
➔ By default, JSF application makes a sequence of POST request to the server.

➔ POST requests are neither bookmarkable nor cacheable.

➔ JSF 2.0 onward GET reqeusts too are supported.

➔ EX

Let create a 'GETReqPjt' directory as shown below



login.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">

    <h:head>
        <title>Login Page</title>
    </h:head>
    <h:body>
        <h:form>
            Hello there <br/>
            Click here : <h:link outcome="/welcome.xhtml"  value="link">
                            <f:param name="name" value="JayJosh"/>
                         </h:link>
        </h:form>
    </h:body>
</html>
```

welcome.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">
    <f:metadata>
        <f:viewParam name="name" value="#{user.name}" />
   </f:metadata>
    <h:head>
        <title>Welcome Page</title>
```

```
    </h:head>
    <h:body>
        Hello #{user.name}
    </h:body>
</html>
```

UserBean.java

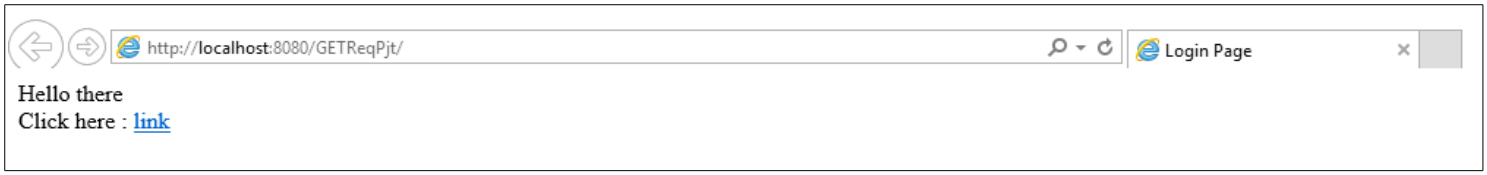
```
package com.get;


import java.io.Serializable;
import javax.inject.Named;
import javax.enterprise.context.SessionScoped;

@Named("user")
@SessionScoped

public class UserBean implements Serializable {

    public UserBean() {
    }

    String name;

    public String getName() {
        return name;
    }

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

web.xml

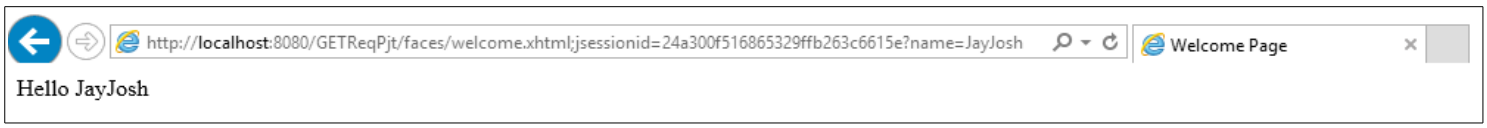
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</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/login.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Output

login.xhtml



welcome.xhtml



6. More Navigation Rules

i. WildCards

➔ One can use wild cards in the <from-view-id> element of the navigation rule.

➔ Ex

```
<navigation-rule>
<from-view-id>/secure/*</from-view-id>
<navigation-case>
. . .
</navigation-case>
</navigation-rule>
```

Now this applies to all the pages that starts with the prefix `/secure`.

➔ If one use

```
<from-view-id>/*</from-view-id>
or
<from-view-id>*</from-view-id>
```

Then its means that it applies to all the pages.

ii. <form-action> tag

➔ When different method expressions in the application have methods which returns same outcome then we can use <form-action> to differentiate between them.

➔ Ex

```
<navigation-case>
      <from-action>#{quizBean.answerAction}</from-action>
      <from-outcome>again</from-outcome>
      <to-view-id>/again.xhtml</to-view-id>
</navigation-case>
<navigation-case>
      <from-action>#{quizBean.startOverAction}</from-action>
      <from-outcome>again</from-outcome>
      <to-view-id>/index.xhtml</to-view-id>
</navigation-case>
```

Above contents of `<from-action>` should match that of `<from-outcome>`.

➔ Here Navigation handler does not invoke the method present in #{ }. The methods will be invoked before navigation handler kicks in. The navigation handler mere uses <form-action> as key to find matching navigation cases.

iii. Conditional Navigation Cases

➔ From JSF 2.0 one can supply 'if' element that activates navigation case only when the condition is fulfilled.

➔ Ex

```
<navigation-case>
<from-outcome>previous</from-outcome>
<if>#{quizBean.currentQuestion != 0}</if>
<to-view-id>/main.xhtml</to-view-id>
</navigation-case>
```

iv. Dynamic Target View IDs

➔ <to-view-id> element could be value expression. It will be evaluated and the result will be used as view ID.

➔ EX

```
<navigation-rule>
      <from-view-id>/main.xhtml</from-view-id>
      <navigation-case>
      <to-view-id>#{quizBean.nextViewID}</to-view-id>
      </navigation-case>
</navigation-rule>
```

Chapter 4 STD JSF Tags

1. Styles

➔ One can use styles, either inline(style) or classes(styleClass) to influence how components are rendered.

```
<h:outputText value="#{customer.name}" styleClass="emphasis"/>
<h:outputText value="#{customer.id}" style="border: thin solid blue"/>
```

➔ CSS style attributes can be value expressions that give programmatic control over style.

2. Resources

➔ One can place all style sheets, java script files, images and other files into 'resources' directory in the root of web app.

➔ Sub-directories of 'resources' directories are called libraries. One can create libraries as one like. Ex css, images, java script etc.

➔ To include style sheet use tag

```
<h:outputStylesheet library="css" name="styles.css"/>
```

The tag adds a link of the form

```
<link href="/context-root/faces/javax.faces.resource/styles.css?ln=css" rel="stylesheet"
type="text/css"/>
```

➔ To include java script use tag `<h:outputScript>`

```
<h:outputScript name="jsf.js" library="javascript" target="head"/>
```

➔ Here 'target' could be 'head' or 'body'. The script is appended to the 'head' or 'body' facet of the root component, which means that it appears at the end of the head or body in the generated HTML. If there is no target the script will be inserted in the current location.

➔ To include image library use <h:graphicImage> tag

```
<h:graphicImage name="logo.png" library="images"/>
```

➔ One can provide version for library. Ex.

```
resources/css/1_0_2
resources/css/1_1
```

Then latest version `resources/css/1_1` will be used.

➔ One can even provide version of file also where one need to replace the resource with directory of same name and then use version name as file name. Example is shown below

```
resources/css/styles.css/1_0_2.css
resources/css/styles.css/1_1.css
```

3. DHTML Events

➔ Dynamic HTML is supported by nearly all JSF HTML tags.

➔ Some DHTML attributes are shown below(there are many such attributes).

DHTML Event Attributes[a]

Attribute	Description
onblur (16)	Element loses focus
onchange (11)	Element's value changes
onclick (17)	Mouse button is clicked over the element
ondblclick (21)	Mouse button is double-clicked over the element

4. Panels

➔ We have `<h:panelGrid>` tag which generates HTML markup for laying out components in rows and columns.

➔ If we have 4 components and <h:panelGrid> specified with 2 columns then one will have two rows with 2 components each.

➔ Ex

1. <h:panelGrid columns="2">

2. <h:panelGrid columns="3">

➔ <h:panelGrid> is often used with <h:panelGroup> with which two or more components are grouped and then they are treated as one

```
<h:panelGrid columns=”1”>
     <h:panelGroup>
     <h:inputText id='name' value=#{user.name}”>
     <h:message for=”name”/>
     </h:panelGroup>
</h:panelGrid>
```

➔ If no. of columns are not mentioned then by default it is '1'.

5. Text Fields and Text Areas

➔ JSF supports three varieties of text inputs

```
h:inputText
h:inputSecret
h:inputTextarea
```

h:inputText and h:inputSecret Examples

Example	Result
`<h:inputText value="#{form.testString}" readonly="true"/>`	12345678901234567890
`<h:inputSecret value="#{form.passwd}" redisplay="true"/>`	********** (shown after an unsuccessful form submit)
`<h:inputSecret value="#{form.passwd}" redisplay="false"/>`	(shown after an unsuccessful form submit)
`<h:inputText value="inputText" style="color: Yellow; background: Teal;"/>`	inputText
`<h:inputText value="1234567" size="5"/>`	123456
`<h:inputText value="1234567890" maxlength="6" size="10"/>`	123456

Note: 'size' attribute specifies the number of visible characters in a text field. But we can see in example 5 where size=5 is defined still 123456 i.e. 6 characters appears in text field. Hence 'size' attribute is not precise where 'maxlength' attribute is precise and gives perfect result as shown in last example.

➔ <h:inputTextArea> examples

h:inputTextarea Examples

Example	Result
`<h:inputTextarea rows="5"/>`	
`<h:inputTextarea cols="5"/>`	
`<h:inputTextarea value="123456789012345" rows="3" cols="10"/>`	456789012345
`<h:inputTextarea value="#{form.dataInRows}" rows="2" cols="15"/>`	line one line two line three

➔ We can see in third example if we place long string as 'value' then the whole string will be displayed in one line. Also to keep data in separate lines one can insert new line character(/n) to force a line break. In last example dataInRows property of backing bean is implemented as shown below.

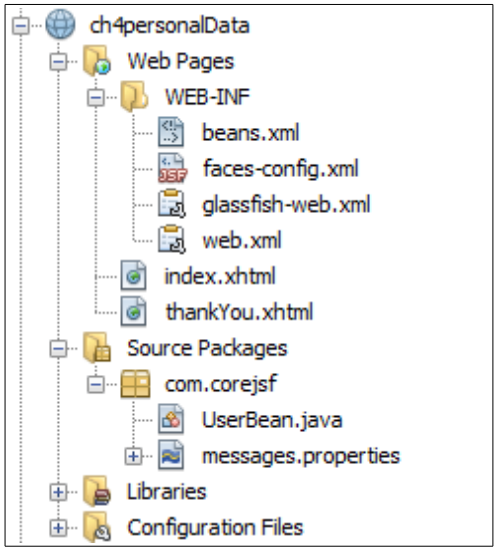
```
private String dataInRows = "line one\nline two\nline three";
```

```
public void setDataInRows(String newValue) {
dataInRows = newValue;
}
public String getDataInRows() {
return dataInRows;
}
```

➔ Example for text field and text area tags

```
h:inputText
h:inputSecret
h:inputTextarea
```

Lets create directory for project 'ch4personalData' as shown below



UserBean.java

```
package com.corejsf;


import java.io.Serializable;

import javax.inject.Named;
   // or import javax.faces.bean.ManagedBean;
import javax.enterprise.context.SessionScoped;
   // or import javax.faces.bean.SessionScoped;

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

   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 String getAboutYourself() { return aboutYourself; }
   public void setAboutYourself(String newValue) { aboutYourself = newValue; }
}
```

messages.properties

```
indexWindowTitle=Using Textfields and Textareas
```

```
thankYouWindowTitle=Thank you for submitting your information
thankYouPageTitle=Thank you!
indexPageTitle=Please enter the following personal information
namePrompt=Name:
passwordPrompt=Password:
tellUsPrompt=Please tell us about yourself:
aboutYourselfPrompt=Some information about you:
submitPrompt=Submit your information
```

index.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.indexWindowTitle}</title>
   </h:head>
   <h:body>
      <h:outputText value="#{msgs.indexPageTitle}"
                    style="font-style: italic; font-size: 1.5em"/>
      <h:form>
         <h:panelGrid columns="2">
            #{msgs.namePrompt}
            <h:inputText value="#{user.name}"/>
            #{msgs.passwordPrompt}
            <h:inputSecret value="#{user.password}"/>
            #{msgs.tellUsPrompt}
            <h:inputTextarea value="#{user.aboutYourself}" rows="5" cols="35"/>
         </h:panelGrid>
         <h:commandButton value="#{msgs.submitPrompt}" action="thankYou"/>
      </h:form>
   </h:body>
</html>
```

thankYou.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.thankYouWindowTitle}</title>
   </h:head>
   <h:body>
      <h:outputText value="#{msgs.namePrompt}" style="font-style: italic"/>
      #{user.name}
      <br/>
      <h:outputText value="#{msgs.aboutYourselfPrompt}" style="font-style: italic"/>
      <br/>
      <pre>#{user.aboutYourself}</pre>
   </h:body>
</html>
```

web.xml

```
<?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>
```

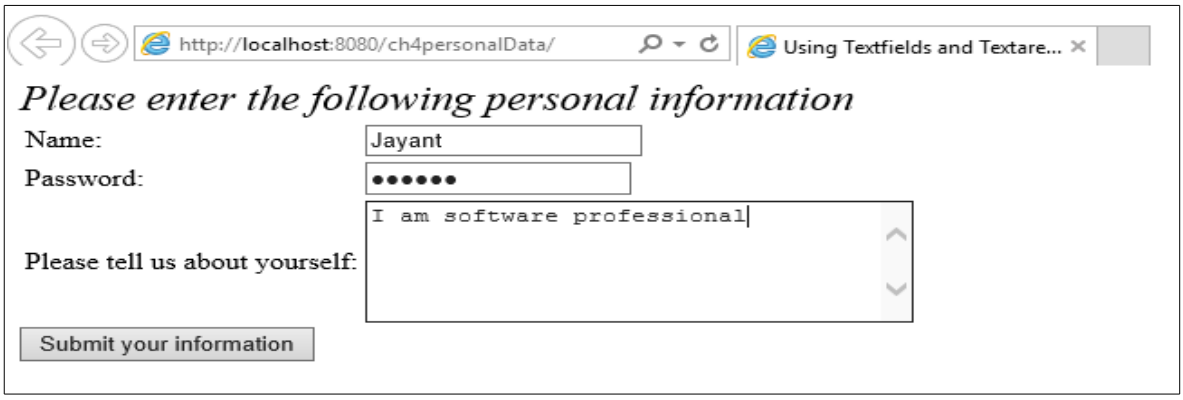
```
      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>/faces/*</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
      <welcome-file>faces/index.xhtml</welcome-file>
   </welcome-file-list>
   <context-param>
      <param-name>javax.faces.PROJECT_STAGE</param-name>
      <param-value>Development</param-value>
   </context-param>
</web-app>
```

faces-config.xml

```
<?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

index.xhtml



thankYou.xhtml

Chapter 5 Facelets

1. Facelets Tags

Facelets Tags

Tag	Description
ui:include	Includes content from another XML file.
ui:composition	When used without a template attribute, a composition is a sequence of elements that can be inserted somewhere else. The composition can have variable parts (specified with ui:insert children). When used with a template attribute, the template is loaded. The children of this tag determine the variable parts of the template. The template contents replaces this tag.
ui:decorate	When used without a template attribute, ui:decorate specifies a page into which parts can be inserted. The variable parts are specified with ui:insert children. When used with a template attribute, the template is loaded. The children of this tag determine the variable parts of the template.
ui:define	Defines content that is inserted into a template with a matching ui:insert.
ui:insert	Inserts content into a template. That content is defined inside the tag that loads the template.
ui:param	Specifies a parameter that is passed to an included file or a template.
ui:component	This tag is identical to ui:composition, except that it creates a component that is added to the component tree.
ui:fragment	This tag is identical to ui:decorate, except that it creates a component that is added to the component tree.
ui:debug	The ui:debug tag lets users display a debug window, with a keyboard shortcut, that shows the component hierarchy for the current page and the application's scoped variables.
ui:remove	JSF removes everything inside of ui:remove tags.
ui:repeat	Iterates over a list, array, result set, or individual object. See Chapter 6.

2. Templating with Facelets `<ui:insert>, <ui:composition>` and `<ui:define>`

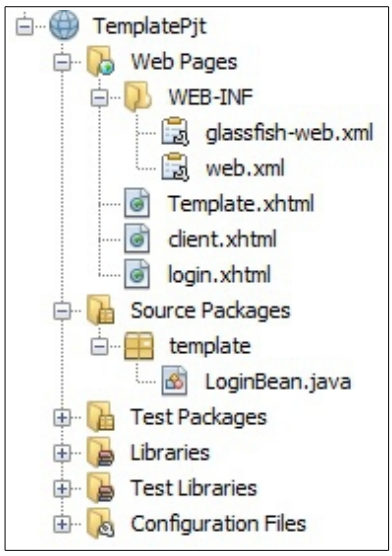
➔ Most of the websites will be have pages with similar layout and styling. They have common header, footer and sidebars etc.

➔ Facelets lets you encapsulate that commonality in a template, so that one can update the site by making changes to template than individual pages.

➔ Template Example

```
<ui:insert>
<ui:composition>
<ui:define>
```

➔ Lets have TemplatePjt directory as shown below



web.xml

```
<?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"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
  <display-name>TemplateProject</display-name>
  <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>
  <context-param>
    <description>State saving method: 'client' or 'server' (=default). See JSF
Specification 2.5.2</description>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
  </context-param>
  <context-param>
    <param-name>javax.servlet.jsp.jstl.fmt.localizationContext</param-name>
    <param-value>resources.application</param-value>
  </context-param>
  <listener>
    <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
  </listener>
  <welcome-file-list>
  <welcome-file>Login.xhtml</welcome-file>
  </welcome-file-list>
</web-app>
```

login.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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:c="http://java.sun.com/jsf/core"
        xmlns:ui = "http://java.sun.com/jsf/facelets"
        xmlns:h = "http://java.sun.com/jsf/html">
```

```
<h:body>
      <h:form>
        Enter name : <h:inputText id="name" value="#{LoginBean.name}"/>
        <h:commandButton id="submit" value="submit" action="client"/>
      </h:form>
</h:body>
</html>
```

template.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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:c="http://java.sun.com/jsf/core"
xmlns:ui = "http://java.sun.com/jsf/facelets"
xmlns:h = "http://java.sun.com/jsf/html">

<h:body>
      <ui:insert name="message"/>
</h:body>
</html>
```

client.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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:c="http://java.sun.com/jsf/core"
xmlns:ui = "http://java.sun.com/jsf/facelets"
xmlns:h = "http://java.sun.com/jsf/html">

<h:body>
      <h:form>
            <ui:composition template = "/template.xhtml">

            <ui:define name = "message">
                  Hello #{LoginBean.name}
            </ui:define>

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

Output

login.xhtml

client.xhtml

➔ Facelets removes all tags *outside* the ui:composition tag—that is, the doctype declaration, html, head, title, and body tags. This is necessary because the ui:composition is replaced with the template that contains its own set of html, head, title, and body tags.

➔ When the template is loaded, each ui:insert is replaced with the contents of the corresponding ui:define.

3. Decorators <ui:decorate>

➔ With decorators first content is designed and then decorated using `<ui:decorate>` tag.

➔ `<ui:decorate>` tag has 'template' attribute and it can be used in the place of `<ui:composition>`.

➔ Difference between `<ui:composition>` and `<ui:decorate>` is, anything outside `<ui:composition>` tag is disregarded but its not true with `<ui:decorate>`. Hence `<ui:decorate>` is beneficial as template-in-template.

4. Parameters <ui:param>

➔ In a template one can supply argument in two ways. One is through <ui:define> and other is through <ui:param>.

➔ While `<ui:define>` provides markup that inserted into the template, In constrast `<ui:param>` sets an EL variable for use in template.

➔ Ex

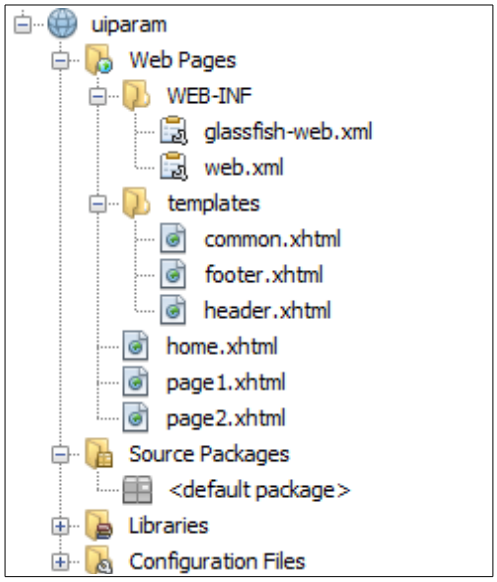
```
<ui:composition template="templates/masterTemplate.xhtml">
      <ui:param name="currentDate" value="#{someBean.currentDate}"/>
</ui:composition>
```

In the corresponding template, you can access the parameter with an EL expression, like this:

```
...
<body>
Today's date: #{currentDate}"/>
</body>
...
```

➔ Ex

Lets create a 'uiparam' project directory as shown below



header.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:ui="http://java.sun.com/jsf/facelets">
   <body>
      <ui:composition>
         <h1>#{defaultHeader}</h1>
      </ui:composition>
   </body>
</html>
```

footer.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:ui="http://java.sun.com/jsf/facelets">
   <body>
      <ui:composition>
         <h1>#{defaultFooter}</h1>
      </ui:composition>
   </body>
</html>
```

common.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"
   xmlns:ui="http://java.sun.com/jsf/facelets">
   <h:head></h:head>
      <h2>#{title}</h2>
      <h:body>
      <div style="border-width:2px; border-color:yellow; border-style:solid;">
         <ui:insert name="header" >
            <ui:include src="/templates/header.xhtml" >
               <ui:param name="defaultHeader" value="Default Header" />
            </ui:include>
         </ui:insert>
      </div>
      <br/>
      <div style="border-width:2px; border-color:black; border-style:solid;">
         <ui:insert name="content" >
            <ui:include src="/templates/contents.xhtml" />
         </ui:insert>
      </div>
      <br/>
      <div style="border-width:2px; border-color:red; border-style:solid;">
         <ui:insert name="footer" >
            <ui:include src="/templates/footer.xhtml">
                <ui:param name="defaultFooter" value="Default Footer" />
            </ui:include>
         </ui:insert>
      </div>
   </h:body>
</html>
```

home.xhtml

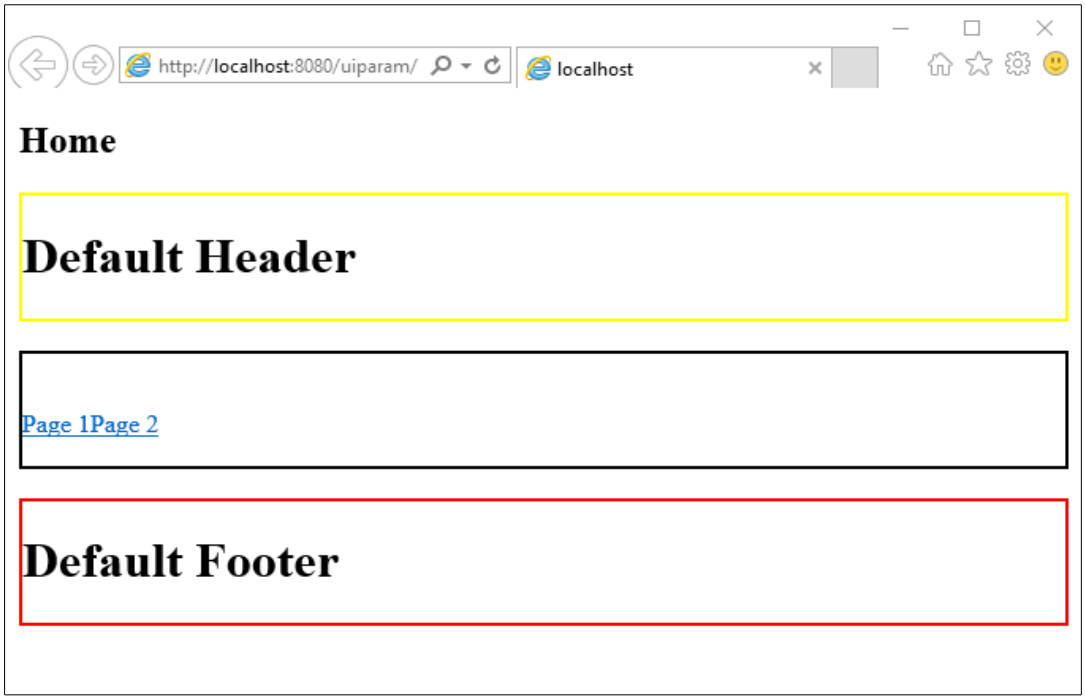
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:ui="http://java.sun.com/jsf/facelets">
```

```
   <h:body>
      <ui:composition template="templates/common.xhtml">
         <ui:param name="title" value="Home" />
         <ui:define name="content">
            <br/><br/>
             <h:link value="Page 1" outcome="page1" />
             <h:link value="Page 2" outcome="page2" />
            <br/><br/>
         </ui:define>
      </ui:composition>
   </h:body>
</html>
```

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/home.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Output



Note : Page1 and Page2 view pages has not been implemented and only kept to enable links.

➔ Here we can see how <ui:param> in two ways

I. As child tag with <ui:include> both for header and footer in common.xhtml file.

Ex `<ui:param name="defaultHeader" value="Default Header" />`

ii. As child tag with <ui:composition> for title

`<ui:param name="title" value="Home" />`

5. Components and Fragements `<ui:component>` and `<ui:fragement>`

➔ <ui:component> tag is similar to <ui:composition> except for two things

I. JSF creates a separate component for it and adds it directly to the component tree.

ii. There is no associated template.

➔ <ui:component> tag is used to create component and specify a file name for the component as either the source of a <ui:include> (as done in the following example) or the source of a Facelets tag.

