Chapter 1 Getting Started

1. Beans

- A bean must have name and scope.
- Ex

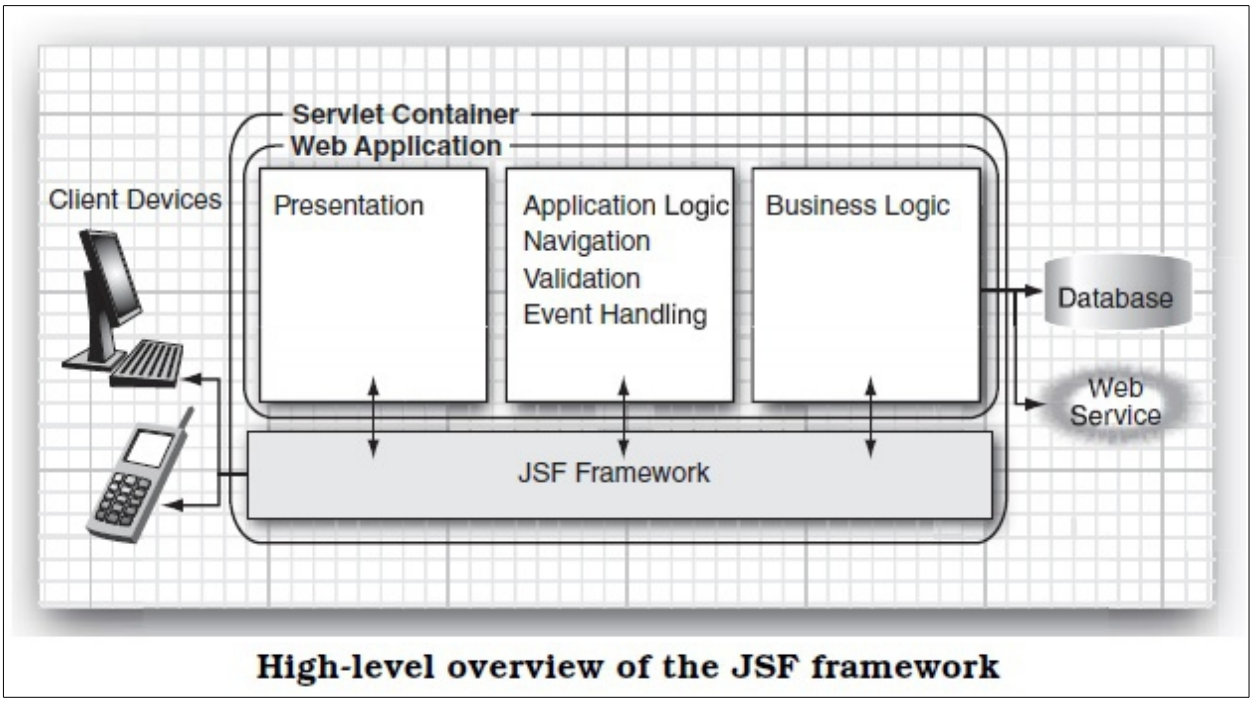
```
@Named('user')
@SessionScoped
```

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

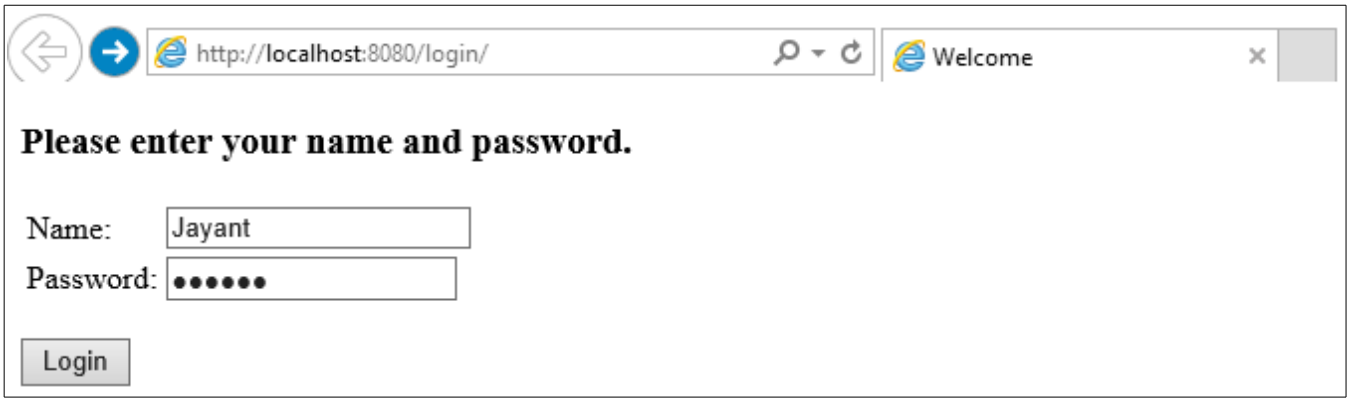
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<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>
```

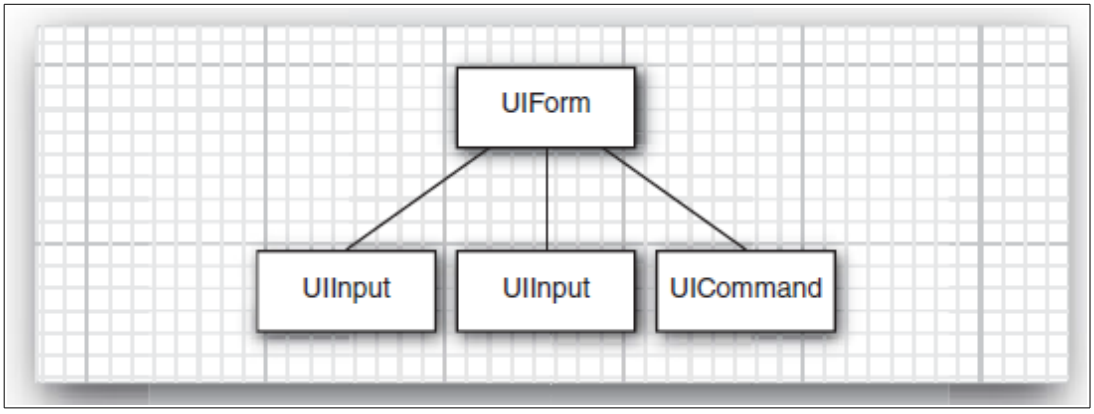
login.xhtml



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.

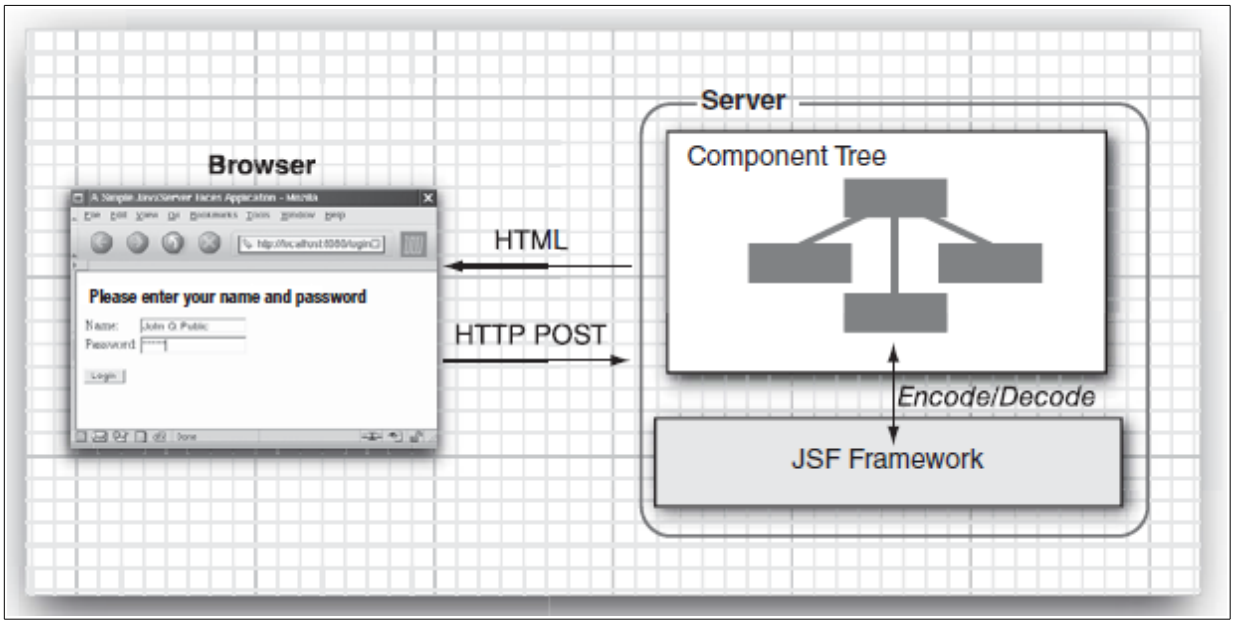


2. Rendering with Encoding: Next, the HTML is rendered. All text which is not JSF tag will pass through. The `h:form`, `h:input`, `h:inputSecret` and `h:commandButton` tags are converted to HTML. For example `h:InputText` component produces following output

