Java Web Application Frameworks

Developed By :

Dr. Kamlendu Pandey

Let us look Back to JSTL/EL

Advantages

1.First time there was an effort to separate the concerns of Design and Coding (Writing core business logic). It came with many useful tag libraries which almost eliminated writing any Java code in JSP. All the core business code behind was supposed to be written as Java Classes and Beans (Reusable Java Components)

2.One can create custom set of Tags and tag library or enhance the existing tags as per requirements

3.It opened the immense possibilities of writing better and efficient web applications Eliminated writing scriplet tags. This become a milestone and the basis of developing frameworks.

Advantages

1.Expression Language (EL) gave a simple to write, use and access the implicit objects like request, response, session, application etc. It also helped in eliminating Java in JSP

2.One need not be Java Expert in writing JSTL/EL . A simple understanding the functioning, arguments and parameters of the tags is enough to write the application. The code behind the tag , classes and beans were supposed to be written by Java programmers

Limitations

- JSTL/EL are not independent but a subset of JSP. While Executing JSTL tags translated into equivalent JSP by virtue of the tag handler classes. Again this JSP was translated into Servlets.

- With this we added one more layer of translation. So there was increase in the latency time of the response to client although only once as later on the servlet was always in the instance of the webserver till it is collected back by the webserver.

- On even trivial modifications whole process of re-translation, re-compilation and re-instantiation was repeated and was required to be redeployed.

-

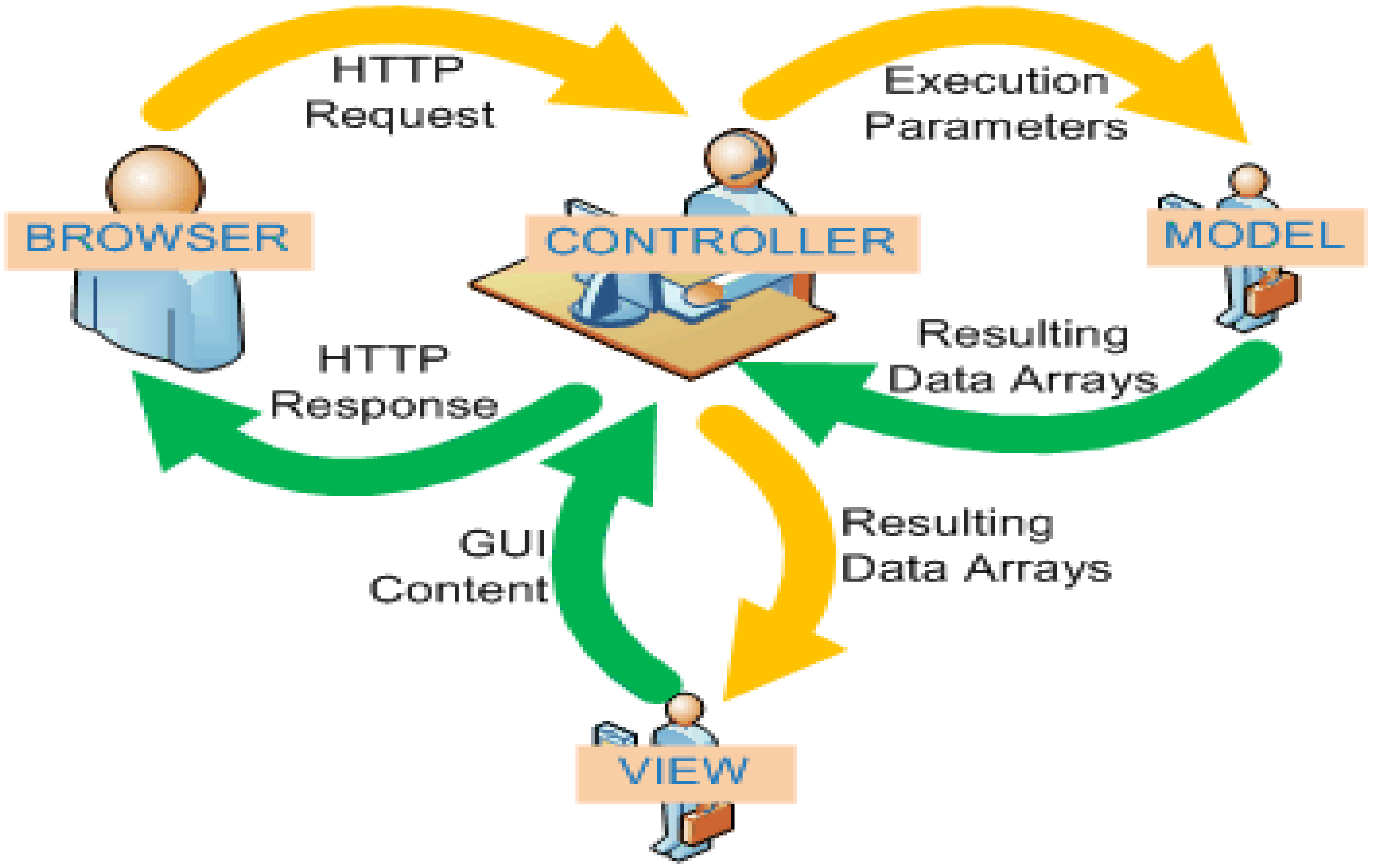
We Require a solution to these limitations and JAVA EE provides the solution by A Design Pattern Called .....