➔ <ui:compostion> is used to define template layout but <ui:component> is encapsulate definition of a facelet component.

➔ It has `id`, `binding` and `rendered` attribute. If '`binding`' attribute has been used then one can programmatically manipulate the component.

➔ Also one can conditionally render the component by setting '`rendered`' attribute to a value expression(which is a boolean value). If its false then component is not rendered in the page.

➔ Any markup occurring outside of the `<ui:component>` tag is not included in the view.

➔ Similarly <ui:fragement> is same as <ui:component> it only includes tags occurring outside itself.

6. <ui:debug> tag

➔ If <ui:debug> tag added to a facelet page then a debug component will be added to the component tree of that page.

➔ If user types hotkey, which is by default CTRL+SHIFT+d, the JSF open a window and displays the state of the component tree and applicatoin's scoped variables.

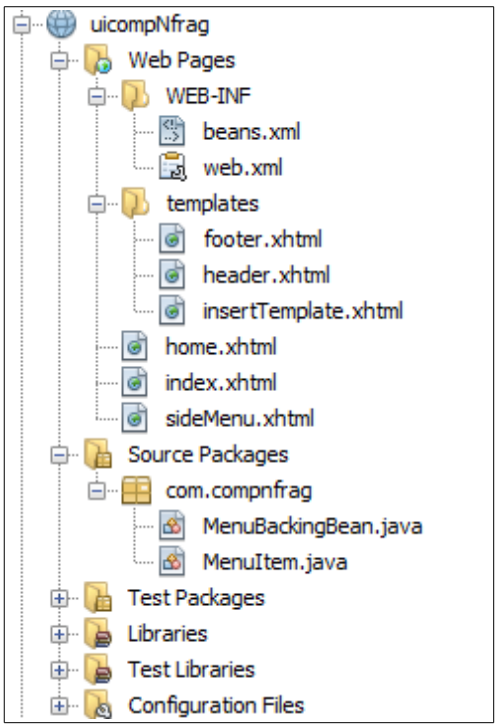
➔ One can change the hotkey using attribute 'hotkey'

Ex `<ui:debug hotkey="i"/>`

then the hotkey becomes CTRL+SHIFT+i.

➔ Ex uses <ui:component>, <ui:fragement>, <ui:debug>

Lets create project directory for project 'uicompNfrag' as shown below



header.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:ui="http://java.sun.com/jsf/facelets">
   <body>
      <div style="width:100%;line-height:48px;background-color:#336699;color:white; font-
size:30px">
      JSF Facelets Application</div>
   </body>
</html>
```

footer.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:ui="http://java.sun.com/jsf/facelets">
   <head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Insert title here</title>
</head>
<body>
<div style="background-color: #336699; width: 100%; color: #FFFFFF">@copyright,RISK
soft</div>
</body>
</html>
```

sideMenu.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:ui="http://java.sun.com/jsf/facelets"
         xmlns:h="http://java.sun.com/jsf/html"
         xmlns:c="http://xmlns.jcp.org/jsp/jstl/core">
This text will not be rendered.<br/>
```

```
<ui:component id="comp">
      <c:forEach var="menu" items="#{menuBackingBean.menus}">
           <a href="#{menu.url}">#{menu.label}</a><br/>
      </c:forEach>
</ui:component>
</html>
```

insertTemplate.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html">
<head>
<title>Facelets example</title>
</head>
<body >
<div style="border-bottom: grey 2px solid; border-left: grey 2px solid; border-right: grey
2px solid; border-top: grey 2px solid; height: 100%; margin: 4px auto; text-align: center;
width: 100%;">
<ui:insert name="header">
       <ui:include src="/templates/header.xhtml" />
</ui:insert></div>
<table width="100%" >
        <tr>
                <td width="20%">
                <div style="height: 250px; width: 100%; background-color: #e0e0e0; text-
align: center;">
                 <br></br>
                <ui:insert name="sidemenu"/>

                 </div>
                 </td>
                 <td width="85%">
                 <ui:insert name="content">Content displayed from Template </ui:insert>
                 </td>
                 </tr>
</table>
<div style="border-bottom: grey 2px solid; border-left: grey 2px solid; border-right: grey
2px solid; border-top: grey 2px solid; height: 100%; margin: 4px auto; text-align: center;
width: 100%;">
<ui:insert name="footer">
       <ui:include src="/templates/footer.xhtml" />
</ui:insert></div>
    <ui:debug hotkey="i"/><!--hotkey is placed here -->
</body>
</html>
```

home.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
         xmlns:h="http://java.sun.com/jsf/html"
         xmlns:f="http://java.sun.com/jsf/core">
<ui:composition template="templates/insertTemplate.xhtml">
<ui:define name="sidemenu">
        <ui:include src="sideMenu.xhtml" />
</ui:define>
<ui:define name="content">
        This is an example of a simple Facelets template.<br/>
        Here Header and Footer appears from template.(inserTemplate.xhtml)<br/>
        Side Menu appears from sideMenu.xhtml<br/>
        This section appears from templateClient.(home.xhtml)
</ui:define>
</ui:composition>
</html>
```

MenuItem.java

```
package com.compnfrag;

public class MenuItem {
    private String url;
    private String label;
    public String getUrl() {
       return url;
    }
    public void setUrl(String url) {
        this.url = url;
    }
    public String getLabel() {
        return label;
    }
    public void setLabel(String label) {
        this.label = label;
    }
    public MenuItem() {
        super();
    }
    public MenuItem(String url, String label) {
        super();
        this.url = url;
        this.label = label;
    }
}
```

MenuBackingBean.java

```
package com.compnfrag;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collection;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;

@Named("menuBackingBean")
@SessionScoped
public class MenuBackingBean implements Serializable {
     private Collection<MenuItem> menus;
     public Collection<MenuItem> getMenus() {
         return menus;
     }
     public void setMenus(Collection<MenuItem> menus) {
         this.menus = menus;
     }
     public MenuBackingBean() {
         super();
         menus = new ArrayList<>();
         menus.add(new MenuItem("home.xhtml", "Home"));
         menus.add(new MenuItem("news.xhtml", "News"));
         menus.add(new MenuItem("articles.xhtml", "Articles"));
         menus.add(new MenuItem("about.xhtml", "About Us"));
     }
}
```

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/home.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

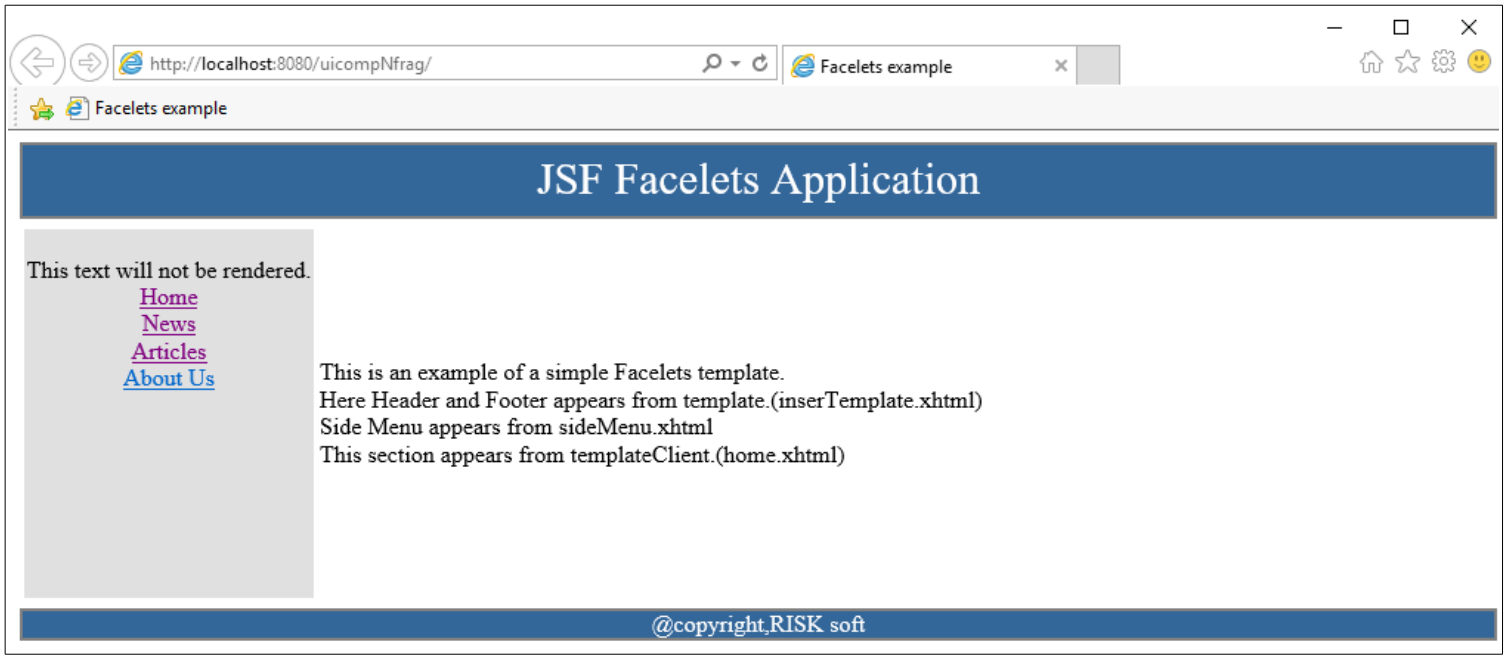
Output

1. Without <ui:component> tag

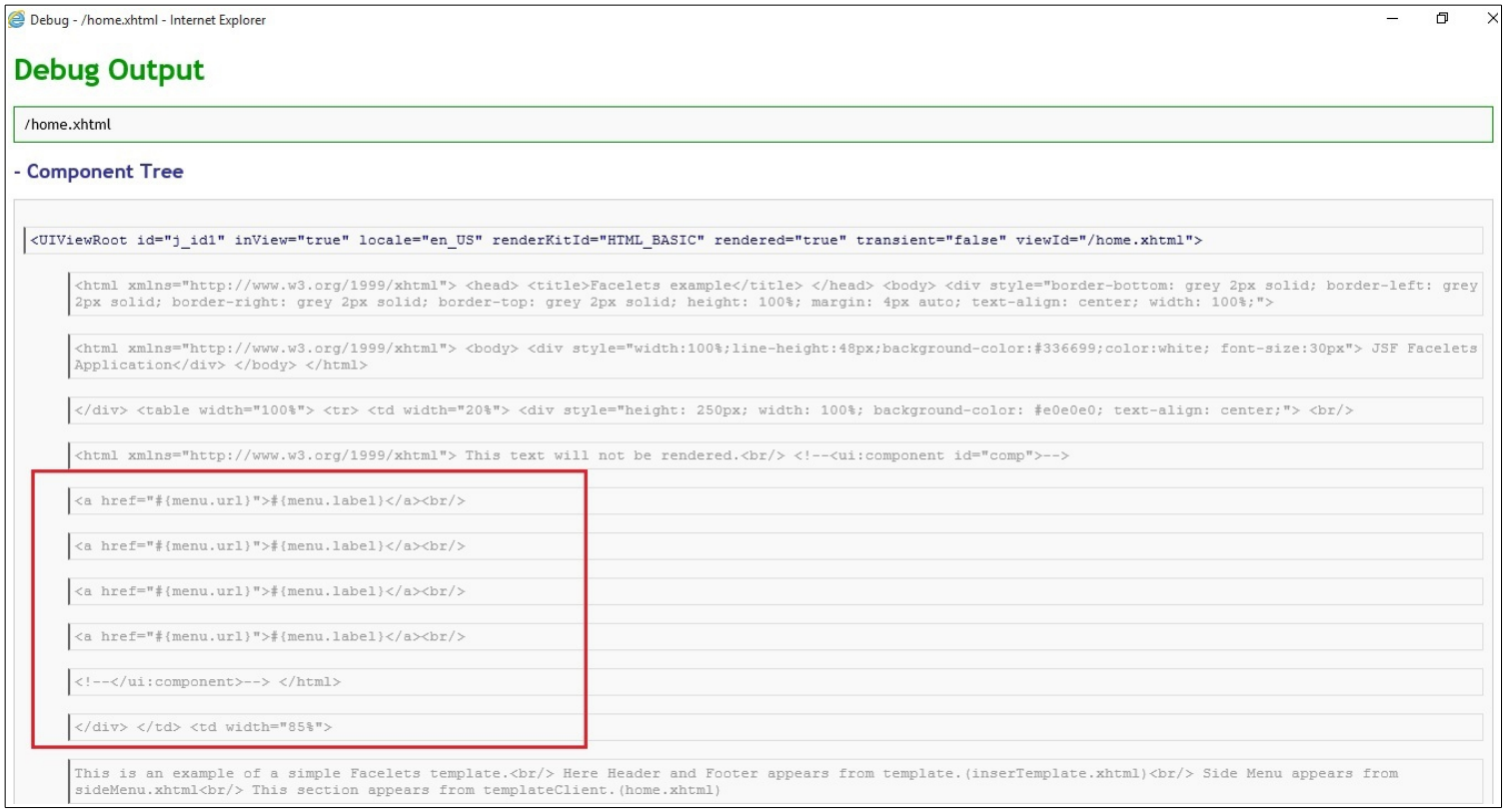
When <ui:component> tag is commented out inside sideMenu.xhtml as shown below

```
This text will not be rendered.<br/>
<ui:component id="comp">
      <c:forEach var="menu" items="#{menuBackingBean.menus}">
           <a href="#{menu.url}">#{menu.label}</a><br/>
      </c:forEach>
</ui:component>
```

home.xhtml



Debug

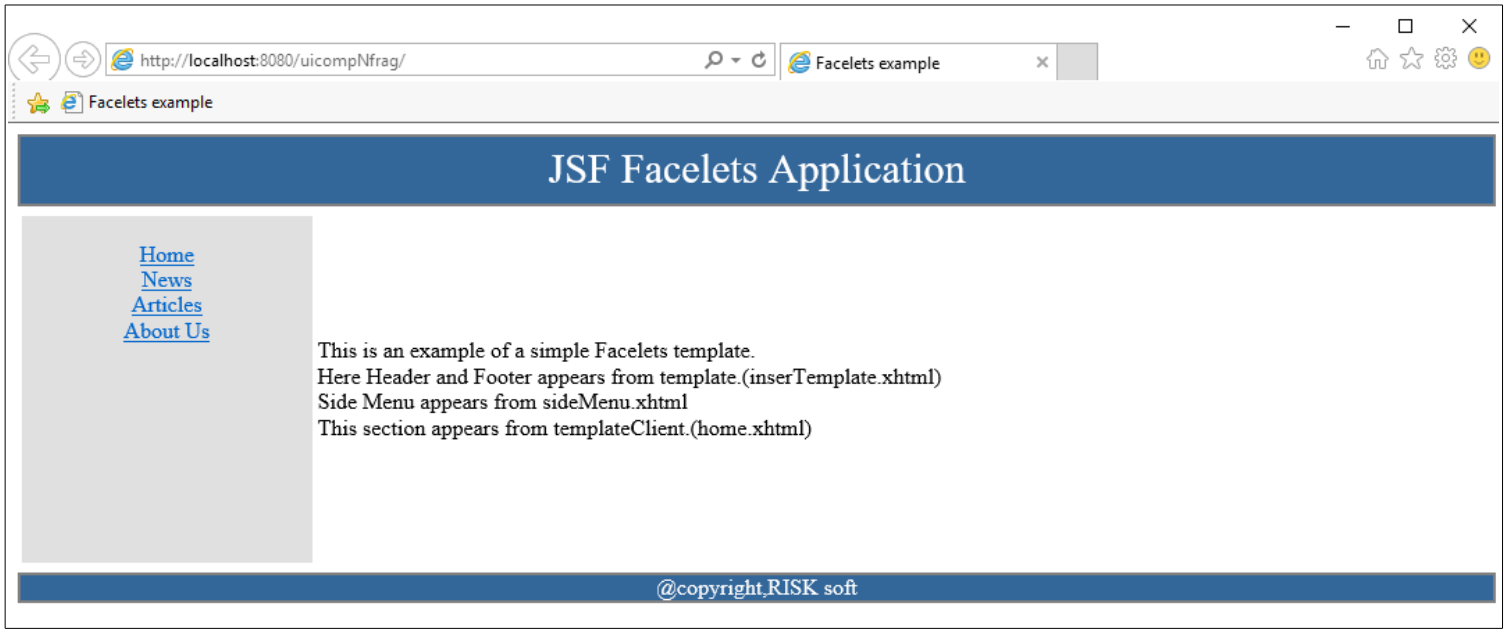


Note : Above we can see as <ui:component> tag is commented out they content of sideMenu.xhtml has appeared with other contents and not as separate component.

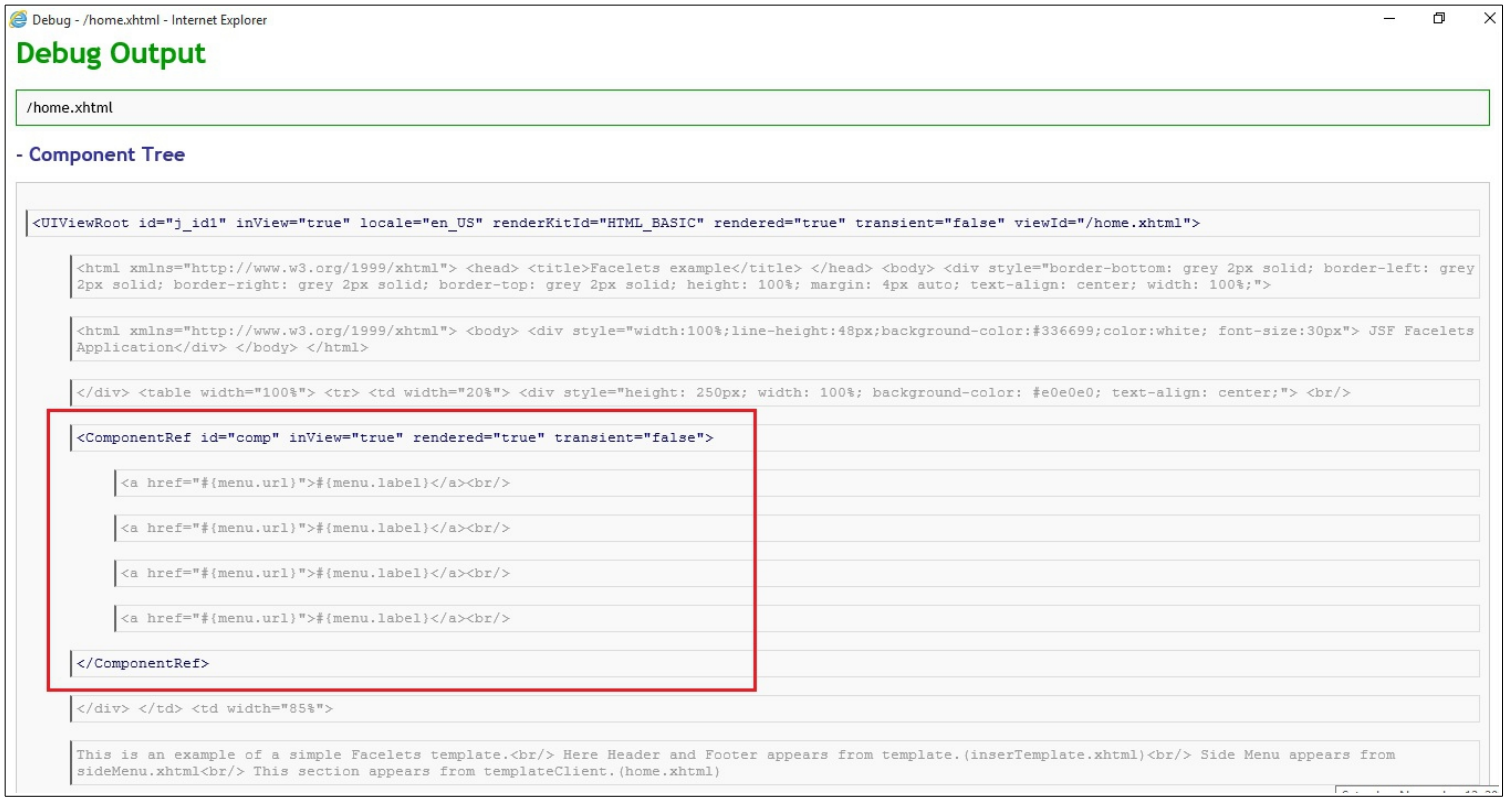
2. With <ui:component> tag

```
This text will not be rendered.<br/>
<ui:component id="comp">
      <c:forEach var="menu" items="#{menuBackingBean.menus}">
           <a href="#{menu.url}">#{menu.label}</a><br/>
      </c:forEach>
</ui:component>
```

home.xhtml



Debug



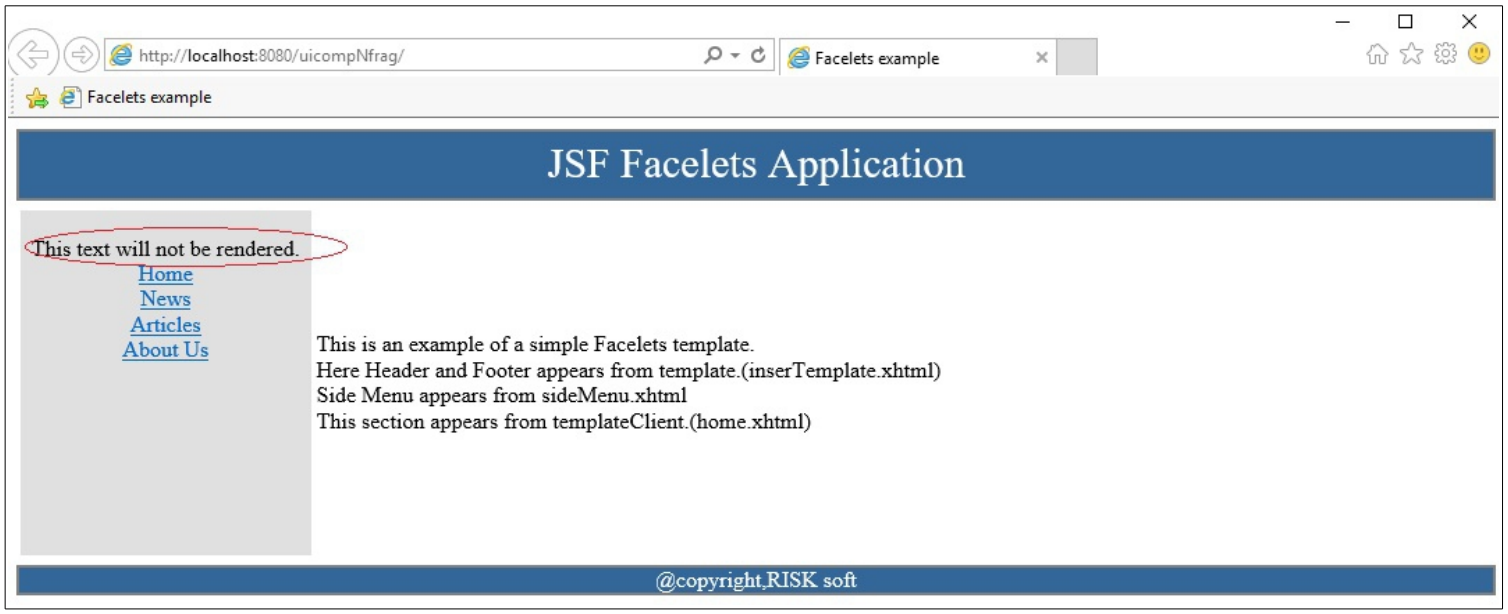
Note: Here we can note that <ui:component> is seen as a separate component in debug window.

3. With <ui:fragement>

Lets replace <ui:component> in sideMenu.xhtml with <ui:fragement> tag as shown below

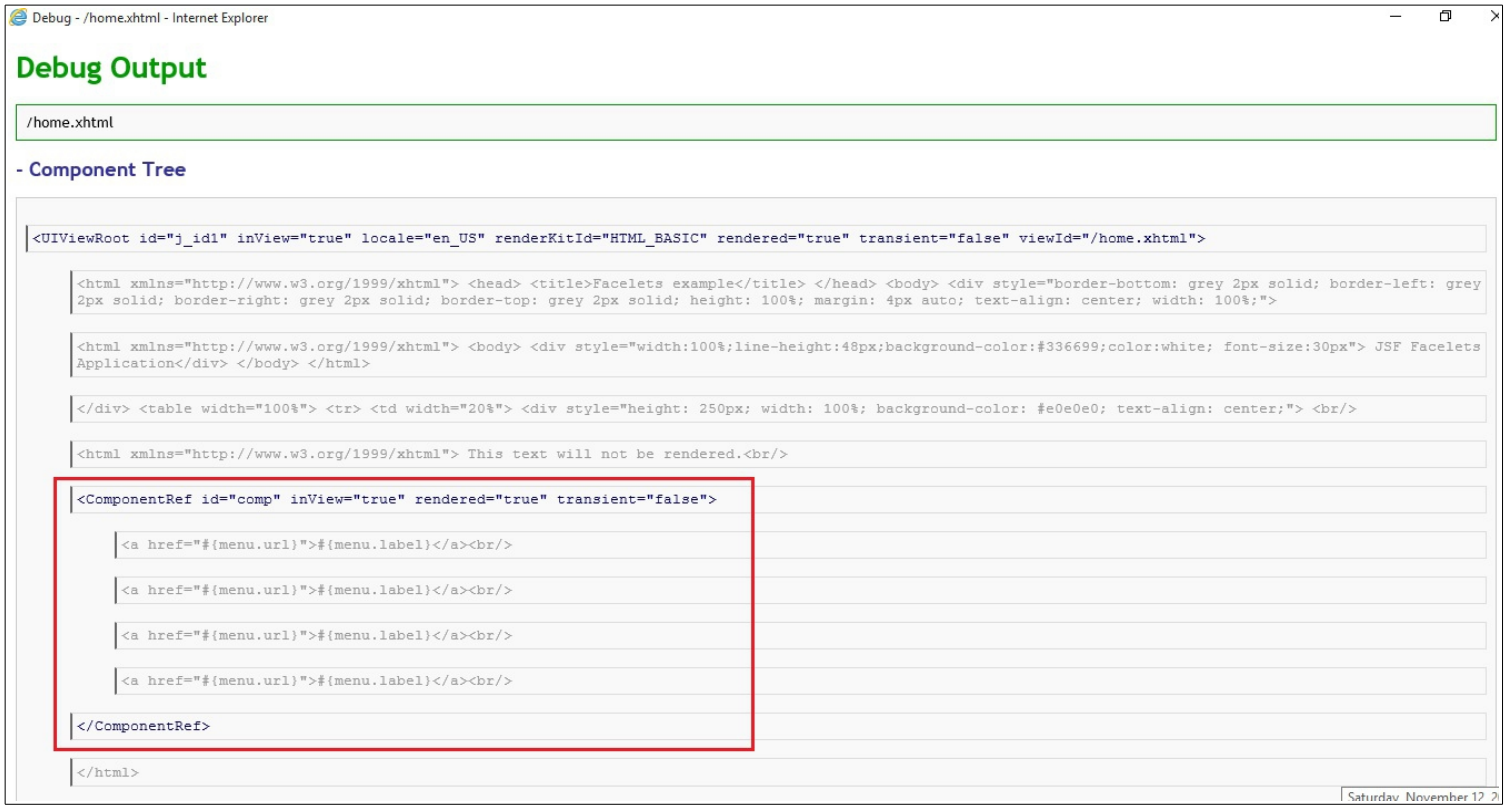
```
<ui:fragement id="comp">
      <c:forEach var="menu" items="#{menuBackingBean.menus}">
            <a href="#{menu.url}">#{menu.label}</a><br/>
      </c:forEach>
</ui:fragement>
```

home.xhtml(same as without component)



Note: Here contents outside <ui:fragement> is also got displayed.

Debug(Same as with Component)



7. <ui:remove> tag

➔ <ui:remove> removes content from a page i.e. dont render content.

➔ XML comment <!-- --> will not be useful as if it contains a 'value expression' then that will be rendered.

Ex