`<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.

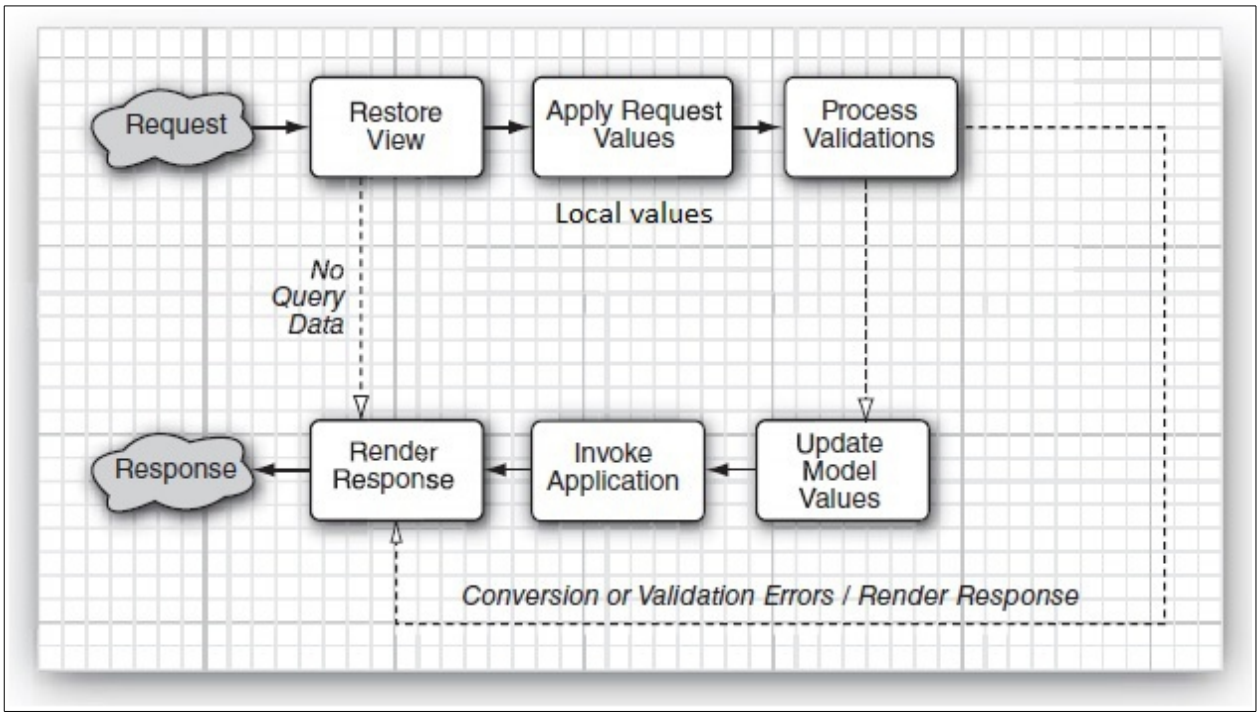
➔ The UIInput components updates the bean properties referenced in the value attribute. They invoke setter methods

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.

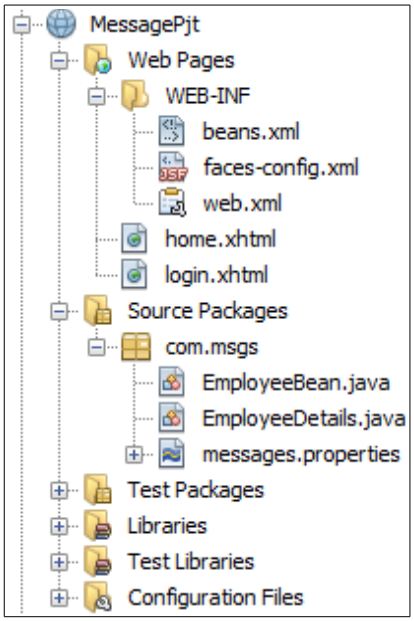
Ex. `email=Email ID is : {0}`

➔ 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.

5. Facelets Example

➔ In a typical JSF application, each page of the application connects to a managed bean that serves the backing bean.

➔ The backing bean defines the method and properties that are associated with the components.

➔ For developing JSF application the following tasks are usually required.

i. Developing the managed beans

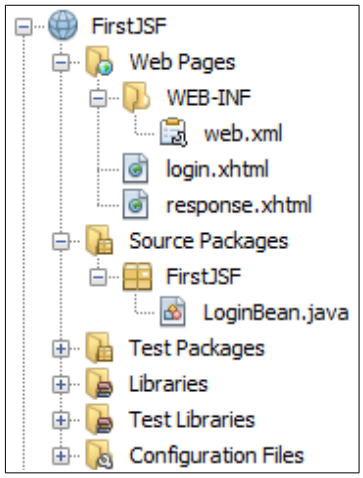
ii. Creating the pages using the component tags

iii. Defining page navigation

iv. Mapping the FacesServlet instance

v. Adding managed bean declarations

➔ Lets have FirstJSF 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 FirstJSF;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.bean.SessionScoped;
import javax.inject.Named;

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

    String fname, lname;
    int id;

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

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

    public void setId(int id) {
        this.id = id;
    }

    public String getFname() {
        return fname;
    }

    public String getLname() {
        return lname;
    }

    public int getId() {
        return id;
    }
}
```