```
<!-- <h:commandButton id="loginButton"
value="#{msgs.loginButtonText}"
action="planetarium"/> -->
```

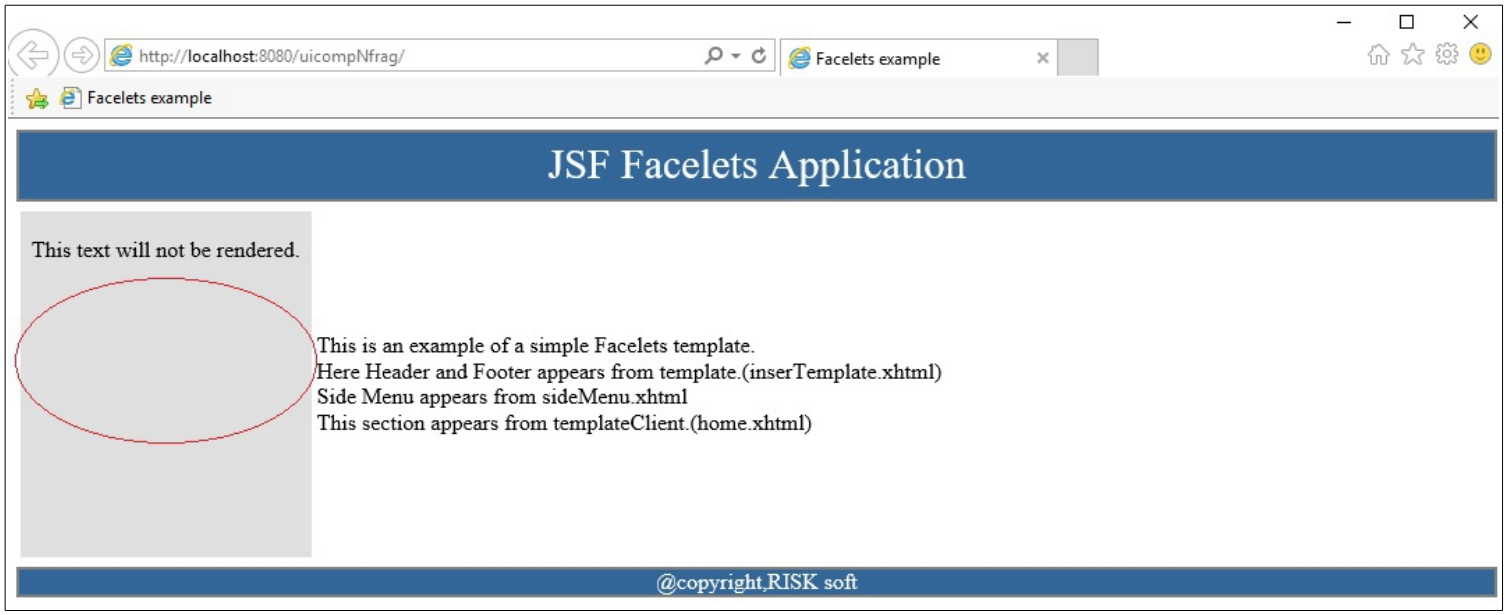
will be rendered as

```
<!-- <h:commandButton id="loginButton"
value="Log In"
action="planetarium"/> -->
```

and if loginButtonText method will throw an exception then it will be a problem. Hence <ui:remove> is the best solution.

➔ If in above example we use <ui:remove> inside sideMenu.xhtml as shown below

```
This text will not be rendered.<br/>
<ui:remove>
      <c:forEach var="menu" items="#{menuBackingBean.menus}">
            <a href="#{menu.url}">#{menu.label}</a><br/>
      </c:forEach>
</ui:remove>
```



8. Handling Whitespace

➔ Be default in JSF white spaces are trimmed around components. For example

```
<h:outputText value="#{msgs.name}"/>
<h:inputText value="#{user.name}"/>
```

Here above two will be separated by white space. The facelets wont turn that white space into the text component.

➔ But same is not true for two links in a row

```
<h:commandLink value="Previous" .../> <h:commandLink value="Next" .../>
```

This will turn into PreviousNext. Hence to provide white space one use value expression with a space #{' '}.

Chapter 7 Conversion and Validation

1. Introduction

➔ Lets look at how user input(form data) travels from browser to bean

1. Browser sends user input to the server using HTTP request. These are called request values.

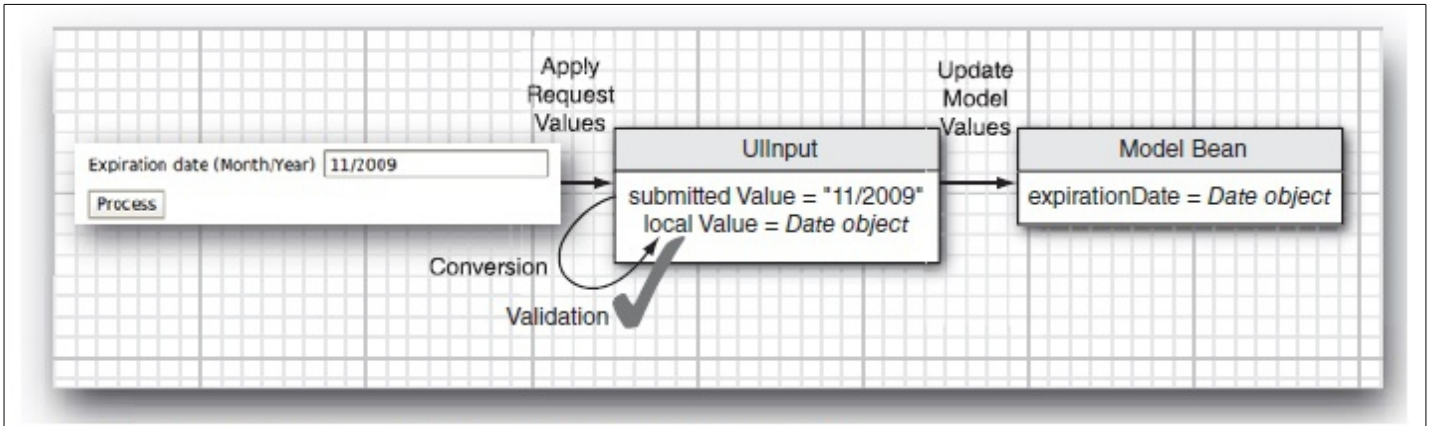
2. Each input tag in JSF implementation has corresponding component object. In 'Apply Request Value Phase' these request values gets stored in component objects. Here they are called submitted values.

3. All request values from browser are Strings and bean properties could be int, String or any other type. Hence these values needs to be converted to a particular type and validated. Both conversion and validation processes starts with 'Process Validation' phase.

4. Once the values are converted they are not directly transferred to the bean. Before that they needed to be validated. Hence they are stored in component objects as local values. Application designers provides validation conditions.

5. After all validations are successful 'Update Model Values' phase will starts and local values are stored in beans, as specified by their value references.

Example



A value travels from the browser to the model

2. Using Standard Converters

converter	Adds an arbitrary converter to a component.
convertDateTime	Adds a datetime converter to a component.
convertNumber	Adds a number converter to a component.

➔ Web application stores data in many types but web user interface deals exclusively with Strings. For example, Suppose user needs to edit a data object stored in business logic. For that first the date object will be converted to String and then sent to web browser to be displayed inside the text field. After user edited the text field, the resulting String returns to the server and must be converted to the Date object.

➔ There are three ways one can use converters

i. Using particular conversion tags

```
<f:convertNumber>
```

```
<f:convertDateTime>
```

➔ Supporse we have a shopping cart application where customer needs to fill amount, credit card number and valid date for the card. Then the converters are used in customerDetails.xhtml page are shown below

```
<h:inputText value="#{payment.amount}">
<f:convertNumber minFractionDigits="2"/>
</h:inputText>
```

and

```
<h:inputText value="#{payment.date}">
<f:convertDateTime pattern="MM/yyyy"/>
</h:inputText>
```

In the result.xhtml page we can again use converter for 'amount' as shown below

```
<h:outputText value="#{payment.amount}">
<f:convertNumber type="currency"/>
</h:outputText>
```

The output are as shown below

➔ If value expression of primitive type, enumerated type or BigInteger/BigDecimal type then one dont need to provide converter as JSF implementation automatically picks a standard converter.

➔ <f:convertNumber> and <f:convertDateTime> attributes

Attributes of the f:convertNumber Tag

Attribute	Type	Value
type	String	number (default), currency, or percent
pattern	String	Formatting pattern, as defined in java.text.DecimalFormat
maxFractionDigits	int	Maximum number of digits in the fractional part
minFractionDigits	int	Minimum number of digits in the fractional part
maxIntegerDigits	int	Maximum number of digits in the integer part
minIntegerDigits	int	Minimum number of digits in the integer part
integerOnly	boolean	True if only the integer part is parsed (default: false)
groupingUsed	boolean	True if grouping separators are used (default: true)
locale	java.util.Locale or String	Locale whose preferences are to be used for parsing and formatting
currencyCode	String	ISO 4217 currency code, such as USD or EUR, for selecting a currency converter
currencySymbol	String	This string is passed to DecimalFormat.setDecimalFormatSymbols, overriding the locale-based symbol; not recommended—use currencyCode instead

Attributes of the f:convertDateTime Tag

Attribute	Type	Value
type	String	date (default), time, or both
dateStyle	String	default, short, medium, long, or full
timeStyle	String	default, short, medium, long, or full
pattern	String	Formatting pattern, as defined in java.text.SimpleDateFormat
locale	java.util.Locale or String	Locale whose preferences are to be used for parsing and formatting
timeZone	java.util.TimeZone	Time zone to use for parsing and formatting; if you do not supply a time zone, the default is GMT Note: As of JSF 2.0, you can change the default to TimeZone.getDefault() by setting javax.faces.DATETIMECONVERTER_DEFAULT_TIMEZONE_IS_SYSTEM_TIMEZONE to true in web.xml.

ii. Using 'converter' attribute

➔ One can use 'converter' attribute available with the tags.

```
<h:outputText value="#{payment.date}" converter="javax.faces.DateTime"/>
```

which is equivalent to

```
<h:outputText value="#{payment.date}">
<f:convertDateTime/>
</h:outputText>
```

iii. Use <f:converter> tag

```
<h:outputText value="#{payment.date}">
<f:converter converterId="javax.faces.DateTime"/>
</h:outputText>
```

➔ All JSF implementations must define a set of converters with predefined IDs as shown below

- `javax.faces.DateTime` (used by `f:convertDateTime`)
- `javax.faces.Number` (used by `f:convertNumber`)
- `javax.faces.Boolean, javax.faces.Byte, javax.faces.Character, javax.faces.Double, javax.faces.Float, javax.faces.Integer, javax.faces.Long, javax.faces.Short` (automatically used for primitive types and their wrapper classes)
- `javax.faces.BigDecimal, javax.faces.BigInteger` (automatically used for BigDecimal/BigInteger)

2.1 Conversion Errors

➔ When the conversion error occurs, JSF implementation carries out the following actions.

i. Component whose conversion fails, posts a message and declares itself invalid.

ii. The JSF implementation re-displays the current page immediately after 'Process Validations' phase has completed. The redisplayed page contains all the values that user provided – no user input is lost.

➔ Whenever user provides an illegal input like for integer value String has been entered etc. The JSF implementtaion automatically re-displays the current page giving the user chance to re-enter the correct values.

2.3. Displaying Error Messages

➔ By default JSF implementation displays error messages whenever conversion error occurs.

➔ For example if we have code

```
<h:outputLabel value="Amount :"/>
   <h:inputText id="amt" value="#{customer.amount}">
      <f:convertNumber minFractionDigits="2"/>
   </h:inputText>
```

'amout' is of type int in customer bean and if 'ten' has been entered then the JSF implementation re-displays current page with an error message at the bottom 'j_idt5:amt: 'ten' is not a number.'

➔ One can customize error messages of a page using `<h:message>` and `<h:messages>` tags.

➔ If one want to display error message next to the component that reported them then give an ID to that component and reference that ID in `<h:message>` as shown below.

```
<h:outputLabel value="Amount :"/>
                    <h:inputText id="amt" label=”Amount is invalid”
```

```
        value="#{customer.amount}">
         <f:convertNumber minFractionDigits="2"/>
    </h:inputText>
    <h:message for="amt"/>
```

Error message will be 'Amount is Invalid: 'ten' is not a number. Example: 99'

➔ JSF implementation shows default(standard) error message because child tag <h:messages/> will be automatically added to the view if the project stage has been set to development(web.xml).

➔ Error messages actually has two parts 'summary' and 'detail'. By default, <h:message> displays 'detail' and hides the summary and <h:messages> displays summary and hides the detail.

➔ In the above example <h:message> is showing 'detail' error message. 'detail' error message shows the label of the component, the offending value, and a sample of a correct value.(note: 'label' has been added in <h:inputText> tag).

➔ 'summary' message omits the sample.

➔ <h:message> and <h:messages> have attributes 'showDetails' and 'showSummary'. Both these attributes are of type Boolean and one can display summary or detail for an error message by setting their values to true or false. Example

```
<h:outputLabel value="Amount :"/>
<h:inputText id="amt" label="Amount is Invalid" value="#{customer.amount}">
   <f:convertNumber minFractionDigits="2"/>
</h:inputText>
<h:message for="amt" showSummary="true" showDetail="false"/>
```

The error message will be 'Amount is Invalid: 'ten' is not a number.'

➔ One can even use the color of an error message using 'style' attribute.

```
<h:message for=”amt” style="color: red"/>
```

➔ One can display more than one error message using <h:messages> tag. It has 'layout' attribute which could be list or table. Default is list.

2.4. Changing Standard Error Message text

➔ Standard Conversion Error Messages are organized in key/value pairs. One example of standard error message is shown below. Key(Resource ID) and value(default value) are shown.

javax.faces.converter.NumberConverter.CURRENCY_detail	{2}: "{0}" could not be understood as a currency value. *Example: {1}*

Above key is for 'detail' message therefore value has 'Example:{1}' part.

➔ To replace standard message, one should set up a message bundle. Add the replacement message to the bundle with the appropriate key(shown in left in the pic above). Then set the base name of the bundle in configuration(faces-config.xml) file as shown below

```
<faces-config>
      <application>
            <message-bundle>com.corejsf.errmsgs</message-bundle>
      </application>
...
</faces-config>
```

➔ One need to only specify the messages that one need to override.

➔ Note that this message bundle is not same as <resource-bundle> which is used to map variables using value expression.

2.5. Using Custom Messages

➔ One can provide custom converter error message for a component using '`converterMessage`' attribute.

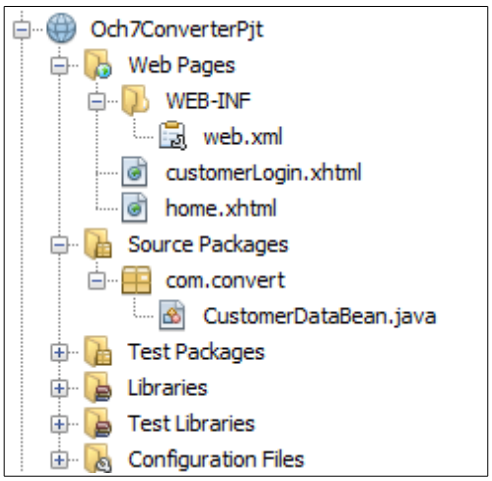
➔ Example

```
<h:outputLabel value="Amount :"/>
<h:inputText id="amt" value="#{customer.amount}" converterMessage="This is not valid
input">
<f:convertNumber minFractionDigits="2"/>
</h:inputText>
```

➔ When '`converterMessage`' is used then detail or summary wont be displayed even 'label' attribute is present and

<`h:message`> tag is present. But it do take '`style`' declared in <`h:message`>.

➔ Lets create a project 'Och7ConverterPjt' as shown below



CustomerBean.java

```
package com.convert;

import java.io.Serializable;
import java.util.Date;
import javax.inject.Named;
import javax.enterprise.context.SessionScoped;

@Named(value = "customer")
@SessionScoped
public class CustomerDataBean implements Serializable {

    public CustomerDataBean() {
    }

    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    public String getCcno() {
        return ccno;
    }

    public void setCcno(String ccno) {
        this.ccno = ccno;
    }
```

```
    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    double amount;
    String ccno;
    Date date;
}
```

customerLogin.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>Customer Login</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="3">

                <h:outputLabel value="Amount :"/>
                <h:inputText id="amount" label="This Is Invalid Input"
                      value="#{customer.amount}">
                         <f:convertNumber minFractionDigits="2"/>
                </h:inputText>
                <h:message for="amount" showSummary="true" showDetail="false"
                style="color: red"/>

                <h:outputLabel value="Credit Card Number :"/>
                <h:inputText id="ccno" value="#{customer.ccno}"/>
                <h:message for="ccno" style="color: red"/>

                <h:outputLabel value="Valid Date :"/>
                <h:inputText id="date" value="#{customer.date}">
                    <f:convertDateTime pattern="mm/yyyy"/>
                </h:inputText>
                <h:message for="date" style="color: red"/>
            </h:panelGrid>
            <h:commandButton id="submit" value="submit" action="home"/>
        </h:form>
    </h:body>
</html>
```

home.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>Home Page</title>
    </h:head>
    <h:body>

        <h:outputLabel value="Hello User"/><br/>
        <h:outputLabel value="Your Details"/><br/>
        <h:panelGrid columns="2">
```

```
        <h:outputLabel value="Amount :"/>
        <h:outputText value="#{customer.amount}"/>
        <ui:remove> <!-- <f:convertNumber type="currency"/>
        </h:outputText>--></ui:remove>
        <h:outputLabel value="Credit Card Number  :"/>
        <h:outputText value="#{customer.ccno}"/>
        <h:outputLabel value="Valid upto :"/>
        <h:outputText value="#{customer.date}"/><br/>

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

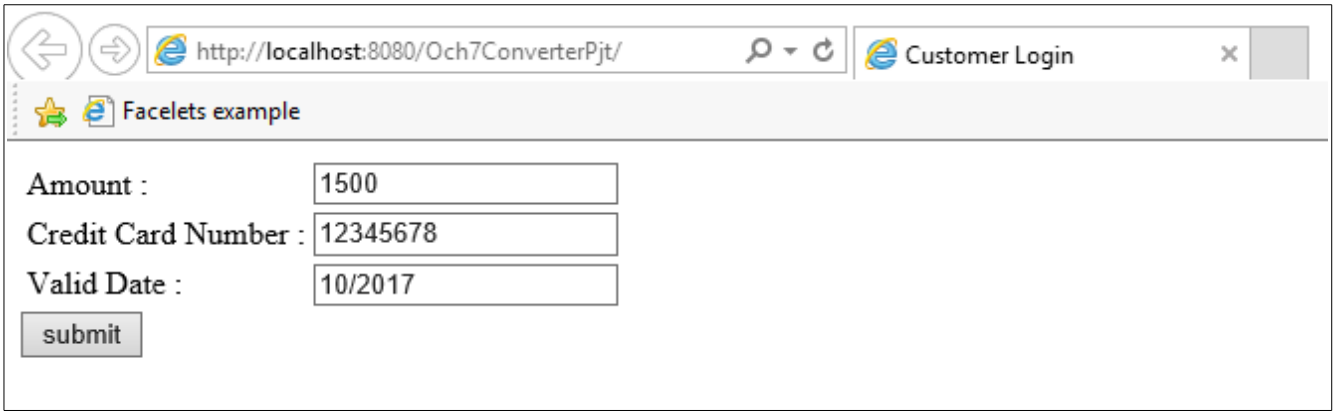
web.xml

```
<?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/customerLogin.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Output

1. When valid input values are entered

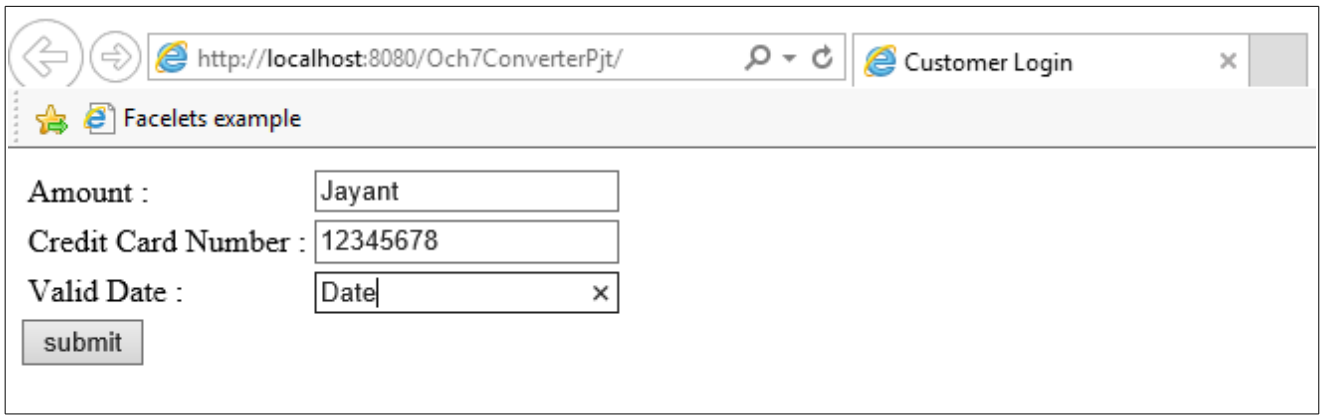
customerLogin.xhtml



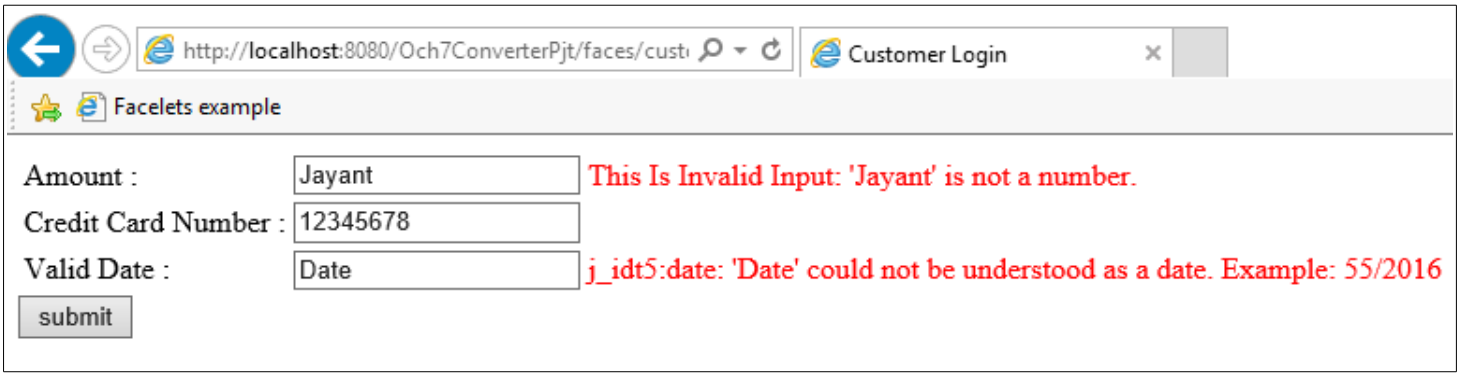
home.xhtml

2. When Invalid values have been entered

customerLogin..xhtml

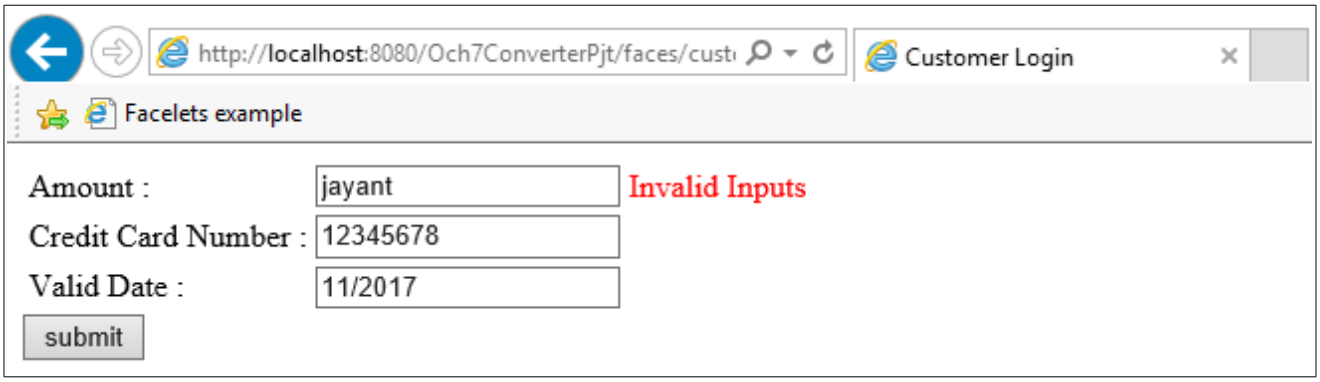


home.xhtml



3. If we use 'converterMessage'

```
<h:outputLabel value="Amount :"/>
      <h:inputText id="amount" label="This Is Invalid Input" value="#{customer.amount}"
converterMessage="Invalid Inputs">
            <f:convertNumber minFractionDigits="2"/>
      </h:inputText>
<h:message for="amount" showSummary="true" showDetail="false" style="color: red"/>
```



Note : Above we can see error message in '`converterMessage`' has overridden all other messages but has taken red color from '`style`' defined inside `<h:message>`.

3. <u>Using Standard Validators</u>

➔ The list of Standard Validators in JSF is shown below

➔ JSF has build in mechanism which let one carry out the following validations.

a. Checking the length of a string

b. Checking limits for a numerical value (for example, > 0 or ≤ 100)

c. Checking against a regular expression (since JSF 2.0)

Standard Validators

JSP Tag	Validator Class	Attribute[a]	Validates
f:validateDoubleRange	DoubleRangeValidator	minimum, maximum	A double value within an optional range
f:validateLongRange	LongRangeValidator	minimum, maximum	A long value within an optional range
f:validateLength	LengthValidator	minimum, maximum	A String with a minimum and maximum number of characters
f:validateRequired JSF 2.0	RequiredValidator		The presence of a value
f:validateRegex JSF 2.0	RegexValidator	pattern	A String against a regular expression
f:validateBean JSF 2.0	BeanValidator	validation-Groups	Specifies validation groups for bean validators (see the JSR 303 specification for details)

d. Checking that a value has been supplied

➔ Some of the validation examples are shown below

1. Validation for an Input field

```
<h:inputText id="ccno" value="#{customer.ccno}"/>
   <f:validateLength minimum="13"/>
</h:inputText>
```

The above code adds validator to a text field. When form is submitted validator makes sure that String contains minimum 13 characters. Otherwise it will generate error messages associated with the guilty component. The message text later can be displayed using `<h:message>` and `<h:messages>` tags.

2. Validating a long range

```
<h:inputText id="amount" value="#{customer.amount}">
   <f:validateLongRange minimum="10" maximum="10000"/>
</h:inputText>
```

The validator checks that supplied value is ≥ 10 and ≤ 10000.

3. To check that a value is supplied or not

```
<h:inputText id="date" value="#{customer.date}">
   <f:validateRequired/>
</h:inputText>
```

'`ValidateRequired`' simply set '`required`' attribute of the enclosing component to true. Hence alternatively one can also use

```
<h:inputText id="date" value="#{customer.date}" required="true"/>
```

➔ An alternative way to attach validation is through `<f:validator>` tag. Here one needs to supply Validator ID and parameters as shown below

```
<h:inputText id="ccno" value="#{customer.ccno}"/>
   <f:validator validatorId="javax.faces.validator.LengthValidator">
   <f:attribute name="ccno" value="13"/>
   </f:validator>
</h:inputText>
```

3.1. Displaying Validation Errors

➔ Displaying message for validation is similar to the displaying conversion errors. A message is added to the component that failed validation, the component is invalidated and the current page is redisplayed immediately after 'Process Validation' phase has completed.

➔ One can use `<h:message>` and `<h:messages>` tags to display messages.

➔ To override standard validation messages one can create message bundle and follow the process defined during standard conversion error overriding.

3.2. Bypassing Validation

➔ Sometimes its necessary to bypass validation process. For example suppose cancel button on the page has 'required' field. So when user clicks on cancel button same page will be redisplayed with the validation error messages.

➔ To bypass it if command has 'immediate' attribute set then command is executed during 'Apply Request Value' phase.

```
<h:commandButton value="Cancel" action="cancel" immediate="true"/>
```

➔ Ex. Lets change customerLogin.xhtml page to as shown below

customerLogin.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>Customer Login</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="3">

                <h:outputLabel value="Amount :"/>
                <h:inputText id="amount" label="This Is Invalid Input"
                      value="#{customer.amount}" required="true">
                         <f:convertNumber minFractionDigits="2"/>
                <f:validateDoubleRange minimum="10" maximum="10000"/>
                </h:inputText>
                <h:message for="amount" showSummary="true" showDetail="false"
                style="color: red"/>

                <h:outputLabel value="Credit Card Number :"/>
                <h:inputText id="ccno" value="#{customer.ccno}" required="true"
                                    requiredMessage="Credit Card No is Required">
                  <f:validateLength minimum="8"/>
                </h:inputText>
                <h:message for="ccno" style="color: red"/>

                <h:outputLabel value="Valid Date :"/>
                <h:inputText id="date" value="#{customer.date}">
                    <f:convertDateTime pattern="mm/yyyy"/>
                </h:inputText>
                <h:message for="date" style="color: red"/>
            </h:panelGrid>
            <h:commandButton id="submit" value="submit" action="home"/>
            <h:commandButton value="Cancel" action="canceled" immediate="true"/>
        </h:form>
    </h:body>
</html>
```

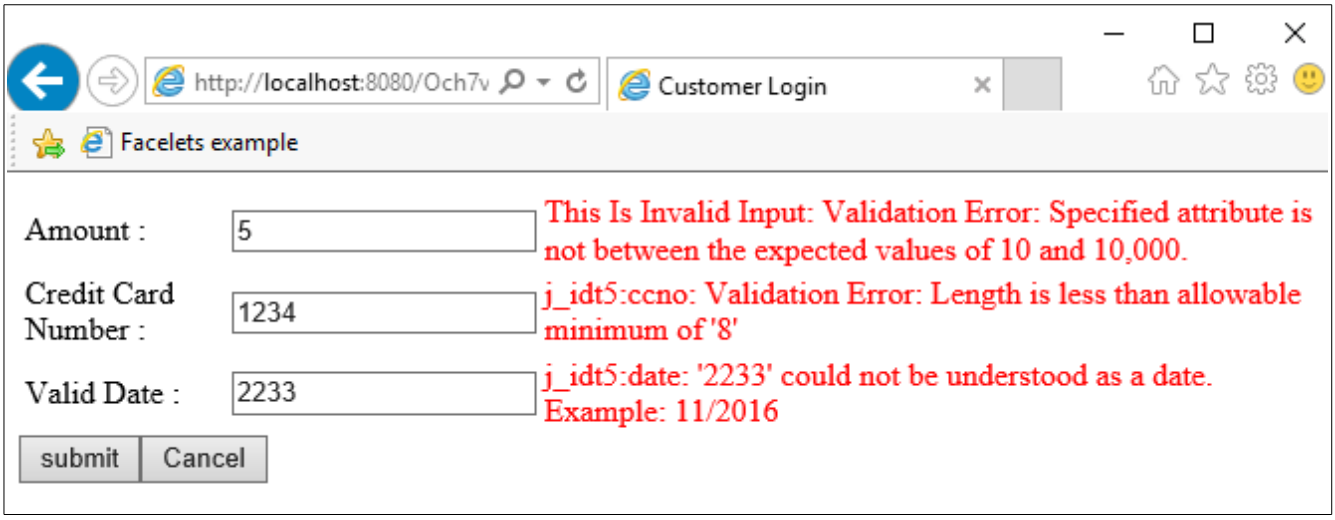
canceled.xhtml

```
<?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>
      <title>Canceled Page</title>
   </h:head>
   <h:body>
      <h:form>
         The transaction has been canceled.<br/>
         <h:commandButton value="Back" action="customerLogin"/>
      </h:form>
   </h:body>
</html>
```

Output:

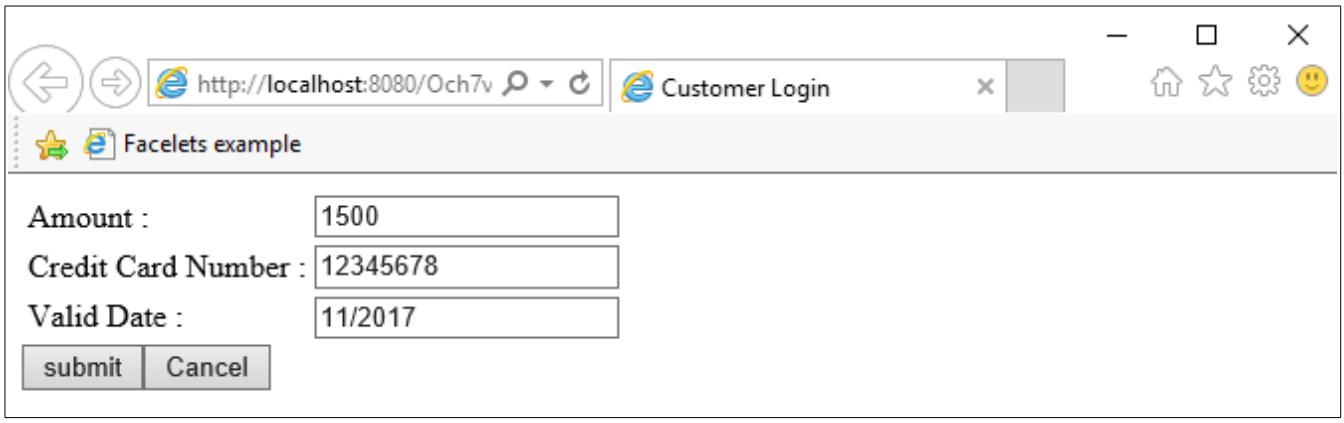
1. When invalid input has been entered

customerLogin.xhtml

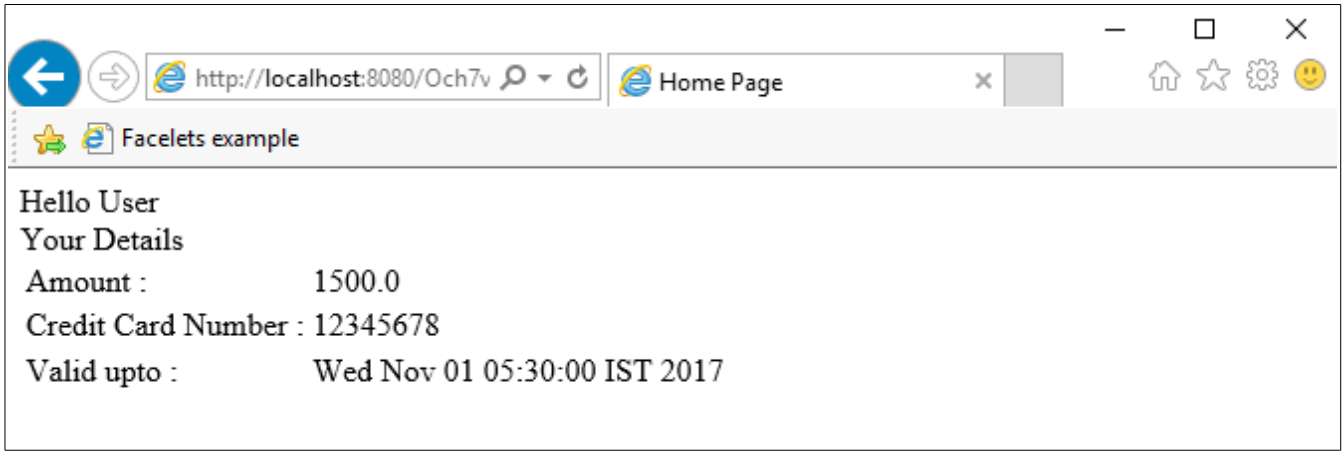


2. When Valid input has been entered

customerLogin.xhtml



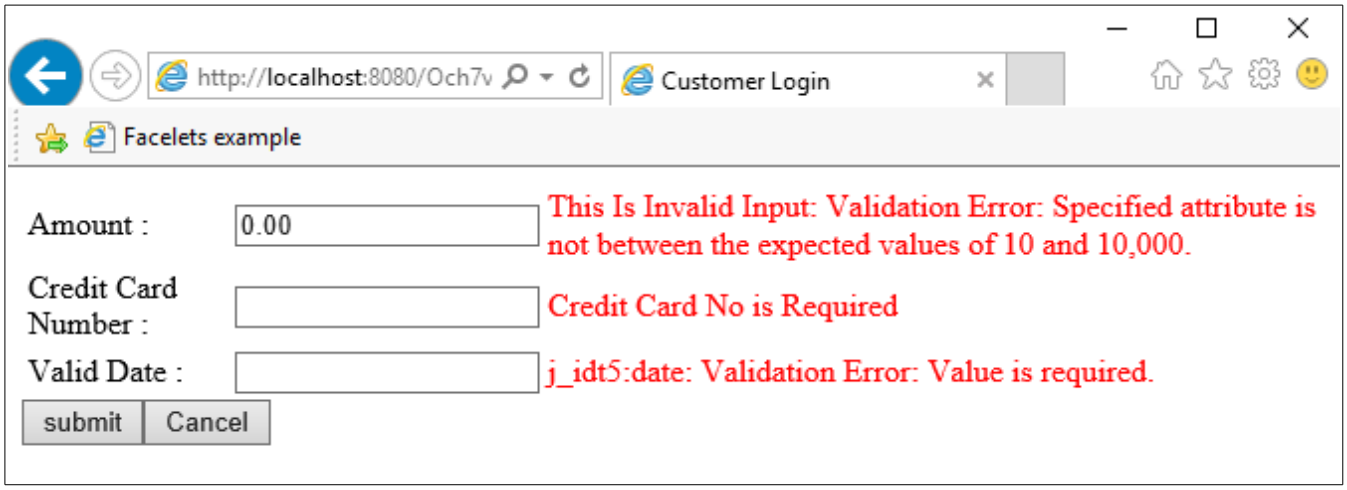
home.xhtml



3. 'Immediate' attribute.

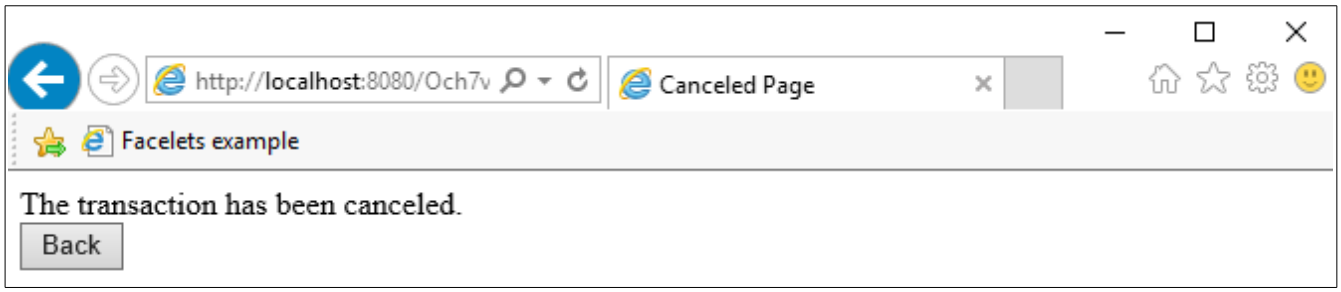
a. When '`immediate`' attribute is not present and cancel button is pressed.

customerLogin.xhtml



b. When '`immediate`' attribute is set and cancel button is pressed.

canceled.xhtml



4. Bean Validation

➔ JSF 2.0 integrates with bean validation framework, a general framework for speficifying validation constraints.

➔ Bean validation framework has tremendous advantage over page-level validation. Suppose a web application updates a bean in several pages. Then one don't need to add validation rules in each page, they will be handled in bean class.

➔ Validations are attached to fields or property getters of a java class.

➔ Ex

```
public class CustomerBean
{
@Size(min=13) private String ccno;
@Future public data getDate(){...}
….
}
```

➔ The list of all the annotations for bean validation is shown below

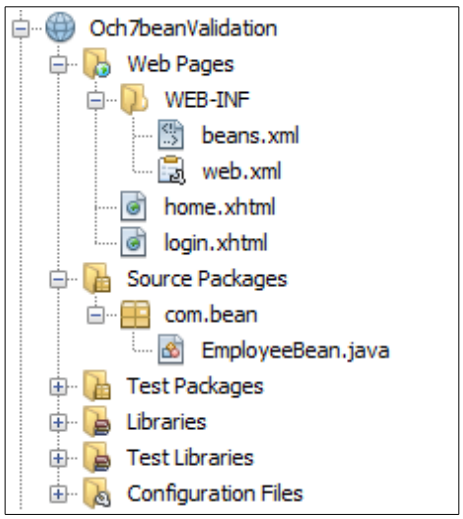
Annotations in the Bean Validation Framework

Annotation	Attribute[a]	Purpose
`@Null`, `@NotNull`	None	Check that a value is null or not null.
`@Min`, `@Max`	The bound as a `long`	Check that a value is at least or at most the given bound. The type must be one of `int`, `long`, `short`, `byte` and their wrappers, `BigInteger`, `BigDecimal`. Note: `double` and `float` are not supported due to roundoff.
`@DecimalMin`, `@DecimalMax`	The bound as a `String`	As above. Can also be applied to a `String`.
`@Digits`	`integer`, `fraction`	Check that a value has, at most, the given number of integer or fractional digits. Applies to `int`, `long`, `short`, `byte` and their wrappers, `BigInteger`, `BigDecimal`, `String`.
`@AssertTrue`, `@AssertFalse`	None	Check that a Boolean value is true or false.
`@Past`, `@Future`	None	Check that a date is in the past or in the future.
`@Size`	`min`, `max`	Check that the size of a string, array, collection, or map is at least or at most the given bound.
`@Pattern`	`regexp`, `flags`	A regular expression and optional compilation flags.

a. All validation annotations have attributes `message` and `groups`. We do not discuss validation groups here.

➔ EX

Lets have directory for project 'Och7beanValidation' as shown below



EmployeeBean.java

```
package com.bean;

import java.io.Serializable;
import javax.inject.Named;
import javax.enterprise.context.Dependent;
import javax.enterprise.context.SessionScoped;
import javax.validation.constraints.Digits;
```

```
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.Size;

@Named(value = "employee")
@SessionScoped
public class EmployeeBean implements Serializable {

    public EmployeeBean() {
    }
    @Size(max=8,message="Name shouldn't exceed 8 characters")
    String name;
    @Digits(integer=4,fraction=0)
    int empid;
    @Min(25) @Max(50)
    int age;

    public String getName() {
        return name;
    }

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

    public int getEmpid() {
        return empid;
    }

    public void setEmpid(int empid) {
        this.empid = empid;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

login.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">
    <h:head>
        <title>Login Page</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel value="Employee Name :"/>
                <h:inputText id="name" value="#{employee.name}" required="true"/>

                <h:outputLabel value="Employee Age :"/>
                <h:inputText id="age" value="#{employee.age}"/>

                <h:outputLabel value="Emp ID :"/>
                <h:inputText id="empid" value="#{employee.empid}"/>
            </h:panelGrid>
            <h:commandButton id="submit" value="submit" action="home"/>

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

home.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">
    <h:head>
        <title>Home Page</title>
    </h:head>
    <h:body>
        <h1>Employee Details</h1>
        <h:panelGrid columns="2">
           <h:outputLabel value="Employee Name : "/>
           <h:outputText value="#{employee.name}"/>
           <h:outputLabel value="Employee Age : "/>
           <h:outputText value="#{employee.age}"/>
           <h:outputLabel value="Employee ID : "/>
           <h:outputText value="#{employee.empid}"/>
        </h:panelGrid>

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

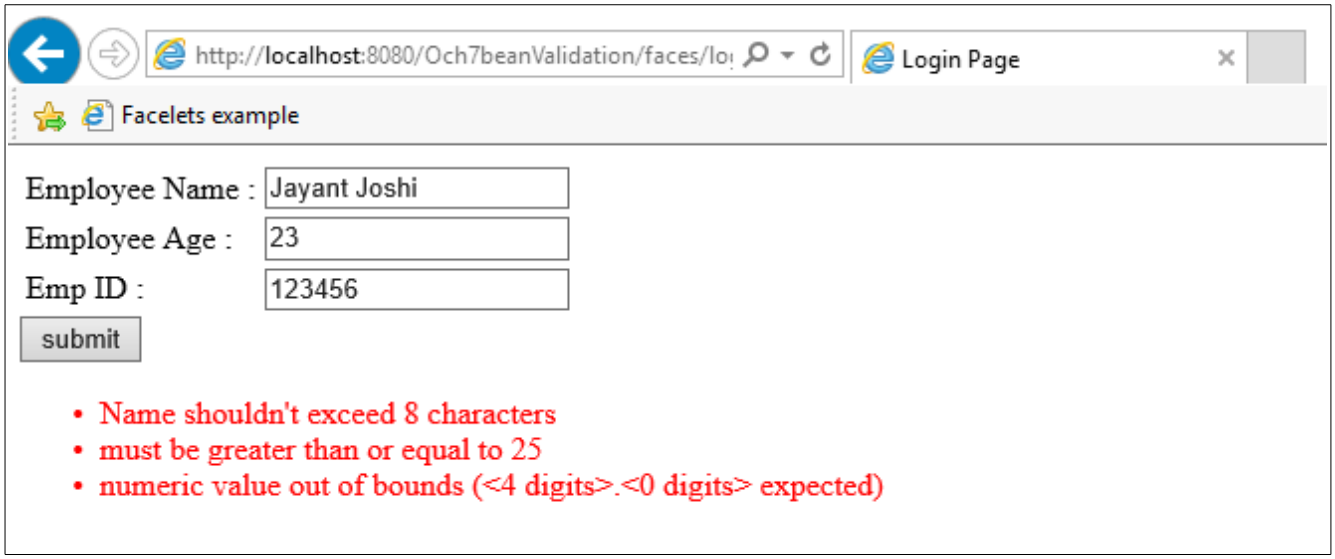
web.xml

```
<?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/login.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Output

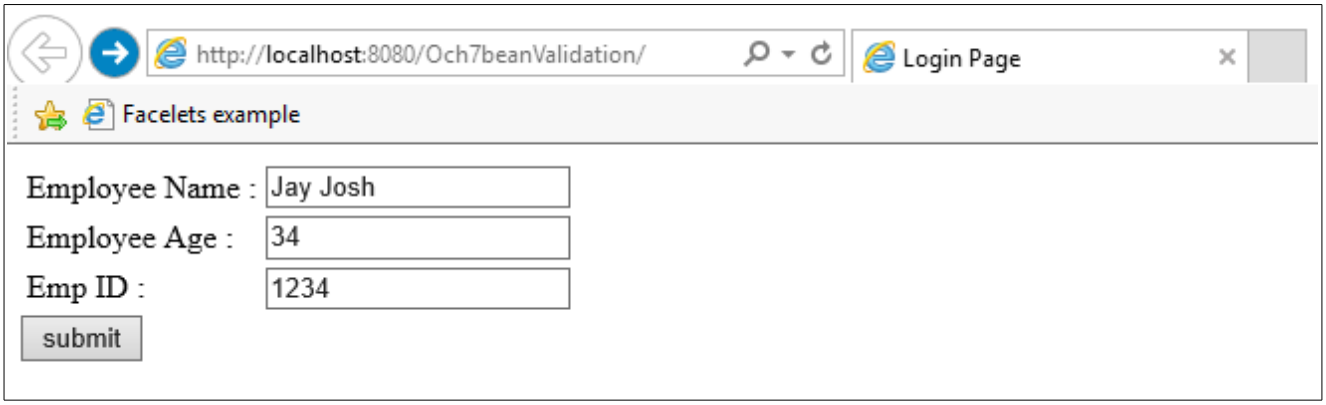
1. Invalid inputs

login.xhtml

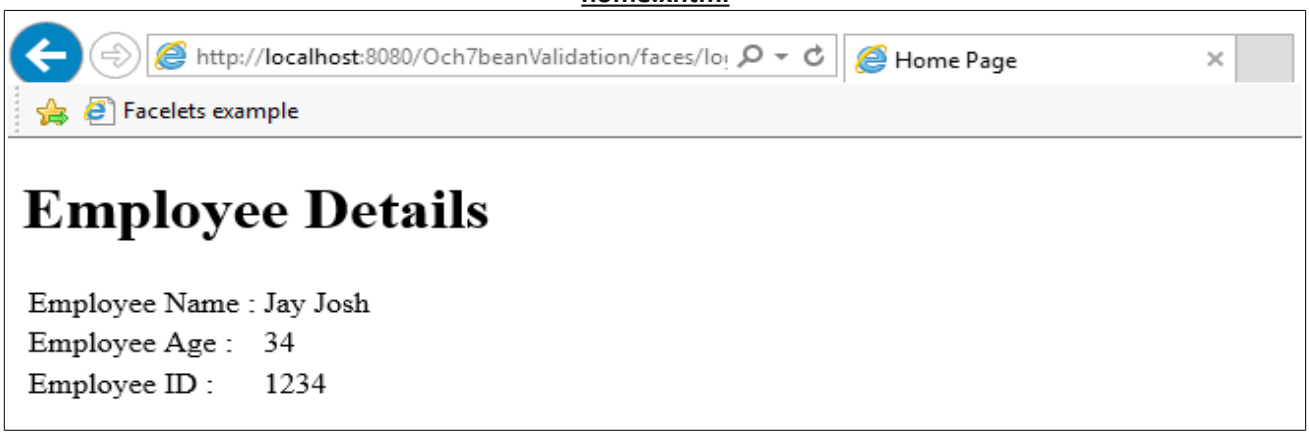


2. Valid Inputs

login.xhtml



home.xhtml



5. Implementing Custom Converter Classes

➔ To implement custom converter one should implement interface '`Converter`'.