login.xhtml

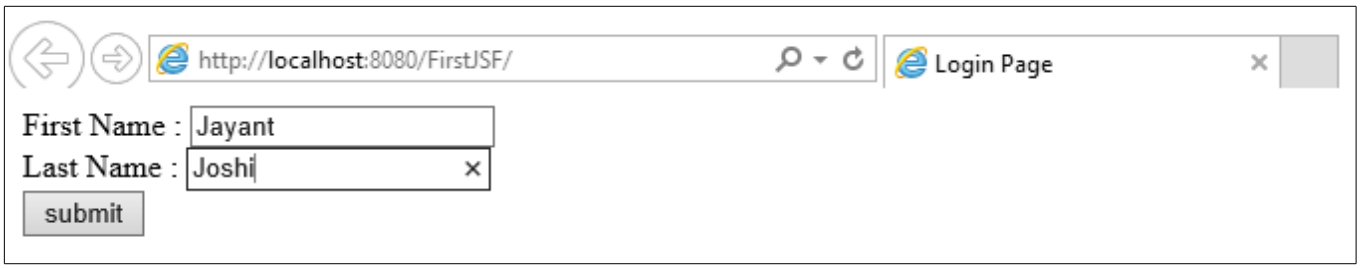
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Login Page</title>
    </h:head>
    <h:body>
        <h:form>
            First Name : <h:inputText id="fname" value="#{LoginBean.fname}"/><br></br>
            Last Name : <h:inputText id="lname" value="#{LoginBean.lname}"/><br></br>
            <h:commandButton id="submit" value="submit" action="response"/>
        </h:form>
    </h:body>
 </html>
```

response.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>Facelet Title</title>
    </h:head>
```

```
    <h:body>
        Hello #{LoginBean.fname} #{LoginBean.lname} !!! <br></br>
        How are you doing ?
    </h:body>
</html>
```

login.xhtml

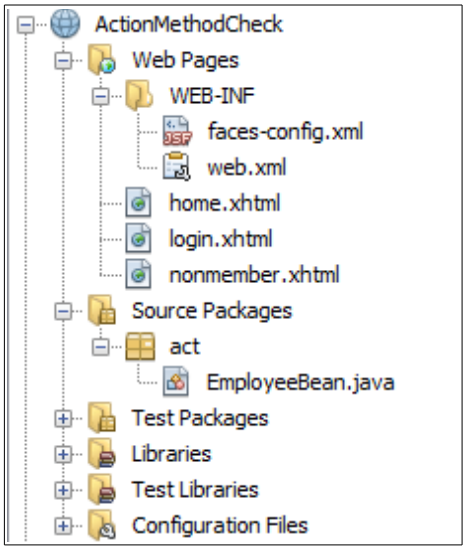


response.xhtml

- ➔ An `<h:inputText>` tag accepts user input and sets the value of the managed bean property `userNumber`.
- ➔ The input value is validated by validator tag `<f:validateLongRange>`.
- ➔ Configuring a JavaServer Faces application involves mapping the Faces Servlet in theweb deployment descriptor file, such as a `web.xml` file, and possibly adding managed bean declarations, navigation rules, and resource bundle declarations to the application configuration resource file, `faces-config.xml`.
- ➔ In `web.xml` we define context parameter PROJECT_STAGE. The stage of application can affect the behavior of application. If PROJECT_STAGE is defined as 'Development' then debugging information is automatically generated. The default stage is 'Production'.

➔ Example : How dynamic navigation takes place through 'action' attribute.

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

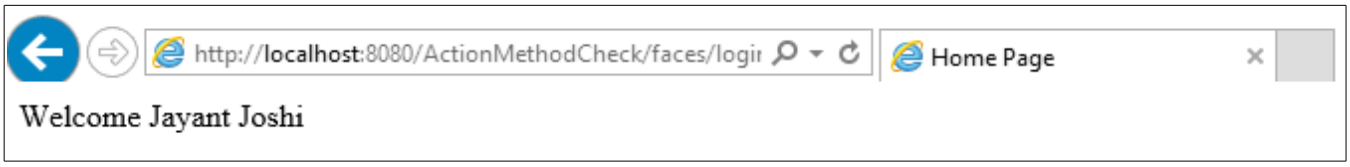
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Non Member Page</title>
    </h:head>
    <h:body>
        Sorry #{empbean.name} you are not allowed !!!
    </h:body>
</html>
```

Output

1)

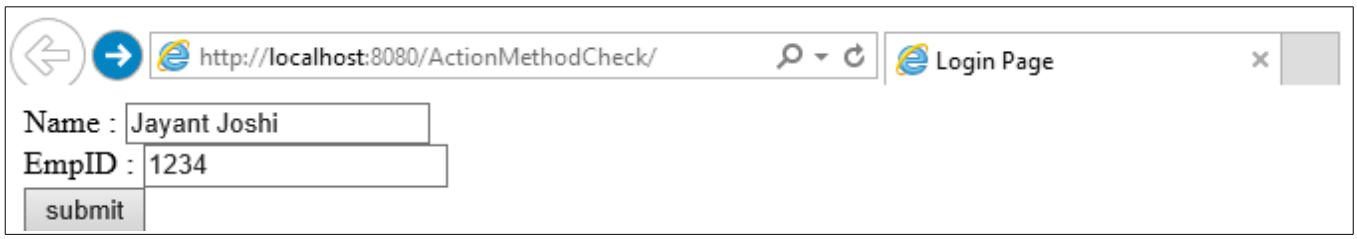
login.xhtml

home.xhtml

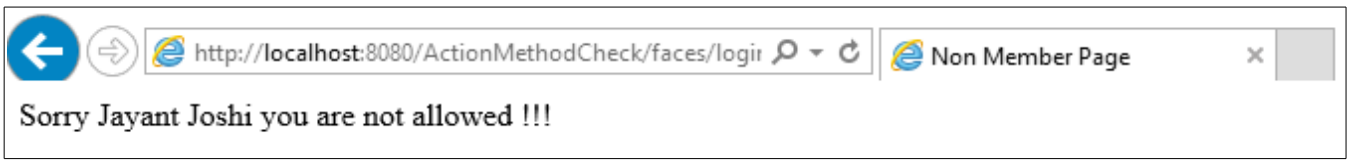


2)

login.xhtml



nonmember.xhtml



6. Facelets Templating Tags

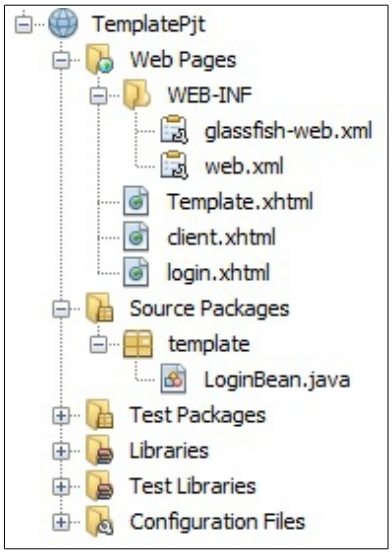
➔ Templating feature allows one to create a page that will act as the base, or template, for other pages in an application. It helps in maintaining standard look and feel in an application with large number of pages.

Facelets Templating Tags

Tag	Function
`ui:component`	Defines a component that is created and added to the component tree.
`ui:composition`	Defines a page composition that optionally uses a template. Content outside of this tag is ignored.
`ui:debug`	Defines a debug component that is created and added to the component tree.
`ui:decorate`	Similar to the composition tag but does not disregard content outside this tag.
`ui:define`	Defines content that is inserted into a page by a template.
`ui:fragment`	Similar to the component tag but does not disregard content outside this tag.
`ui:include`	Encapsulates and reuses content for multiple pages.
`ui:insert`	Inserts content into a template.
`ui:param`	Used to pass parameters to an included file.
`ui:repeat`	Used as an alternative for loop tags, such as `c:forEach` or `h:dataTable`.
`ui:remove`	Removes content from a page.