➔ `Converter` interface has two methods

```
Object getAsObject(FacesContext context, UIComponent component, String newValue)
String getAsString(FacesContext context, UIComponent component, Object value)
```

➔ Method `getAsObject()` convert String (`newValue`, which comes from `client/browser`) into object of particular type. It throws `ConverterException` if conversion fails. Method `getAsString()` converts `Object(value)` from object to String and throws `ConversionException` if conversion fails.

➔ Converter can be specify using annotation `@FacesConverter` in two ways

i. Using '`converterId`'

In bean class use annotation as shown below

```
@FacesConverter("com.custom.amountConverter")
public class AmountConverter implements Converter {
```

Later it will be used in view as

```
<h:outputLabel value="Amount :"/>
<h:inputText id="amount" value="#{customer.amount}">
   <f:converter converterId="com.custom.amountConverter"/>
 </h:inputText>
 <h:message for="amount"/>
```

2. Using '`forClass`'

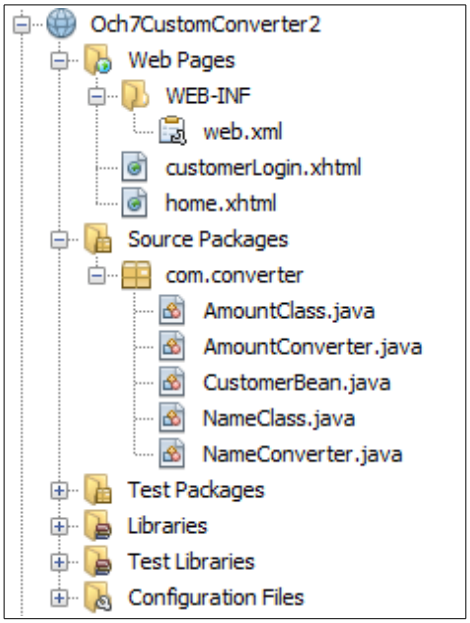
```
@FacesConverter(forClass=NameClass.class)
public class NameConverter implements Converter {
```

With this no need to mention converter any longer. It will be automatically used whenever a value reference has the type '`NameClass`'.

```
 <h:outputLabel value="Name :"/>
   <h:inputText id="name" value="#{customer.nc}"/>
<h:message for="name"/>
```

➔ Example

Lets create a directory 'Och7CustomConverter2'



customerBean.java

```
package com.converter;

import java.io.Serializable;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.Size;
```

```
@Named("customer")
@SessionScoped
public class CustomerBean implements Serializable {

    private NameClass nc = new NameClass("");
    private String amount;
    private String ccno;

    public NameClass getNc() {
        return nc;
    }

    public void setNc(NameClass nc) {
        this.nc = nc;
    }

    public String getAmount() {
        return amount;
    }

    public void setAmount(String amount) {
        this.amount = amount;
    }

    public String getCcno() {
        return ccno;
    }

    public void setCcno(String ccno) {
        this.ccno = ccno;
    }
}
```

NameClass.java

```
package com.converter;

import java.io.Serializable;

public class NameClass implements Serializable {

    String name;

    public NameClass(String name)
    {
        this.name = name;
    }

    @Override
    public String toString()
    {
        return name;
    }
}
```

NameConverter.java

```
package com.converter;

import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.ConverterException;
import javax.faces.convert.FacesConverter;

@FacesConverter(forClass=NameClass.class)
public class NameConverter implements Converter {
```

```
    @Override
    public Object getAsObject(FacesContext context,
            UIComponent component, String value) throws ConverterException{

        String str="";
        char [] valArr = value.toCharArray();
        for(int i=0; i<value.length(); i++)
        {
         str = str+valArr[i]+" ";
        }

        NameClass nc = new NameClass(str);
        return nc;
    }

    @Override
    public String getAsString(FacesContext context,
            UIComponent component, Object value) throws ConverterException {

        String str = value.toString();
        return str;
    }
}
```

AmountClass.java

```
package com.converter;

import java.io.Serializable;
import javax.validation.constraints.Size;

public class AmountClass implements Serializable{


    private String amount;

    public AmountClass(String amount)
    {
        this.amount=amount;
    }

    public String toString()
    {
        return amount;
    }

}
```

AmountConverter.java

```
package com.converter;

import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.ConverterException;
import javax.faces.convert.FacesConverter;

@FacesConverter("com.custom.amountConverter")
public class AmountConverter implements Converter {

    @Override
    public Object getAsObject(FacesContext context,
            UIComponent component, String value) throws ConverterException {

       String str = "";
       str = "â,¹"+value;

       AmountClass ac = new AmountClass(str);
```

```
        return ac;

    }

    @Override
    public String getAsString(FacesContext context,
            UIComponent component, Object value) throws ConverterException {
        String str = value.toString();
        return str;
    }
}
```

Customer.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>Customer Login</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="3">
                <h:outputText value="Name :"/>
                <h:inputText id="name" value="#{customer.nc}"/>
                <h:message for="name" style="color: red"/>

                 <h:outputText value="Amount :"/>
                 <h:inputText id="amount" value="#{customer.amount}">
                    <f:converter converterId="com.custom.amountConverter"/>
                </h:inputText>
                 <h:message for="amount" style="color: red"/>

                 <h:outputText value="Credit Card Number:"/>
                 <h:inputText id="ccno" value="#{customer.ccno}"/>
                 <h:message for="ccno" style="color: red"/>
            </h:panelGrid>

            <h:commandButton id="submit" value="submit" action="home"/>
        </h:form>
    </h:body>
</html>
```

home.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">
    <h:head>
        <title>Home Page</title>
    </h:head>
    <h1>Customer Details</h1>
    <h:body>
        <h:panelGrid columns="2">
          <h:outputText value="Name: "/>
          <h:outputText value="#{customer.nc}"/>
          <h:outputText value="Amount Entered: "/>
          <h:outputText value="#{customer.amount}"/>
          <h:outputText value="Credit Card Number: "/>
          <h:outputText value="#{customer.ccno}"/>
        </h:panelGrid>

    </h:body>
```

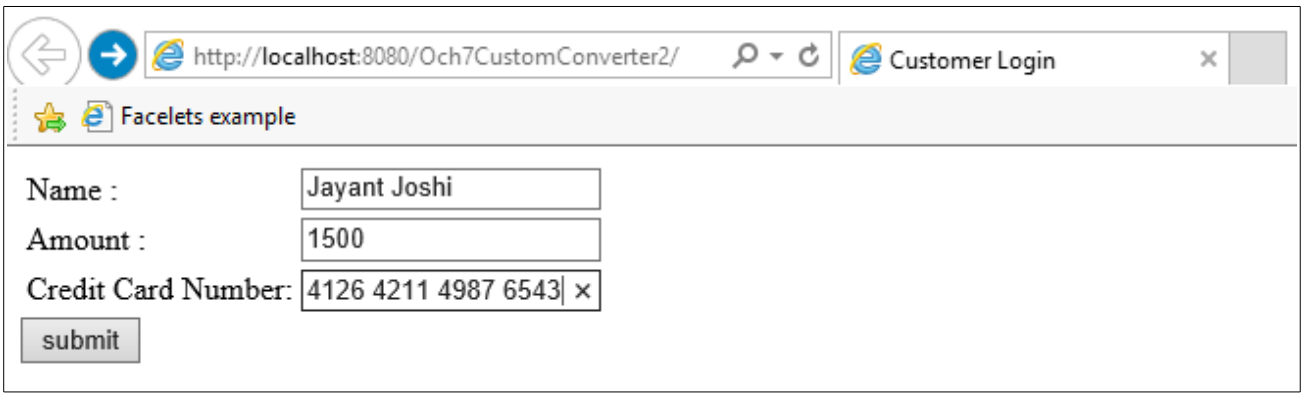
```
</html>
```

web.xml

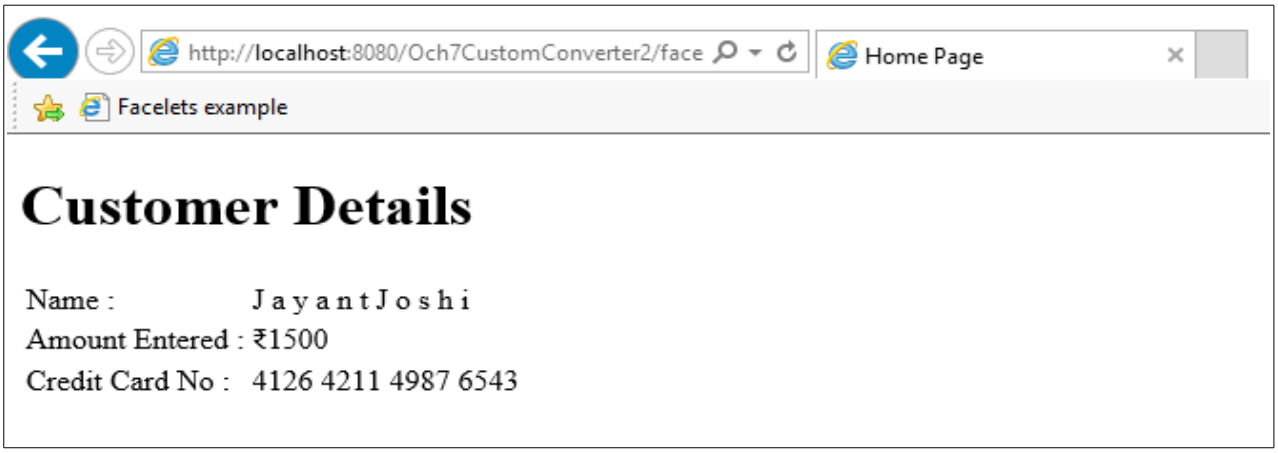
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</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/customerLogin.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Output

customerLogin.xhtml



home.xhtml



5.1. Supplying Attributes to Converter

One can supply attributes to a converter for example in above example if we want credit card number to be separated by '-' then it can be supplied to converter using

```
<h:outputText value="Credit Card Number:"/>
      <h:inputText id="ccno" value="#{customer.ccno}"/>
      <f:attribute name="separator" value="-"/>
<h:message for="ccno" style="color: red"/>
```

Inside converter this attribute will be retrieved using

```
separator = (String) component.getAttributes().get("separator");
```

6. Implementing Custom Validator Classes

➔ Implementing Custom Validator is similar to implementing Custom Converters.

➔ To implement Custom Validator one should implement interface '`Validator`' and it will throw '`ValidatorException`' if validation fails.

➔ Validator interface has only one method as shown below

```
void validate(FacesContext context, UIComponent component, Object value)
```

➔ One should register custom validator similar to custom validator

```
@FacesValidator("com.converter.Name")
public class NameValidator implements Validator {
```

And then used in view page as

```
<h:outputText value="Name :"/>
                <h:inputText id="name" value="#{customer.nc}">
                    <f:validator validatorId="com.converter.Name"/>
                </h:inputText>
<h:message for="name" style="color: red"/>
```

➔ '`ValidatorException`' class has following Constructors.

```
ValidatorException(java.util.Collection<FacesMessage> messages)
ValidatorException(java.util.Collection<FacesMessage> messages, java.lang.Throwable cause)
ValidatorException(FacesMessage message)
ValidatorException(FacesMessage message, java.lang.Throwable cause)
```

➔ Examples for generating 'Validation Errors' messages.

1) By directly providing the error message

```
FacesMessage message = new FacesMessage("Name exceeded 10 characters");
message.setSeverity(FacesMessage.SEVERITY_ERROR);
throw new ValidatorException(message);
```

2) By getting message through message bundle

```
FacesMessage message = com.converter.util.Messages.getMessage("com.converter.messages",
"nameExceeded", null);
message.setSeverity(FacesMessage.SEVERITY_ERROR);
throw new ValidatorException(message);
```

Above `com.converter.util` is a message which contains `Message.java` class which has `getMessage()` method which gets the method from '`messages.properties`' bundle present in `com.converter` package.

➔ Examples

Lets create '`NameValidator`' and '`AmountValidator`' for the project '`Och7CustomConverter2`' as shown below

NameValidator.java

```
package com.converter;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;
import javax.validation.ValidationException;

@FacesValidator("com.converter.nameValidator")
public class NameValidator implements Validator {

    @Override
    public void validate(FacesContext context,
            UIComponent component, Object value)  {

        String str = value.toString();

         if(str.length()>8)
         {
             FacesMessage message = new FacesMessage("Name exceeded 10 characters");
             message.setSeverity(FacesMessage.SEVERITY_ERROR);
             throw new ValidatorException(message);
         }
    }
}
```

AmountValidator.java

```
package com.converter;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

@FacesValidator("com.convert.amountValidator")
public class AmountValidator implements Validator {

    @Override
    public void validate(FacesContext context,
            UIComponent component, Object value) throws ValidatorException {

        String str = value.toString();
        String str1 = "";
        char [] strArr = str.toCharArray();
        for(int i=1; i<strArr.length; i++)
        {
            str1=str1+strArr[i];
            str=str1;
        }
        double amt = Double.parseDouble(str);
        if(amt>2000)
        {
            FacesMessage message = new FacesMessage("Amount Should be < 2000");
            message.setSeverity(FacesMessage.SEVERITY_ERROR);
            throw new ValidatorException(message);
        } } }
```

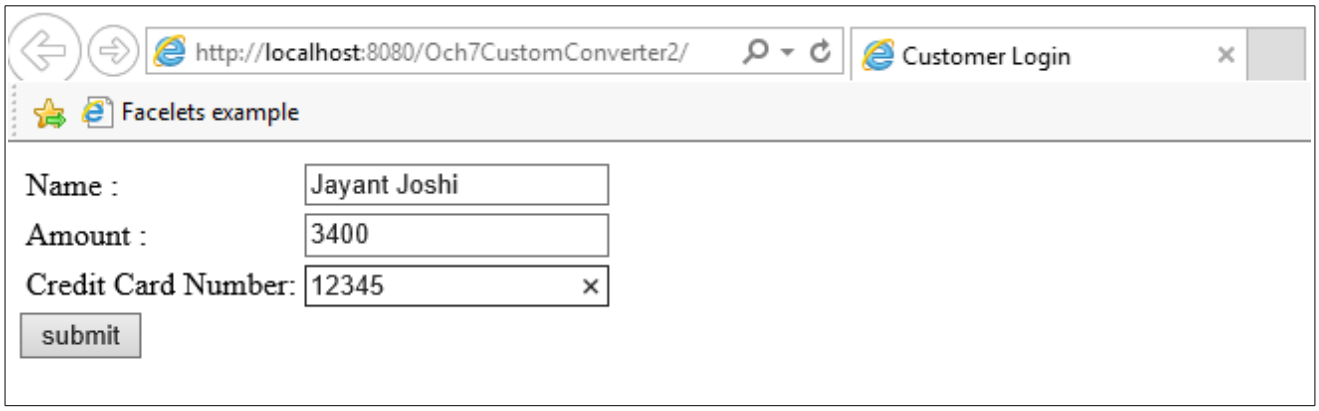
customerLogin.xhtml

```
<?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"
```

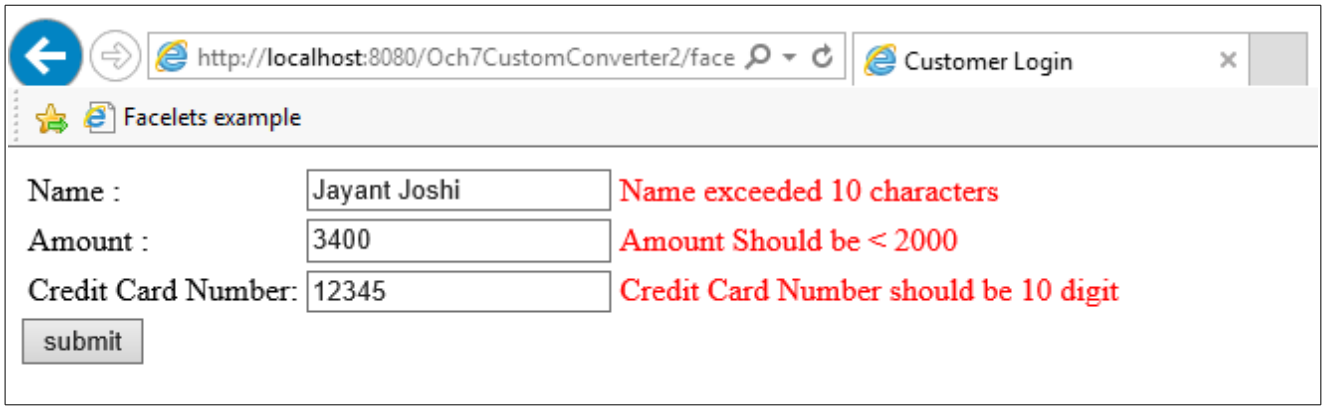
```
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Customer Login</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="3">
                <h:outputText value="Name :"/>
                <h:inputText id="name" value="#{customer.nc}">
                    <f:validator validatorId="com.converter.nameValidator"/>
                </h:inputText>
                <h:message for="name" style="color: red"/>

                 <h:outputText value="Amount :"/>
                 <h:inputText id="amount" value="#{customer.amount}">
                    <f:converter converterId="com.custom.amountConverter"/>
                    <f:validator validatorId="com.convert.amountValidator"/>
                 </h:inputText>
                 <h:message for="amount" style="color: red"/>

                 <h:outputText value="Credit Card Number:"/>
                 <h:inputText id="ccno" value="#{customer.ccno}"/>
                 <h:message for="ccno" style="color: red"/>
            </h:panelGrid>

            <h:commandButton id="submit" value="submit" action="home"/>
        </h:form>
    </h:body>
</html>
```

Output

customerLogin.xhtml



home.xhtml



Note: Annotation

```
@Size(min=10,max=10,message="Credit Card Number should be 10 digit")
private String ccno;
```

has been added to '`CustomerBean.java`' class

➔ One can do validation with bean methods too using method expression. Its advantage is validation method can access instance variable of the class. But the disadantage is such validator cannot be moved to other application for re-use.

6.1. Validating relationships between different components

➔ Validation mechanism could be used only for one component. If there are related components example 'Date' where three different input filed has been given for data, month and year as shown below

➔ In such cases we don't want any field don't send wrong value to the web app example feb 30.

➔ To resolve this we can use validator for last component and could also validate other related component thereusing their component IDs . Example is shown below

```
public class BackingBean {

...
public void validateDate(FacesContext context, UIComponent component,
Object value) {
UIInput dayInput = (UIInput) component.findComponent("day");

UIInput monthInput = (UIInput) component.findComponent("month");
int d = ((Integer) dayInput.getLocalValue()).intValue();
int m = ((Integer) monthInput.getLocalValue()).intValue();
int y = ((Integer) value).intValue();
if (!isValidDate(d, m, y)) {
FacesMessage message = ...;
throw new ValidatorException(message);
}
}
...
}
```

➔ An alternative approach is to attach the validator to a *hidden input field* that comes after all other fields on the form:

```
<h:inputHidden id="datecheck" validator="#{bb.validateDate}" value="needed"/>
```

The hidden field is rendered as a hidden HTML input field. When the field value is posted back, the validator kicks in. (It is essential that you supply some field value. Otherwise, the component value is never updated.) With this approach, the validation function is more symmetrical since all other form components already have their local values set.

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

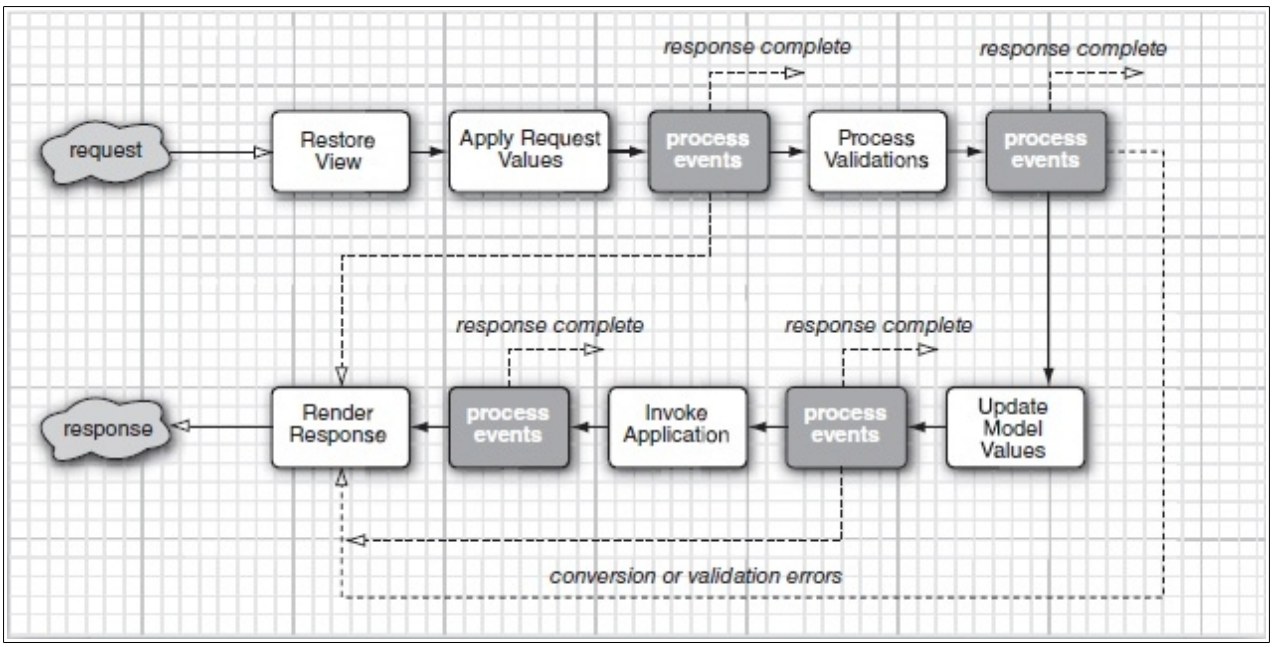
JSF 2.0 added system events. Like its possible to carry out an action before a view or component is rendered.

➔ 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

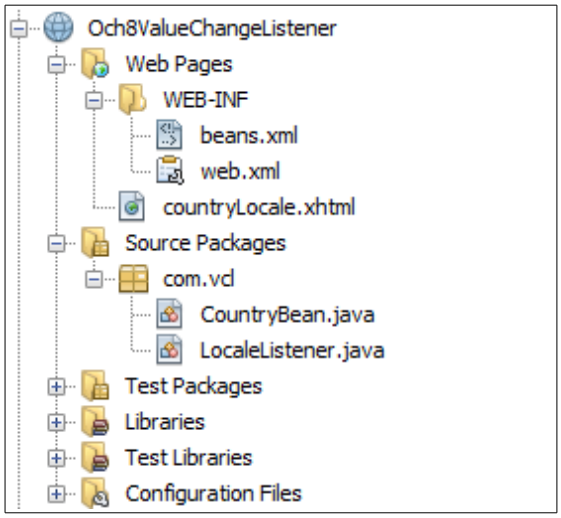
➔ '`ValueChangeEvent`' can be handled in two ways.