```
<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

7. Facelets Composite Component

➔ Composite components is a special type of template that acts as a component.

➔ The web page that uses composite component is generally called as **using page**.

Most commonly used composite components are listed below

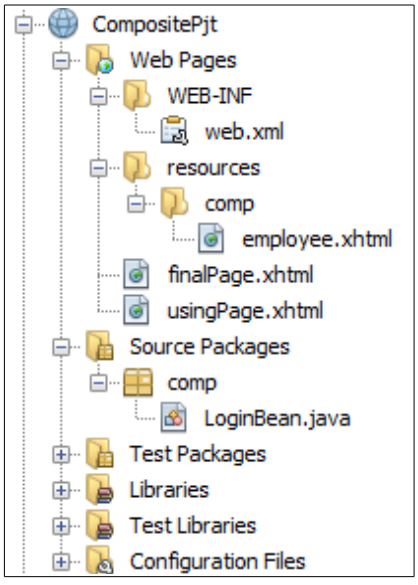
Composite Component Tags

Tag	Function
`composite:interface`	Declares the usage contract for a composite component. The composite component can be used as a single component whose feature set is the union of the features declared in the usage contract.
`composite:implementation`	Defines the implementation of the composite component. If a `composite:interface` element appears, there must be a corresponding `composite:implementation`.
`composite:attribute`	Declares an attribute that may be given to an instance of the composite component in which this tag is declared.
`composite:insertChildren`	Any child components or template text within the composite component tag in the using page will be reparented into the composite component at the point indicated by this tag's placement within the `composite:implementation` section.
`composite:valueHolder`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `ValueHolder` suitable for use as the target of attached objects in the using page.
`composite:editableValueHolder`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `EditableValueHolder` suitable for use as the target of attached objects in the using page.
`composite:actionSource`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `ActionSource2` suitable for use as the target of attached objects in the using page.

➔ **Example**

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

Lets create a CompositePjt 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/usingPage.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

employee.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="emid" required="false"/>
        </composite:interface>

        <composite:implementation>

            <br> <h:outputLabel value ="first name: "/>
            <h:inputText id="fname" value="#{cc.attrs.fname}"/></br>

            <br> <h:outputLabel value="last name: "/>
            <h:inputText id="lname" value="#{cc.attrs.lname}"/></br>

            <br> <h:outputLabel value="Email id: "/>
            <h:inputText value="#{cc.attrs.emid}"/></br>

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

Note : Above the local composite component library is defined in xmlns namespace with declaration

```
xmlns:composite="http://xmlns.jcp.org/jsf/composite"
```

LoginBean.java

```
package comp;

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

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

    String fname, lname, emid;

    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 String getEmid() {
        return emid;
    }

    public void setEmid(String emid) {
        this.emid = emid;
    }
}
```

usingPage.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:em="http://xmlns.jcp.org/jsf/composite/comp">
    <h:head>
        <title>Using Page</title>
    </h:head>
    <h:body>
        <h:form>
            <em:employee
                fname="#{loginbean.fname}"
                lname="#{loginbean.lname}"
                emid="#{loginbean.emid}"
            />

            <h:commandButton id="submit" value="submit" action="finalPage"/>

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

Note : Above we can see the components are accessed through em:employee tag.

finalPage.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>
       <br>Hello #{loginbean.fname} #{loginbean.lname}</br>
       your email id is : #{loginbean.emid}
    </h:body>
</html>
```

Output

usingPage.xhtml

finalPage.xhtml

7. Web Resources

➔ Web resources are software artifacts which application requires for proper working. Resources must be collected at a standard location which will be one of the following.

a. A resource packaged in the web application root must be in a subdirectory of a `resources` directory at the web application root: `resources/`*`resource-identifier`*.

b. A resource packaged in the web application's classpath must be in a subdirectory of the `META-INF/resources` directory within a web application: `META-INF/resources/`*`resource-identifier`*. You can use this file structure to package resources in a JAR file bundled in the web application.

➔ (*) The JSF runtime will look for the resources in the preceding listed locations, in that order.

➔ Resource identifiers are unique strings that conform to the following format.

[*locale-prefix/*][*library-name/*][*library-version/*]*resource-name*[*/resource-version*]

➔ Example :

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

This tag specifies that the `default.css` style sheet is in the directory `web/resources/css`.

```
<h:graphicImage value="#{resource['images:wave.med.gif']}"/>
```

This tag specifies that the image named `wave.med.gif` is in the directory `web/resources/images`.

➔ Resources can be considered as a library location. Any component or template stored in `resource` directory becomes accessible to other application components.

8. <u>Resource Library Contracts</u>

➔ One could have a different look and feel for one or more part of the application. To do this one has to create `contracts` section of web application.

➔ Inside contracts section one can specify number of named areas each of which is called contract. Within each contract one can specify resources such as template files, style sheets, JavaScript files and Images.

➔ For example, two contracts c1 and c2 are as shown below

```
src/main/webapp
    WEB-INF/
    contracts
        c1
            template.xhtml
            style.css
            myImg.gif
            myJS.js
        c2
            template.xhtml
            style2.css
            img2.gif
            JS2.js
    index.xhtml
```

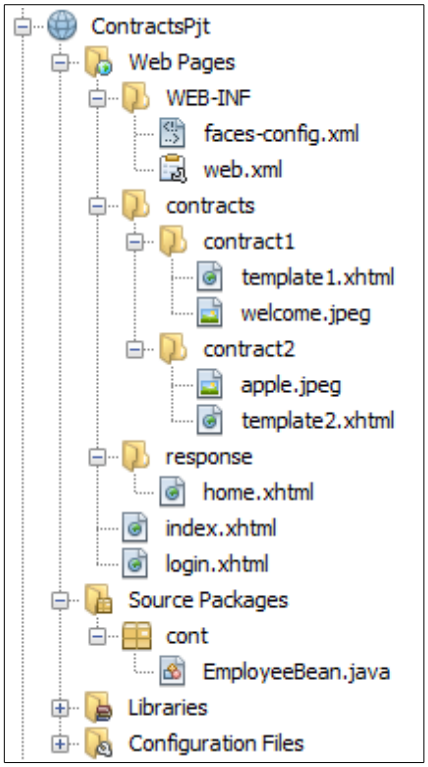
➔ One can package resource library contract in a JAR file to reuse in different applications. If one do so then contract must be located under META-INF/contracts. JAR file then can be placed under WEB-INF/lib directory of an application.

➔ (*) One can also specify contract usage within the applcation's `faces-config.xml` file under `resource-libarary-contracts` element as shown below. This element will be used only if application is using more than one contract.

```
<resource-library-contracts>
      <contract-mapping>
            <url-pattern>/reply/*</url-pattern>
            <contracts>reply</contracts>
      </contract-mapping>
      <contract-mapping>
            <url-pattern>*</url-pattern>
            <contracts>hello</contracts>
      </contract-mapping>
</resource-library-contracts>
```

➔ **Example**

Lets create a directory ContractsPjt 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-library-contracts>
```

```
            <contract-mapping>
                <url-pattern>*</url-pattern>
                <contracts>contract1</contracts>
            </contract-mapping>
            <contract-mapping>
                <url-pattern>/response/*</url-pattern>
                <contracts>contract2</contracts>
            </contract-mapping>
        </resource-library-contracts>
    </application>
</faces-config>
```

template1.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:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Template 1</title>
    </h:head>
    <h:body>
        <h1>Welcome to Login Page</h1>
        <h:graphicImage url="#{resource['welcome.jpeg']}"/><br/>
        <ui:insert name="content"/>
    </h:body>
</html>
```

template2.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:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:head>
        <title>Template 2</title>
    </h:head>
    <h:body>
        <h1>Home Page</h1>
        <h:graphicImage url="#{resource['apple.jpeg']}"></h:graphicImage><br/>
        <ui:insert name="message"/>
    </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:ui="http://xmlns.jcp.org/jsf/facelets" >

    <h:head>
        <title>Login Page</title>
    </h:head>

    <ui:composition template="/template1.xhtml">

         <ui:define name="content">
                 <h:form>
                   First Name : <h:inputText id="fname" value="#{empbean.fname}"/><br/>
                   Last Name  : <h:inputText id="lname" value="#{empbean.lname}"/><br/>
                   <h:commandButton id="submit" value="Submit" action="response/home">
```

```
                    </h:commandButton>
                </h:form>
            </ui:define>

    </ui:composition>

</html>
```