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.

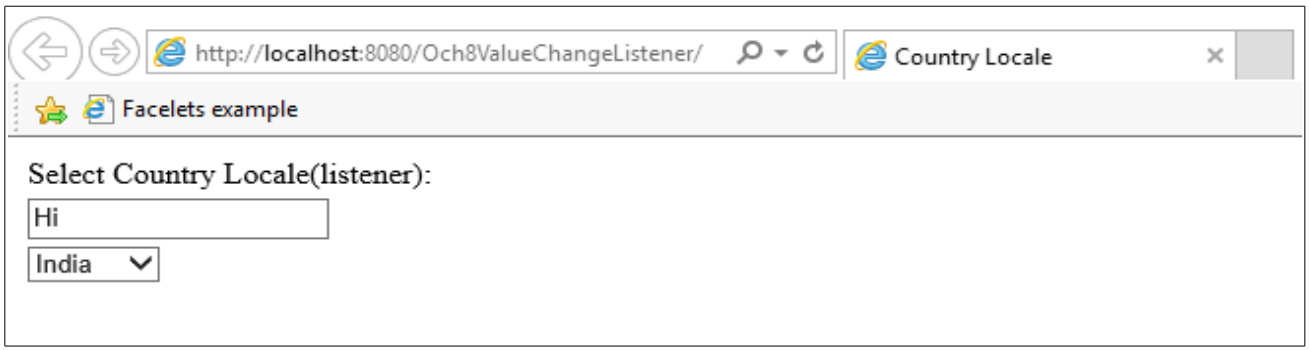
web.xml

```
<?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

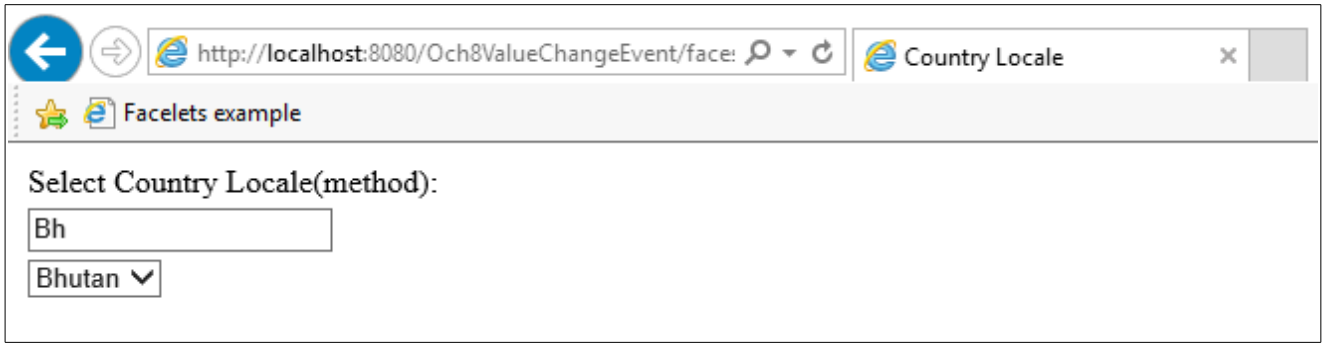


countryLocale.xhtml

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.

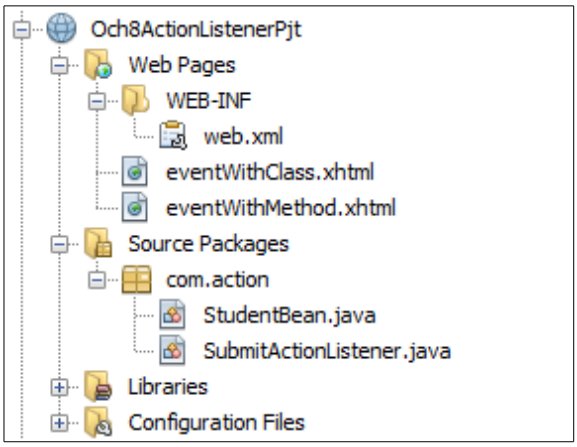
➔ ActionListener too handled in two ways

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.

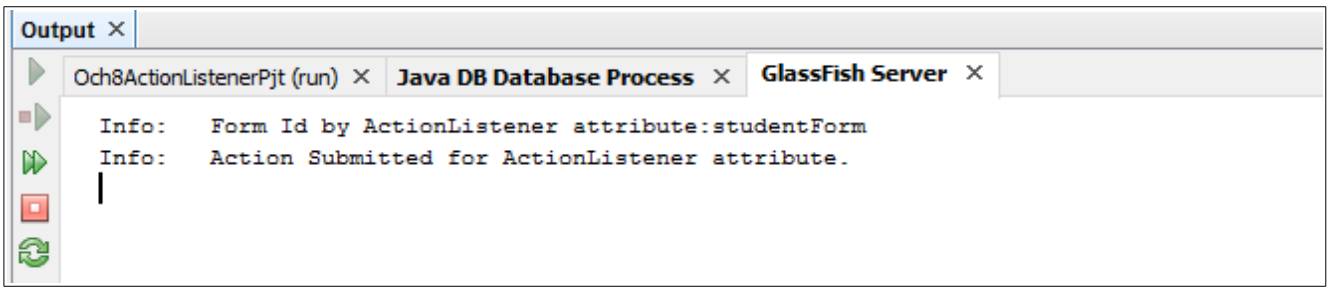
Output

1. Event with method

eventWithMethod.xhtml

eventWithMethod.xhtml(after clicking submit)

Server Console

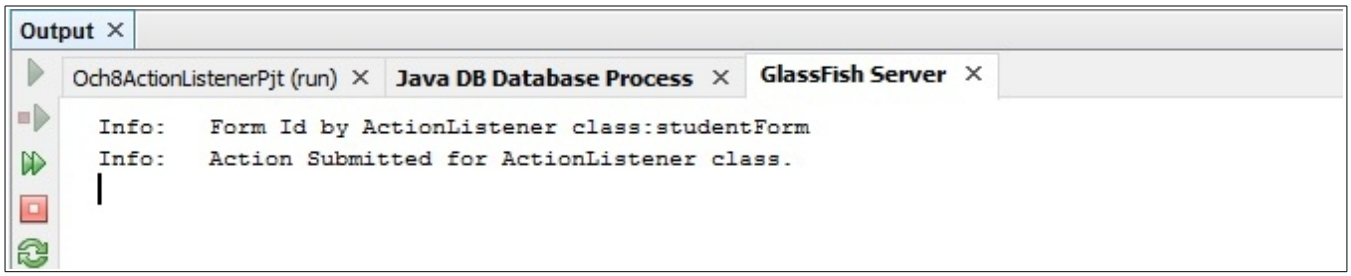


2. Event with Class

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. <u>Immediate Components</u>

➔ 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.

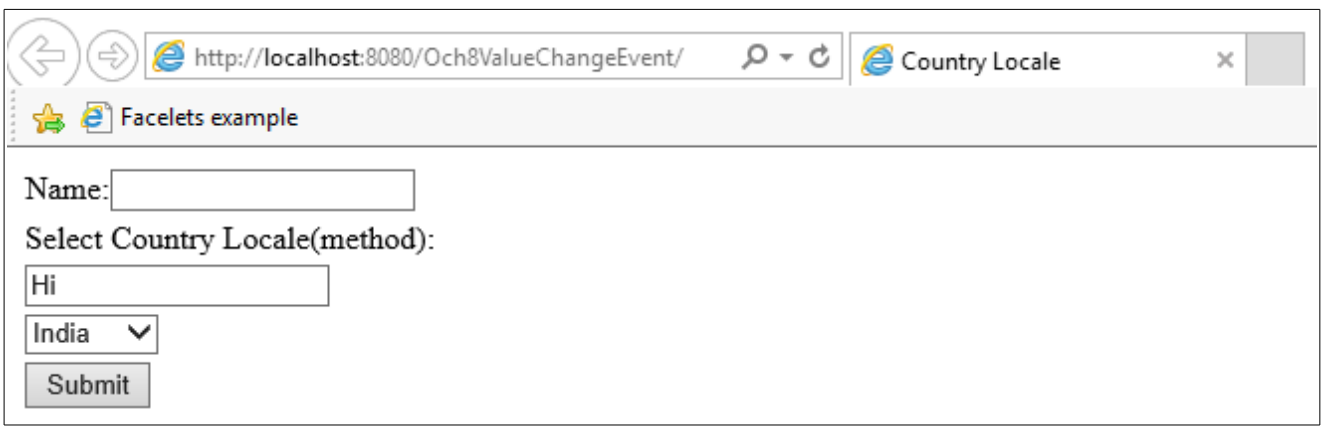
➔ Hence we have Immediate command components which fires events earlier than usual i.e. after 'Apply Request Value' phase. This kicks in the navigation handler which directly goes to 'Render Response' bypassing rest of the life cycle.

➔ 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

<u>countryLocale.xhtml</u>



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>`

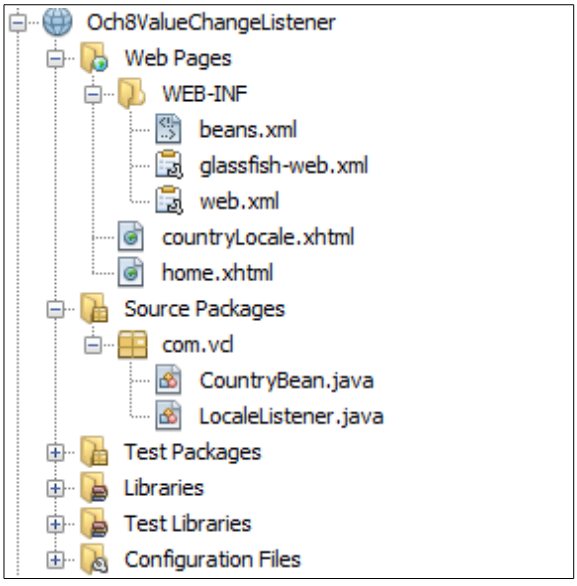
as shown below

```
<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>
```

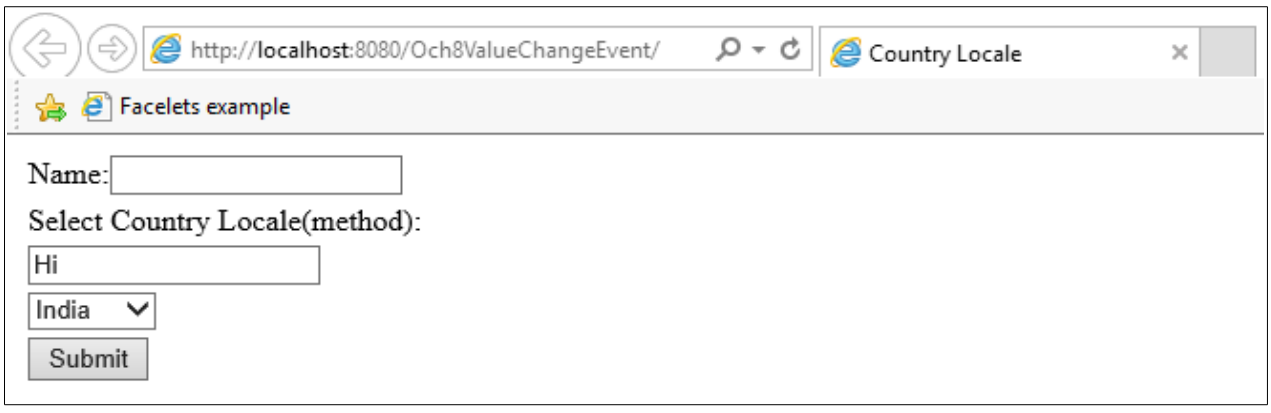
```
        <title>Country Locale</title>
    </h:head>
    <h:body>
        <h:form>

            <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)

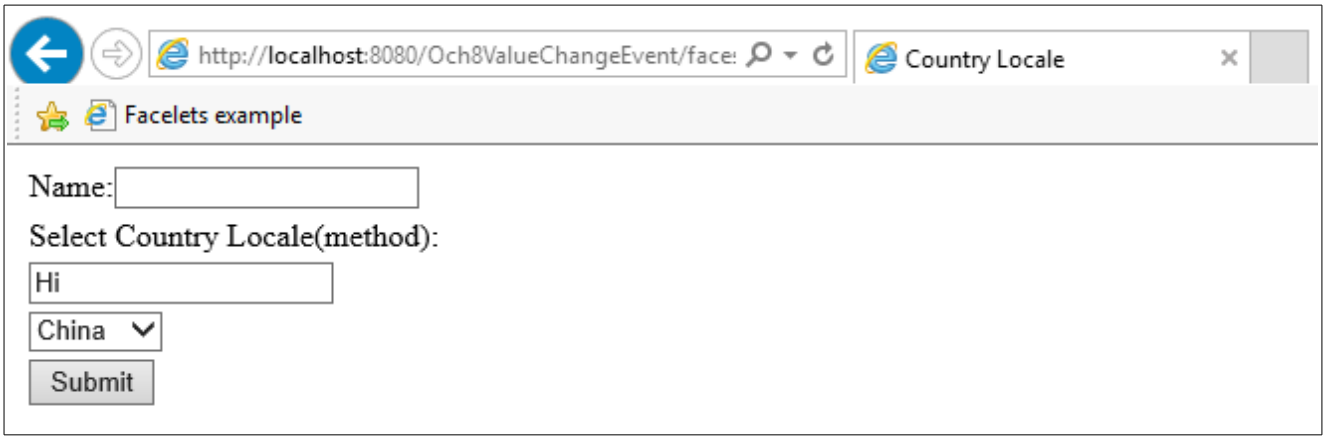
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Home Page</title>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

➔ 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.

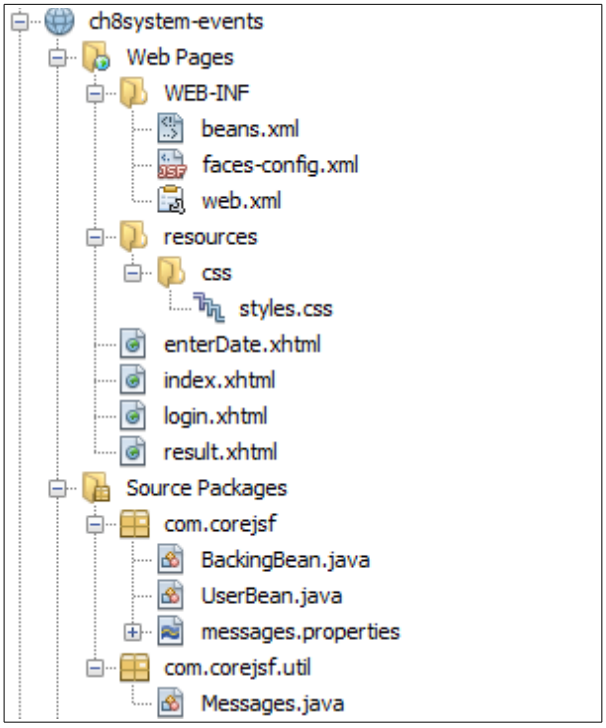
➔ For example, one wants to be sure that user has been logged in before showing a particular page. For that enclose the view inside <f:view> tag and attach a listener.

```
<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>
```

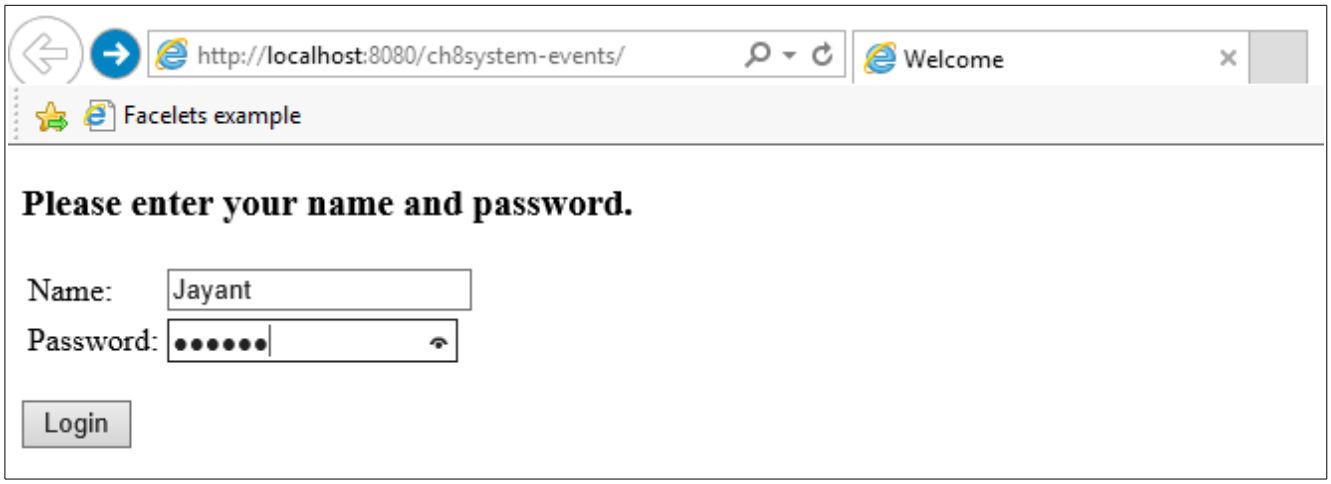
faces-config.xhtml

```
<?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>
```

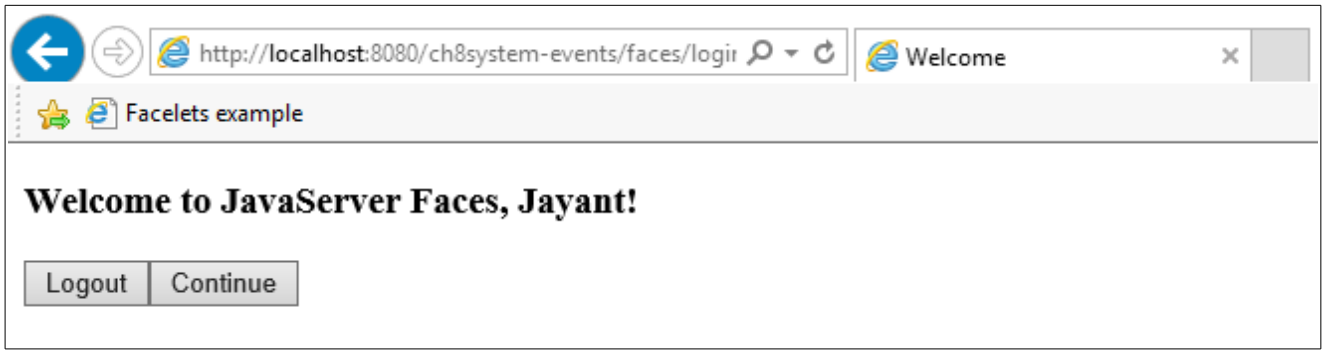
```
      </resource-bundle>
   </application>
</faces-config>
```

Output

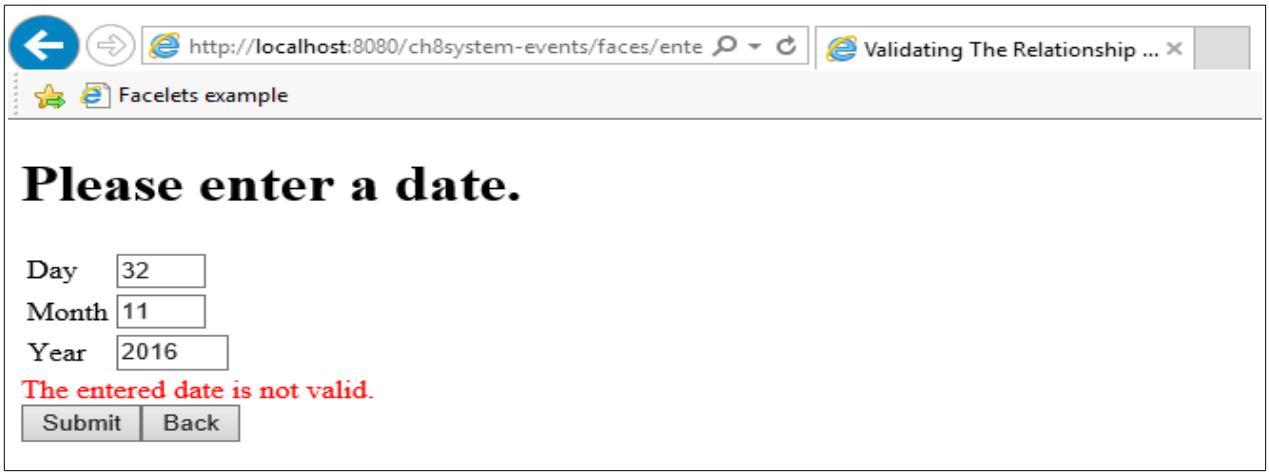
login.xhtml



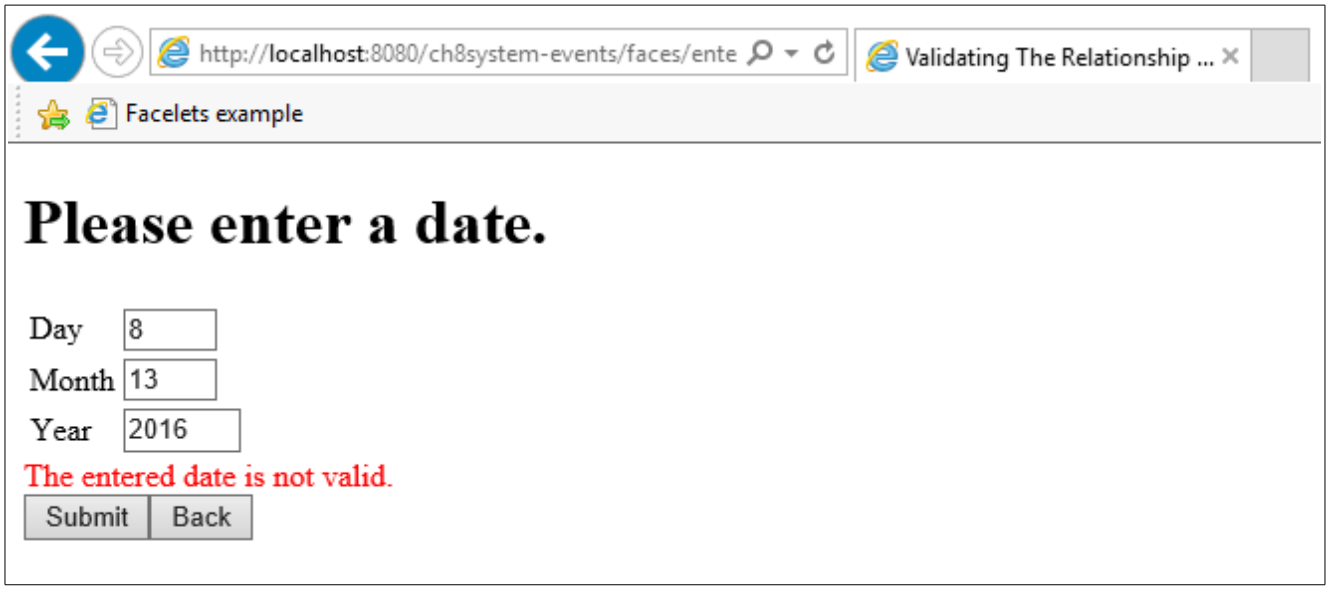
index.xhtml



enterDate..xhtml(day is invalid)

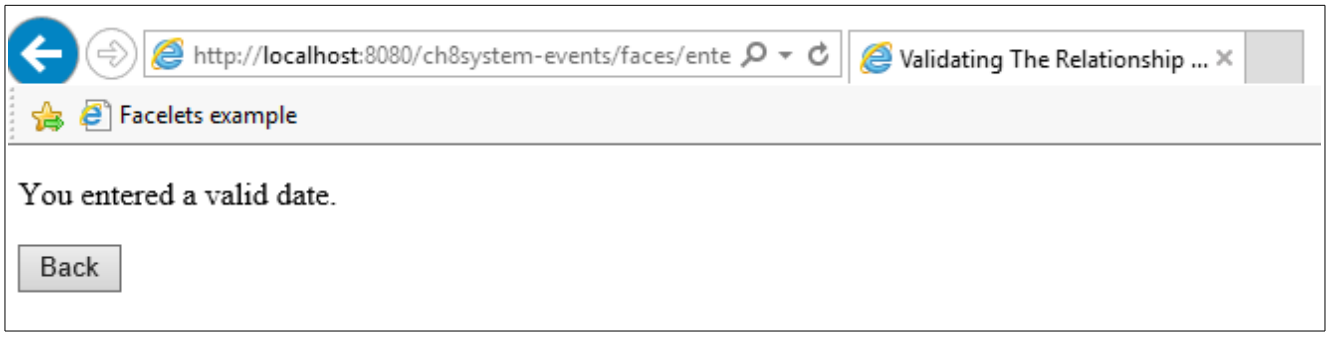


enterDate.xhtml(Month is invalid)



enterDate.xhtml(Valid Date)

result.xhtml



Chapter 9 Composite Component

1. Introduction

➔ Composite components are those components that are implemented using composite library. The name composite components is because new components are composed using existing components.