home.xhtml

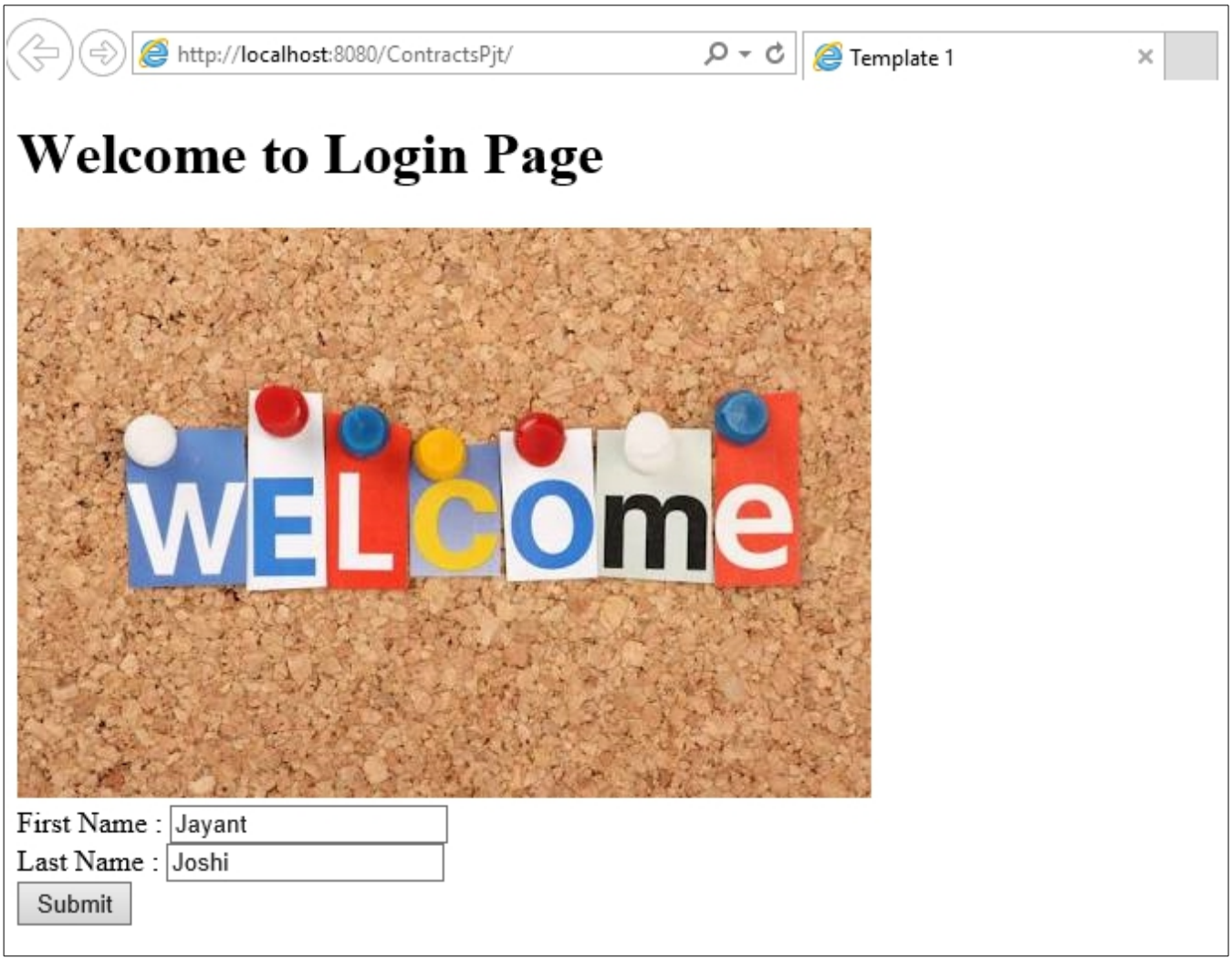
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

    <h:head>
        <title>Home Page</title>
    </h:head>
    <ui:composition template="/template2.xhtml">

            <ui:define name="message">
                <h:body>
                    Hello #{empbean.fname} #{empbean.lname}
                </h:body>
            </ui:define>
    </ui:composition>

</html>
```

LoginBean.java

```
package cont;


import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
/**
 * @author Jayant
 */
@ManagedBean(name="empbean")
@SessionScoped
public class EmployeeBean {
    public EmployeeBean() {
    }

    String fname,lname;

    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;
    }
}
```

Output

login.xhtml



home.xhtml