➔ To use composite tag library add a namespace declaration to an XHTML file as shown below

```
<html xmlns="http://www.w3.org/1999/xhtml"...
xmlns:composite="http://java.sun.com/jsf/composite">
...
</html>
```

Above one can use any prefix for namespace apart from 'composite' but it is widely used prefix.

➔ Once the library is declared one can use any of the following tag belonging to composite library.

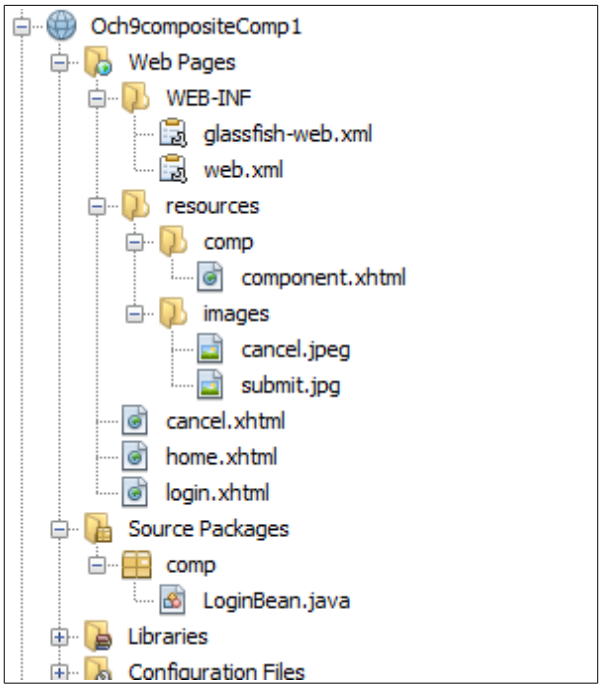
Composite Component Tags

Tag	Description	Used In
interface	Contains other composite tags that expose a composite component's *attributes*, *action sources*, *value holders*, *editable value holders*, and *facets*.	N/A
implementation	Contains the XHTML markup that defines the component. Inside the implementation tag, the component author can access attributes with the expression #{cc.attrs.*attributeName*}.	N/A
attribute	Exposes an attribute of a component to page authors.	Interface
valueHolder	Exposes a component that holds a value to page authors.	Interface
editableValueHolder	Exposes a component that holds an editable value to page authors.	Interface
actionSource	Exposes a component that fires action events, such as a button or a link, to page authors.	Interface
facet	Declares that this component supports a facet with a given name.	Interface
extension	The component author can place this tag inside of any element in an interface. The extension tag can contain arbitrary XML.	Interface subelement
insertChildren	Inserts any child components specified by the page author.	Implementation
renderFacet	Renders a facet that was specified by the page author as a child component.	Implementation
insertFacet	Inserts a facet, specified by the page author, as a facet of the enclosing component.	Implementation

➔ Example

Here `<composite:interface>, <composite:attribute>` and `<composite:implementation>` is used.

Lets create a '`Och7compositeComp1`' directory as shown below



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/login.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

LoginBean.java

```
package comp;

import javax.inject.Named;
import javax.enterprise.context.Dependent;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
```

```
@ManagedBean(name="loginbean")
@SessionScoped
public class LoginBean {

    public LoginBean() {
    }

    String fname, lname;
    int empid;

    public String getFname() {
        return fname;
    }

    public void setFname(String fname) {
        this.fname = fname;
    }

    public String getLname() {
        return lname;
    }

    public void setLname(String lname) {
        this.lname = lname;
    }

    public int getEmpid() {
        return empid;
    }

    public void setEmpid(int empid) {
        this.empid = empid;
    }

    public String action()
    {
        return "cancel";
    }

}
```

component.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:composite="http://xmlns.jcp.org/jsf/composite">
        <h:head>
            <title>This content will not be displayed</title>
        </h:head>
        <h:body>
        <composite:interface>
            <composite:attribute name="fname" required="false"/>
            <composite:attribute name="lname" required="false"/>
            <composite:attribute name="empid" required="false"/>
        </composite:interface>
            <h:form>

                <composite:implementation>

                <h:panelGrid columns="2">

                <h:outputLabel value="First Name : "/>
                <h:inputText id="fname" value="#{cc.attrs.fname}"/>
```

```
                <h:outputLabel value="Last Name : "/>
                <h:inputText id="lname" value="#{cc.attrs.lname}"/>

                <h:outputLabel value="Emp ID : "/>
                <h:inputText id="empid" value="#{cc.attrs.empno}"/>

               </h:panelGrid>

            </composite:implementation>

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

Note: Here we can see attributes have implemented using `cc.attrs`. Here '`cc`' means composite component. Thus, `#{cc.attrs.attributeName}` accesses the *`attributeName`* attribute of the composite component.

login.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:compnt="http://xmlns.jcp.org/jsf/composite/comp"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

    <h:head>
        <title>login page</title>
    </h:head>
    <h:body>
        <h:form>
            <compnt:component
                fname="#{loginbean.fname}"
                lname="#{loginbean.lname}"
                empno="#{loginbean.empid}"
                />

            <h:commandLink action="home">
                <h:graphicImage url="#{'resources/images/submit.jpg'}"/>
            </h:commandLink>

            <h:commandLink action="#{loginbean.action}">
                <h:graphicImage url="#{resource['/images/cancel.jpeg']}"/>
            </h:commandLink>

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

Note : Above we can see the component is accessed through `<compnt:component>` tag. Also note `<h:graphicImage>` tag's '`url`' attribute can be used in two ways.

home.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">
    <h:head>
        <title>Final Page</title>
    </h:head>
    <h:body>
```

```
        Hello #{loginbean.fname} #{loginbean.lname}!!!<br/>
        Emp No : #{loginbean.empid}
    </h:body>
</html>
```

cancel.xhtml

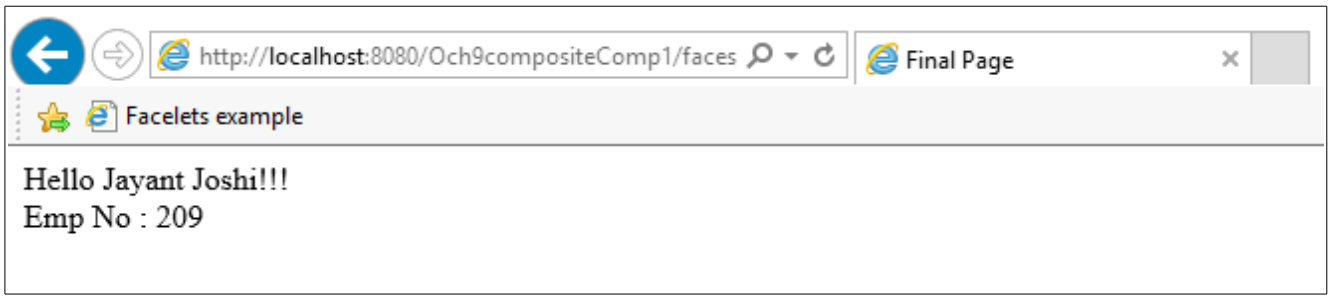
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Cancel Page</title>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

Output

1.

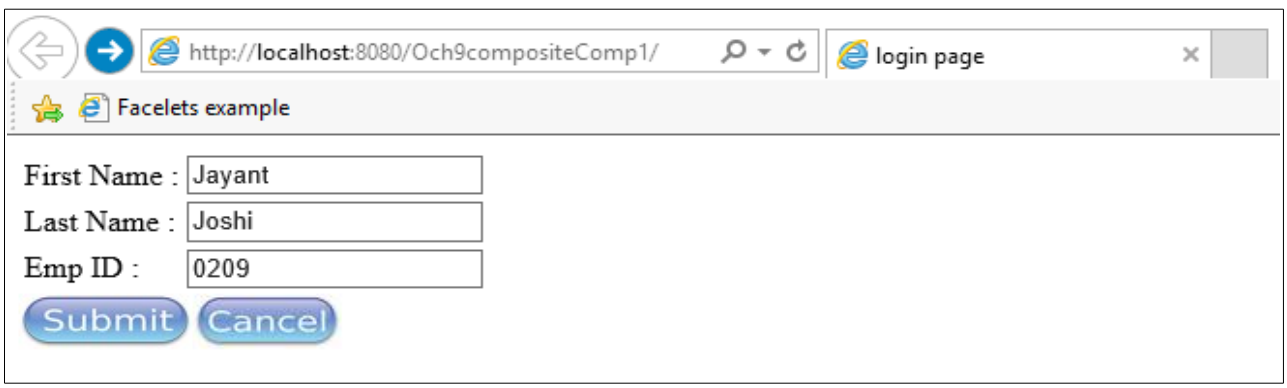
login.xhtml

finalPage.xhtml

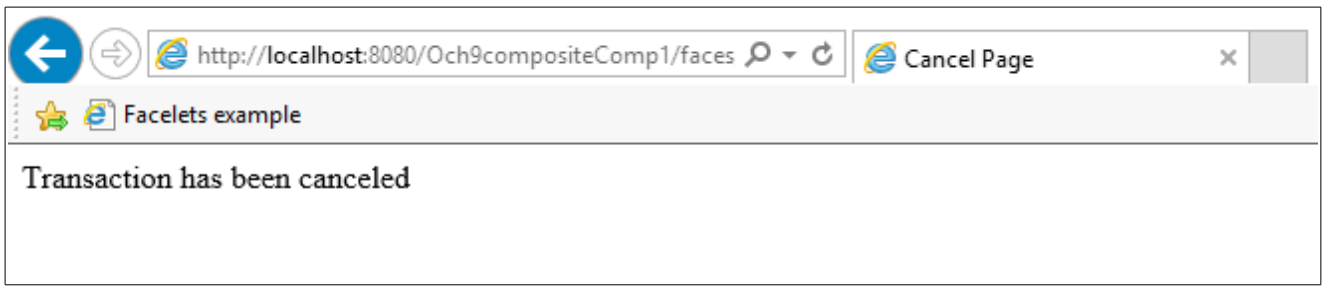


2.

login.xhtml



cancel.xhtml

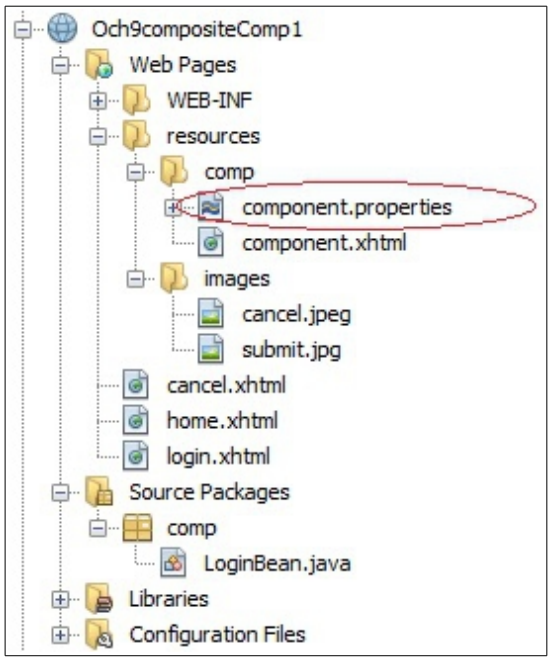


2. Localizing Composite Components

➔ Sometimes one may want to localize the component part for example to sell advertising space in one's component.

➔ This is achieved by creating a resource bundle with the same name as component and saved in the component folder. This bundle will be associated only with the particular component.

➔ This bundle will only contains key/value pairs. In previous example we had component file with name component.xhtml. Hence we should have bundle component.properties as shown below



➔ To access it we should use #{cc.resourceBundle.key}. In out case key is 'footer' hence we have

```
#{cc.resourceBundleMap.footer}
```

➔ Lets see if we insert it in component.xhtml file as shown below

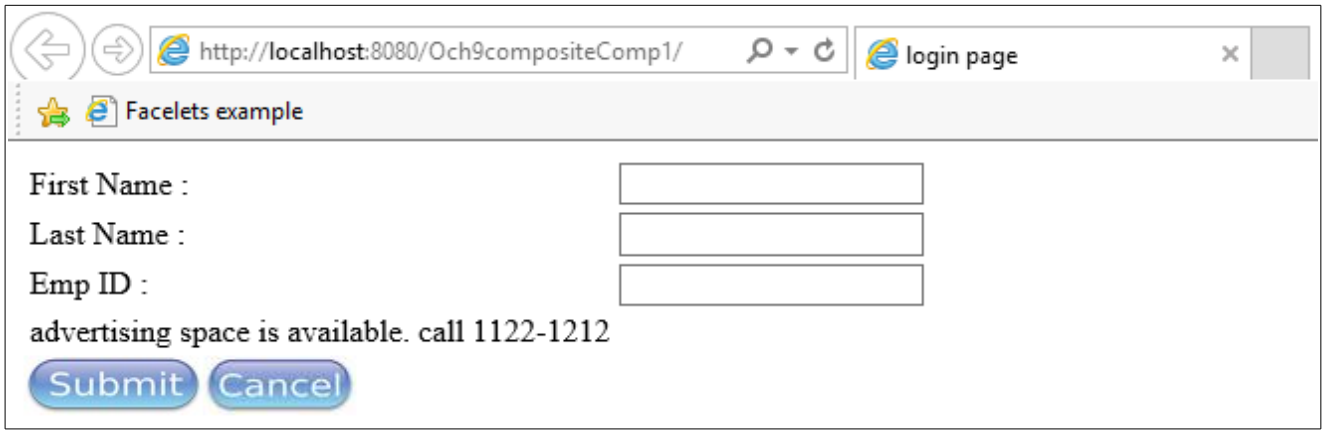
component.xhtml

```
<?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:composite="http://xmlns.jcp.org/jsf/composite">
        <h:head>
            <title>This content will not be displayed</title>
        </h:head>
        <h:body>
        <composite:interface>
            <composite:attribute name="fname" required="false"/>
            <composite:attribute name="lname" required="false"/>
```

```
            <composite:attribute name="empid" required="false"/>
        </composite:interface>
            <h:form>

                <composite:implementation>

                <h:panelGrid columns="2">

                <h:outputLabel value="First Name : "/>
                <h:inputText id="fname" value="#{cc.attrs.fname}"/>

                <h:outputLabel value="Last Name : "/>
                <h:inputText id="lname" value="#{cc.attrs.lname}"/>

                <h:outputLabel value="Emp ID : "/>
                <h:inputText id="empid" value="#{cc.attrs.empid}"/>

                <h:outputText value=#{cc.resourceBundleMap.footer}"/>

               </h:panelGrid>

            </composite:implementation>

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

Ouput

login.xhtml



3. Adding validators to components

➔ In the previous example we have supplied inputs for composite component as

```
<compnt:component
                fname="#{loginbean.fname}"
                lname="#{loginbean.lname}"
                empno="#{loginbean.empid}"
                />
```

with this there will be one disadvantage, we cannot supply validators to the components here.

➔ To add validators we should use `<composite:editableValueHolder>` tag as shown below

```
<composite:interface>
. . . . .
<composite:editableValueHolder name="fname" />
<composite:editableValueHolder name="lname" />
<composite:editableValueHolder name="empid" />

</composite:interface>
```

No need to add anything to the <compose:implementation> part.

Later the view where component is used validators are added as

```
<compnt:component
                fname="#{loginbean.fname}"
                lname="#{loginbean.lname}"
                empno="#{loginbean.empid}">


<f:validateLength minimum="4" minimum="10" for="fname"/>
<f:validateLength minimum="4" minimum="10" for="lname"/>
<f:validator validatorId="com.validator.empIdValidator" for="empid"/>
</compnt:component>
```

➔ Above we can see validators for 'fname' and 'lname' specify minimum length of 4 and maximum length of 10.

Validator for 'empid' is a custom validator with validatorId 'com.validator.empIdValidator'.

➔ Lets check the project 'Och9CompositeComp1' after adding validators.

component.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:composite="http://xmlns.jcp.org/jsf/composite">
        <h:head>
            <title>This content will not be displayed</title>
        </h:head>
        <h:body>
        <composite:interface>
            <composite:attribute name="fname" required="false"/>
            <composite:attribute name="lname" required="false"/>
            <composite:attribute name="empid" required="false"/>

            <composite:editableValueHolder name="fname" />
            <composite:editableValueHolder name="lname" />
            <composite:editableValueHolder name="empid" />
        </composite:interface>
            <h:form>

                <composite:implementation>

                <h:panelGrid columns="2">

                <h:outputLabel value="First Name : "/>
                <h:inputText id="fname" value="#{cc.attrs.fname}"/>

                <h:outputLabel value="Last Name : "/>
                <h:inputText id="lname" value="#{cc.attrs.lname}"/>

                <h:outputLabel value="Emp ID : "/>
                <h:inputText id="empid" value="#{cc.attrs.empid}"/>

                <h:outputText value=#{cc.resourceBundleMap.footer}"/>

               </h:panelGrid>

            </composite:implementation>

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

Note: Above value used in 'name=fname' attribute of <composite:editableValueHolder> must match 'id=fname' of related tag inside <composite:implementation>.

Example `<h:inputText id=”fname” value=#{cc.attrs.fname}/>`.

login.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:compnt="http://xmlns.jcp.org/jsf/composite/comp"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

    <h:head>
        <title>login page</title>
    </h:head>
    <h:body>
        <h:form>
            <compnt:component
                fname="#{loginbean.fname}"
                lname="#{loginbean.lname}"
                empno="#{loginbean.empid}">

                <f:validateLength minimum=”4” minimum=”10” for="fname"/>
                <f:validateLength minimum=”4” minimum=”10” for="lname"/>
                <f:validator validatorId="com.validator.empIdValidator"
for="empid"/>
            </compnt:component>

            <h:commandLink action="home">
                <h:graphicImage url="#{'resources/images/submit.jpg'}"/>
            </h:commandLink>

            <h:commandLink action="#{loginbean.action}">
                <h:graphicImage url="#{resource['/images/cancel.jpeg']}"/>
            </h:commandLink>

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

empIdValidator.java

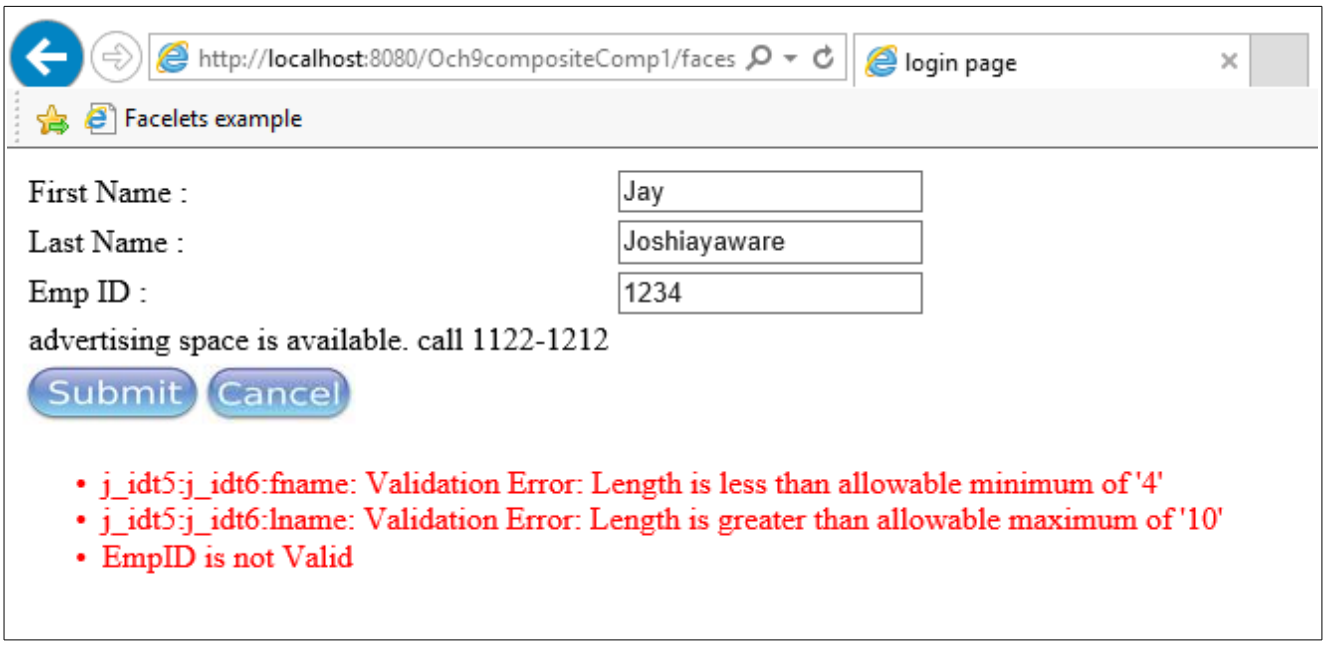
```
package comp;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

@FacesValidator("com.validator.empIdValidator")
public class empIdValidator implements Validator {

    @Override
    public void validate(FacesContext context,
           UIComponent component, Object value) throws ValidatorException {

        String [] empIds={"1111","2222","3333","4444"};
        String empId = value.toString();
        boolean match=false;
        for(int i=0; i<empIds.length; i++)
        {
            if(((empId.equals(empIds[i]))))
            {
                match=true;
            }
```

```
        }
        if(!match)
        {
            FacesMessage message = new FacesMessage("EmpID is not Valid");
            message.setSeverity(FacesMessage.SEVERITY_ERROR);
            throw new ValidatorException(message);
        }
    }
}
```

Output

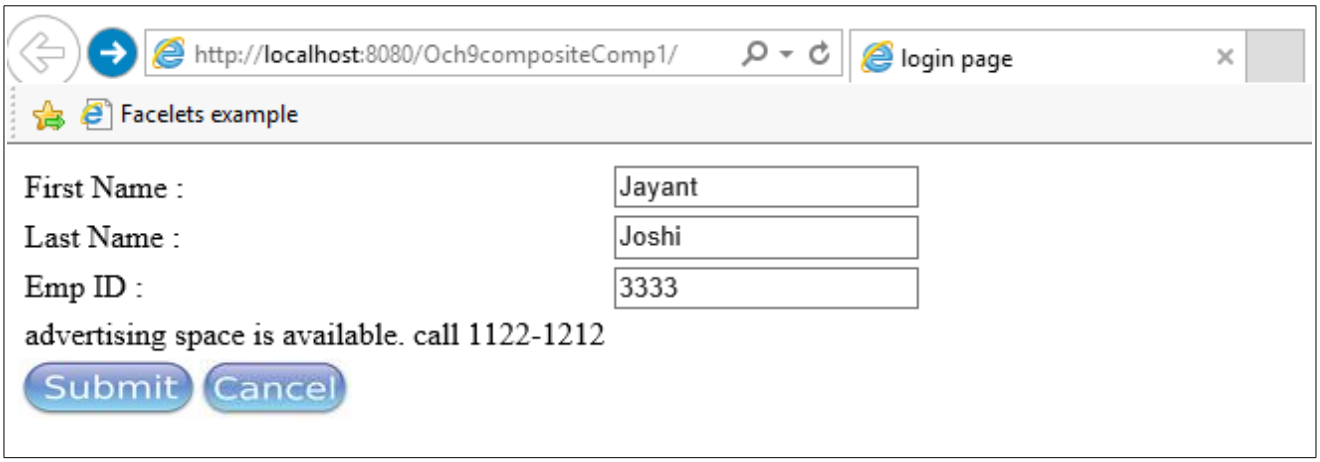
1. Invalid Inputs

login.xhtml

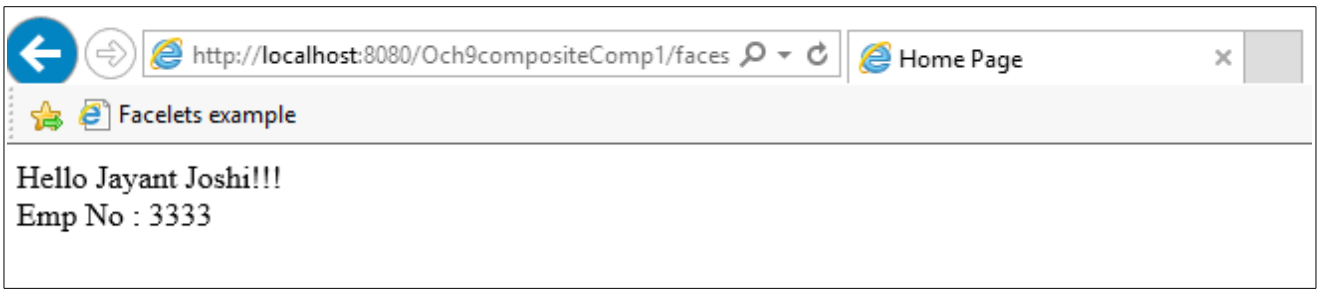


2. Valid Inputs

login.xhtml



home.xhtml



➔ Another way to add validator is shown below with changes in red.

Component.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:composite="http://xmlns.jcp.org/jsf/composite">
        <h:head>
            <title>This content will not be displayed</title>
        </h:head>
        <h:body>
        <composite:interface>
            <composite:attribute name="fname" required="false"/>
            <composite:attribute name="lname" required="false"/>
            <composite:attribute name="empid" required="false"/>

           <composite:editableValueHolder name="form:fname" />
           <composite:editableValueHolder name="form:lname" />
           <composite:editableValueHolder name="form:empid" />
        </composite:interface>
            <h:form id=form>

                <composite:implementation>

                <h:panelGrid columns="2">

                <h:outputLabel value="First Name : "/>
                <h:inputText id="fname" value="#{cc.attrs.fname}"/>

                <h:outputLabel value="Last Name : "/>
                <h:inputText id="lname" value="#{cc.attrs.lname}"/>

                <h:outputLabel value="Emp ID : "/>
                <h:inputText id="empid" value="#{cc.attrs.empid}"/>

                <h:outputText value=#{cc.resourceBundleMap.footer}"/>

               </h:panelGrid>

            </composite:implementation>

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

Note: Above we can see `<h:form id="form">` has been given an id as '`form`' and it becomes prefix for '`fname`' and '`lname`' since both of these inputs reside in '`form`'. Rest `login.xhtml` page remains the same.

4. Exposing Action Sources

➔ We know three tags one can use inside `<composite:interface>` tags. They are

1. `<composite:actionSource>`
2. `<compsite:valueHolder>`
3. `<compsite:editableValueHolder>`

We have already used `<composite:editableValueSource>`. Here we are going to discuss about `<composite:actionSource>` and `<compsite:valueHolder>`.

4.1. <composite:actionSource>

➔ If one want to use action Listener then he should use `<compsite:actionSource>` in component file. It will be use same way `<composite:editableValueHolder>` have been used in previous example.

➔ Example

```
<composite:interface>
   . . . . . .
    <composite:attribute name="submitBtnAction" method-signature="java.lang.String
                              action()"/>
    <composite:actionSource name="submitButton" targets="form:submitButton"/>
   . . . . . .
</composite:interface>
```

and inside `<composite:implementation>` we have

```
<composite:implementation>
    <h:form id="form">
       . . . . . .
       <h:commandButton id="submitButton" value="Submit"
                        action="#{cc.attrs.submitBtnAction}"/>
       </h:panelGrid>
    </h:form>
</composite:implementation>
```

Note: Above we can see that inside `<composite:interface>` we have `<composite:actionSource>` tag with attribute `target="form:submitButton"`. This is because it reside in `<h:form id="form">` tag. Also note that `name="submitButton"` attribute inside `<composite:actionSource>` tag must match with `id="submitButton"` attribute of `<h:commonButton>` tag inside `<composite:implementation>`.

4.2. <composite:valueHolder>

➔ The tag `<composite:editableValueHolder>` could be used for value Change Listeners, converter and validators(In previous example we have used it only for validators). But if one want to use only validators then we can use `<composite:valueHolder>`.

➔ Example

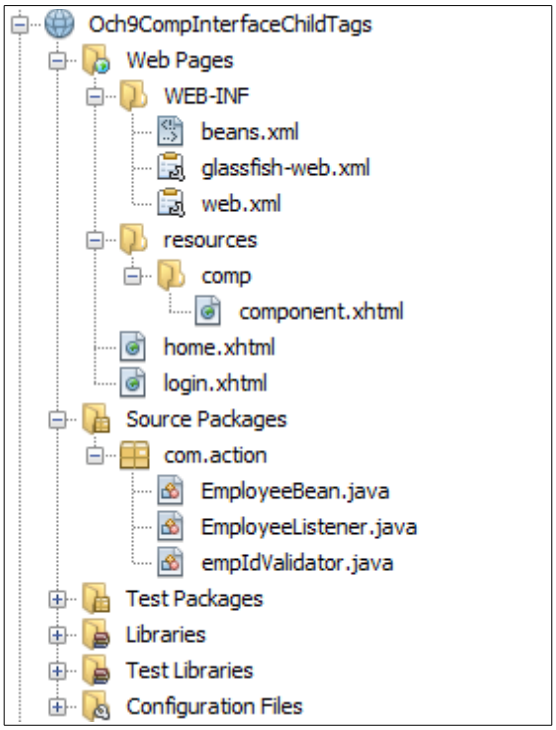
```
<composite:interface>
    . . . . . .
    <composite:attribute name="empIdLable"/>
    <composite:attribute name="empId"/>
    <composite:valueHolder name="empId" targets="form:empId"/>
    . . . . . .
</composite:interface>
```

and inside `<composite:implementation>`

```
<composite:implementation>
    <h:form id="form">
       <h:panelGrid columns="2">
        . . . . .
            <h:inputText id="empId" value="#{cc.attrs.empId}"/>
        . . . . .
       </h:panelGrid>
    </h:form>
</composite:implementation>
```

Note: Here too attribute `name="empId"` inside `<compsite:valueHolder>` must match attribute `id="empId"` of `<h:inputText>` tag. Also attribute `target="form:empId"` has been used because it reside inside `<form id="form">` tag.

➔ Lets create directory for project 'Och9CompInterfaceChildTags' as shown below



EmployeeBean.java

```
package com.action;


import javax.inject.Named;
import javax.enterprise.context.SessionScoped;
import java.io.Serializable;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;

@Named(value = "employee")
@SessionScoped
public class EmployeeBean implements Serializable {

    public EmployeeBean() {
    }

    String name, empId;

    public String getName() {
        return name;
    }

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

    public String getEmpId() {
        return empId;
    }

    public void setEmpId(String empId) {
        this.empId = empId;
    }

   public String checkEmpId()
    {
        System.out.println("Inside bean method");
        return "home";
    }
}
```

EmployeeBean.java

```
package com.action;


import javax.inject.Named;
import javax.enterprise.context.SessionScoped;
import java.io.Serializable;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;

@Named(value = "employee")
@SessionScoped
public class EmployeeBean implements Serializable {

    public EmployeeBean() {
    }

    String name, empId;

    public String getName() {
        return name;
    }

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

    public String getEmpId() {
        return empId;
    }

    public void setEmpId(String empId) {
        this.empId = empId;
    }

   public String checkEmpId()
    {
        System.out.println("Inside bean method");
        return "home";
    }

}
```

EmployeeListener.java

```
package com.action;

import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;
import javax.faces.event.ActionListener;
public class EmployeeListener implements ActionListener {
      @Override
      public void processAction(ActionEvent event)
                  throws AbortProcessingException {
               System.out.println("Inside listener");



           System.out.println("Form Id by ActionListener
class:"+event.getComponent().getParent().getId());
      }
}
```

component.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:composite="http://java.sun.com/jsf/composite">

<composite:interface>
    <composite:attribute name="nameLable"/>
    <composite:attribute name="name"/>
    <composite:attribute name="empIdLable"/>
    <composite:attribute name="empId"/>
    <composite:valueHolder name="empId" targets="form:empId"/>
    <composite:attribute name="submitBtnAction" method-signature="java.lang.String
action()"/>
    <composite:actionSource name="submitButton" targets="form:submitButton"/>
</composite:interface>

 <h:body>
<composite:implementation>
    <h:form id="form">
       <h:panelGrid columns="2">
       <h:outputLabel value="#{cc.attrs.nameLable}"/>
       <h:inputText id="name" value="#{cc.attrs.name}"/>
       <h:outputLabel value="#{cc.attrs.empIdLable}"/>
       <h:inputText id="empId" value="#{cc.attrs.empId}"/>
       <h:commandButton id="submitButton" value="Submit"
                        action="#{cc.attrs.submitBtnAction}"/>
       </h:panelGrid>
    </h:form>
</composite:implementation>

 </h:body>

</html>
```

login.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:comp="http://java.sun.com/jsf/composite/comp"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
        <title>Login Page</title>
    </h:head>
    <h:body>
        <h1>Login Page</h1>
        <h:form>
          <comp:component
              nameLable="Name :"
              name="#{employee.name}"
              empIdLable="EmpID :"

              empId="#{employee.empId}"
              submitBtnAction="#{employee.checkEmpId}">
              <f:validator for="empId" validatorId="com.validator.empIdValidator"/>
              <f:actionListener for="submitButton" type="com.action.EmployeeListener"/>
          </comp:component>
        </h:form>
    </h:body>
</html>
```

home.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">
    <h:head>
        <title>Home Page</title>
    </h:head>
    <h:body>
        Hello #{employee.name}
    </h:body>
</html>
```

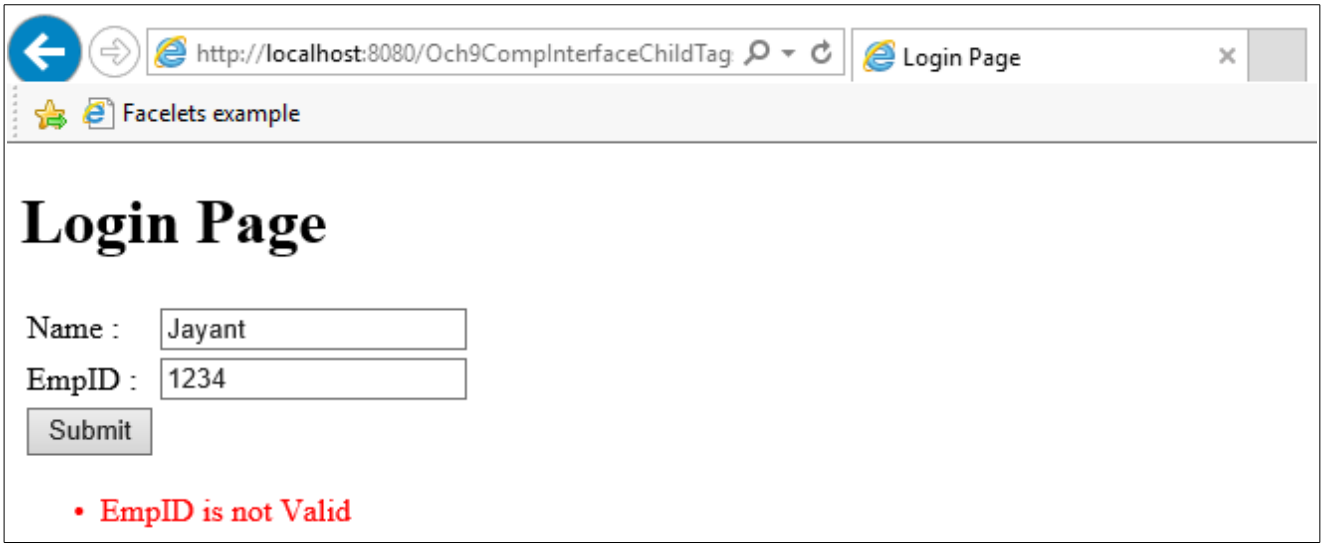
web.xml

```
<?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/login.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

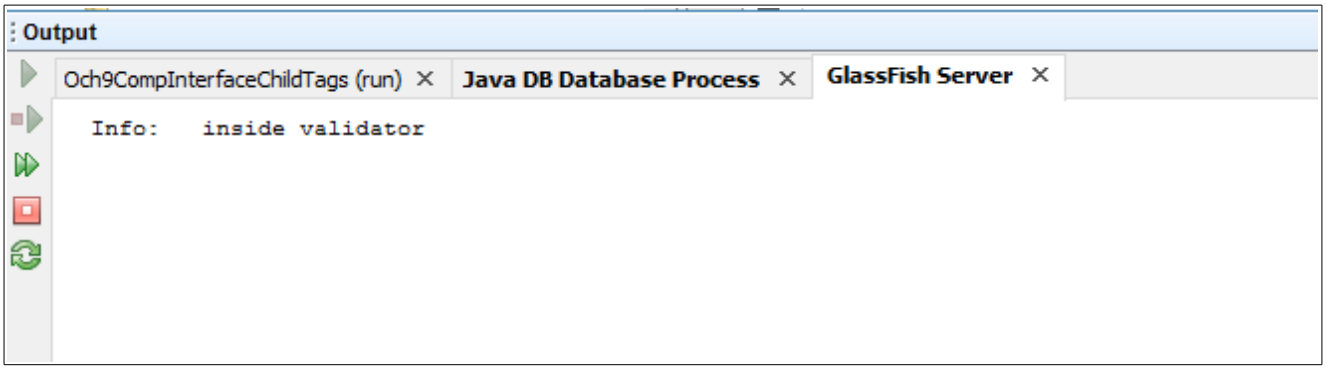
Output

1. When 'employee ID' is not validate

login.xhtml



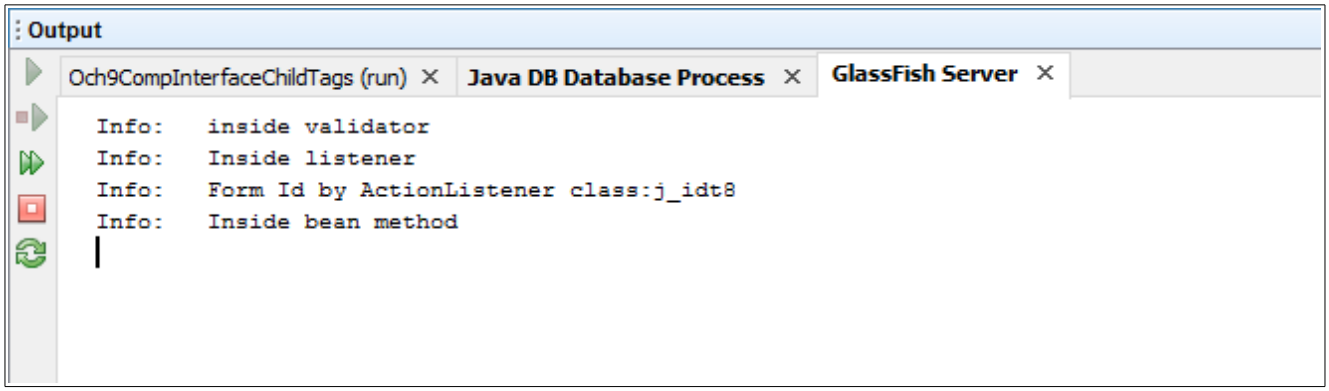
Server Console



2. Valid Inputs

login.xhtml

Server Console



home.xhtml

5. Facets

➔ Adding converters, validators and listeners inside composite component is one way of adding funcationality to it.

Another way is through facets.

➔ Facet is used to encapsulate the set of elements specifying the details for the facet.

➔ Example

```
<composite:interface>

...
<composite:facet name="header"/>
<composite:facet name="error"/>
</composite:interface>
<composite:implementation>
...
<composite:renderFacet name="header"/>
<h:form ...>
...
</h:form>
<composite:renderFacet name="error"/>
</composite:implementation>
```

Later inside the view where composite component has been used facets are used as

```
<comp:component

            <f:facet name="heading" class="header">
                <h:outputText value="Login Page"/><br/><br/>
            </f:facet>

            <f:facet name="error" class="error">
                <h:messages layout="table" style="color: blue"/>
            </f:facet>

            <h:link outcome="register">Register..</h:link>
</comp:component>
```

Above when link is clicked `outcome="register"` mentioned take control to JSF page `register.xhtml`.

6. Children <composite:insertChildren>

➔ This component is used inside <composite:implementation> section.

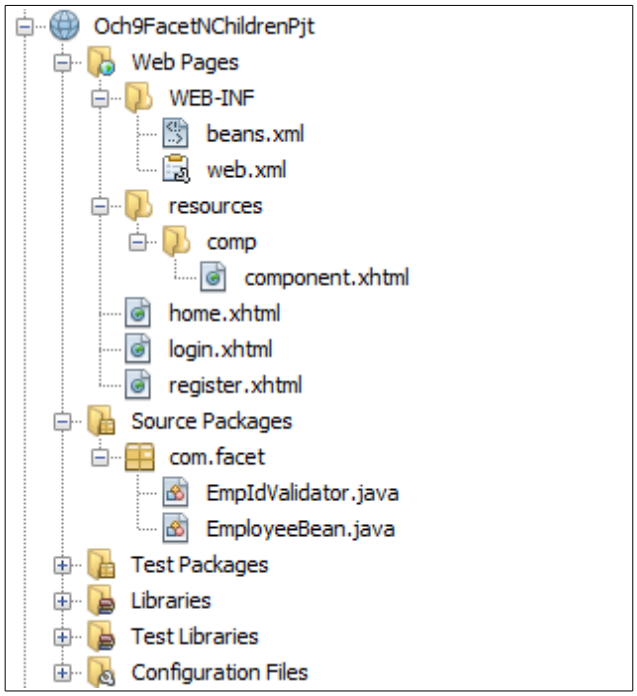
➔ They are used to have component tags placed at a specific point inside composite component.

➔ Example

```
<cc:implementation>
...
<cc:insertChildren />
...
</cc:implementation>
```

➔ Example

Lets create project directory 'Och9FacetNChildrenPjt' as shown below



EmployeeBean.java

```
package com.facet;

import java.io.Serializable;
import javax.inject.Named;
import javax.enterprise.context.SessionScoped;

@Named(value = "employee")
@SessionScoped
public class EmployeeBean implements Serializable {

    public EmployeeBean() {
    }

    String name, empId;

    public String getName() {
        return name;
    }

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

    public String getEmpId() {
        return empId;
    }

    public void setEmpId(String empId) {
        this.empId = empId;
    }

    public String toPage()
    {
        return "home";
    }
```

```
}
```

EmpIdValidator

```
package com.facet;


import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

@FacesValidator("com.validator.empIdValidator")
public class EmpIdValidator implements Validator {

    @Override
    public void validate(FacesContext context,
           UIComponent component, Object value) throws ValidatorException {

        System.out.println("inside validator");
        String [] empIds={"1111","2222","3333","4444"};
        String empId = value.toString();
        boolean match=false;
        for(int i=0; i<empIds.length; i++)
        {
            if(((empId.equals(empIds[i]))))
            {
                match=true;
            }
        }
        if(!match)
        {
            FacesMessage message = new FacesMessage("This EmpID is not Valid");
            message.setSeverity(FacesMessage.SEVERITY_ERROR);
            throw new ValidatorException(message);
        }
    }
}
```

component.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:composite="http://java.sun.com/jsf/composite"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
        <title>Component</title>
    </h:head>
    <composite:interface>
        <composite:attribute name="nameLabel"/>
        <composite:attribute name="name"/>
        <composite:attribute name="empIdLabel"/>
        <composite:attribute name="empId"/>
        <composite:valueHolder name="empId" targets="form:empId"/>
        <composite:attribute name="submitButton"/>
        <composite:attribute name="submitAction" method-signature="java.lang.String
        action()"/>
        <f:facet name="heading"/>
        <f:facet name="error"/>
    </composite:interface>
    <h:body>
    <composite:implementation>
        <composite:renderFacet name="heading"/><br/>
        <h:form id="form">
        <h:panelGrid columns="2">
```

```
        <h:outputLabel value="#{cc.attrs.nameLabel}"/>
        <h:inputText id="name" value="#{cc.attrs.name}"/>
        <h:outputLabel id="empIdLabel" value="#{cc.attrs.empIdLabel}"/>
        <h:inputText id="empId" value="#{cc.attrs.empId}"/>
        <h:commandButton id="submitButton" value="#{cc.attrs.submitButton}"
            action="#{cc.attrs.submitAction}"/>
        </h:panelGrid>
        </h:form>
        <composite:renderFacet name="error"/>
        <composite:insertChildren/>
        </composite:implementation>
        </h:body>

</html>
```

login.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:comp="http://java.sun.com/jsf/composite/comp"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Login Page</title>
    </h:head>
    <h:body>
        <h:form>
            <comp:component
                nameLabel="Name :"
                name="${employee.name}"
                empIdLabel="EmpID :"
                empId="${employee.empId}"
                submitButton="Submit"
                submitAction="#{employee.toPage}">

            <f:validator for="empId" validatorId="com.validator.empIdValidator"/>

            <f:facet name="heading" class="header">
                <h:outputText value="Login Page"/><br/><br/>
            </f:facet>

            <f:facet name="error" class="error">
                <h:messages layout="table" style="color: blue"/>
            </f:facet>

            <h:link outcome="register">Register..</h:link>
            </comp:component>

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

home.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">
    <h:head>
        <title>Home Page</title>
    </h:head>
    <h:body>
```

```
        Hello #{employee.name}
    </h:body>
</html>
```

register.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">
    <h:head>
        <title>Register Page</title>
    </h:head>
    <h:body>
        This is Register page
    </h:body>
</html>
```

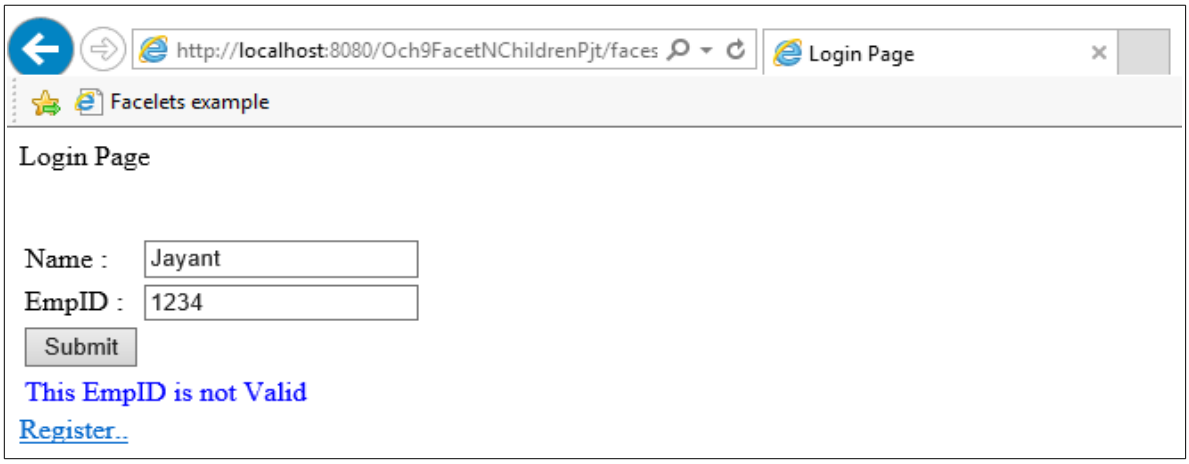
web.xml

```
<?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/login.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Output

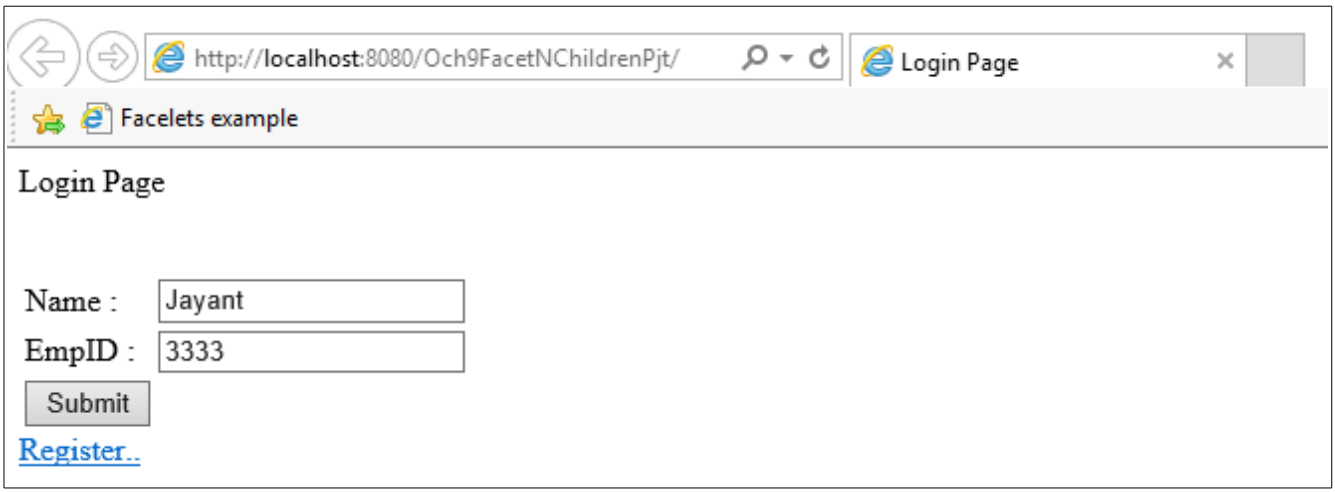
1. When inputs are not valid

login.xhtml



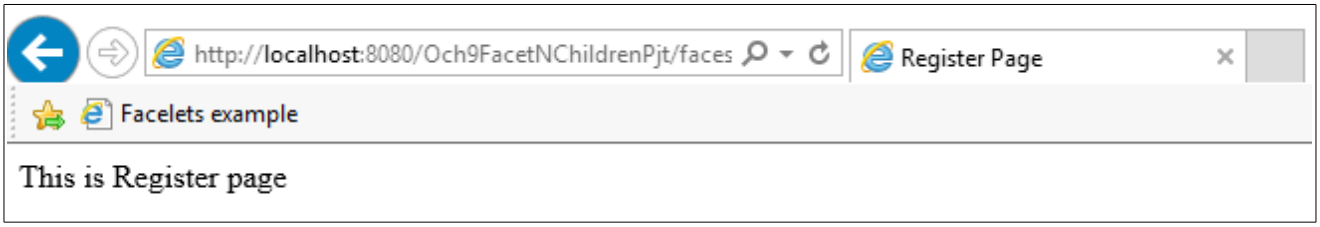
2. Valid Inputs

login.xhtml



3. Regsiter Link

register.xhtml



7. Backing Components

➔