The MVC Framework

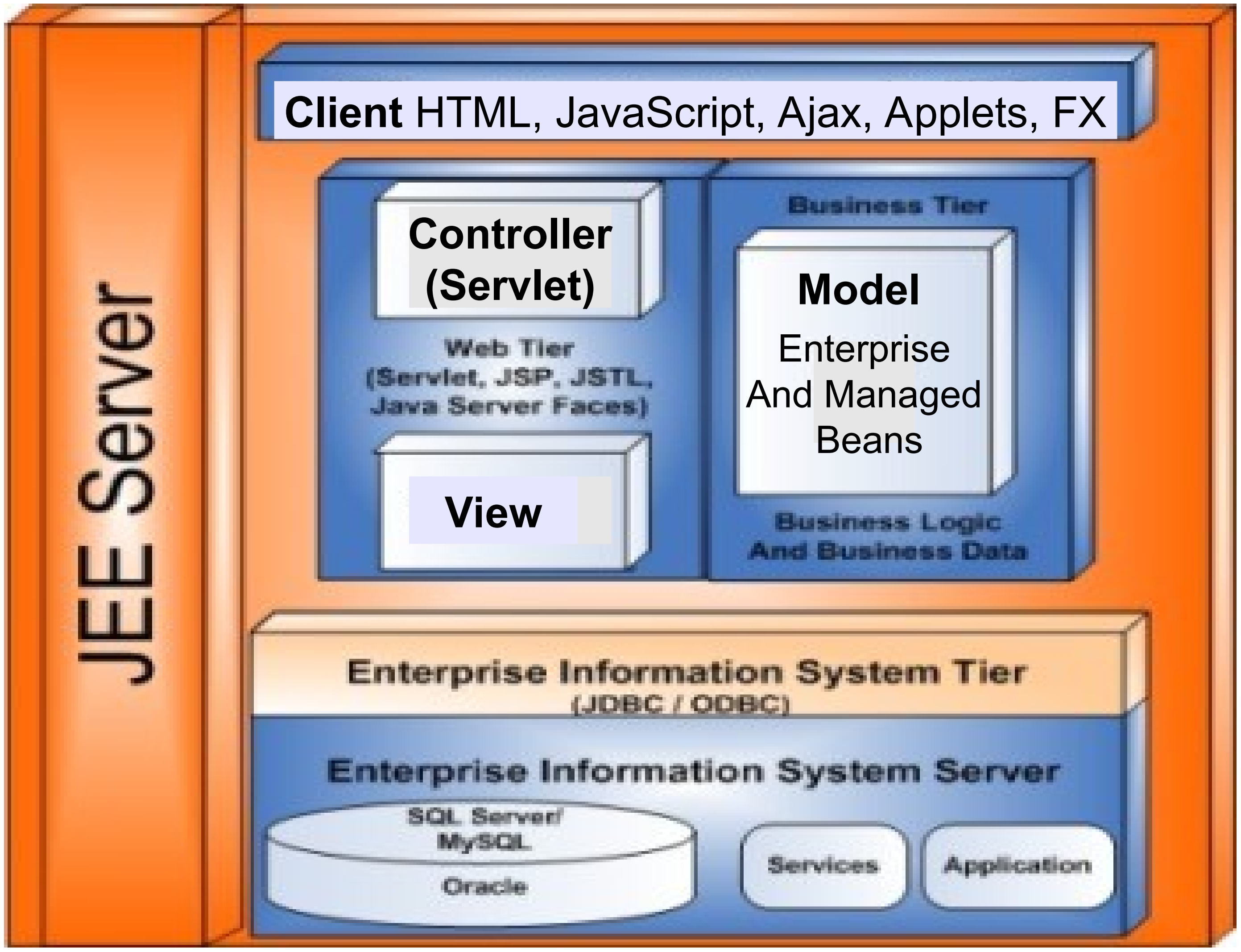
MVC Architecture..

MVC became an important milestone in web application development as the Design and Architectural pattern Normally known as Model View Control was actually based upon separation of concerns as well as establishing a centralized operational control on the application.

MVC Architecture (contd.)..



By Virtue of its Wonderful Architecture


JEE Server
Client HTML, JavaScript, Ajax, Applets, FX
Controller (Servlet)
Web Tier (Servlet, JSP, JSTL, Java Server Faces)
View
Business Tier
Model
Enterprise And Managed Beans
Business Logic And Business Data
Enterprise Information System Tier (JDBC / ODBC)
Enterprise Information System Server
SQL Server/ MySQL
Oracle
Services
Application

Model in MVC Architecture

- The Model object knows about all the data that need to be displayed. It also knows about all the operations that can be applied to transform that object.
- It knows nothing about the GUI, the manner in which the data are to be displayed, nor the GUI actions that are used to manipulate the data.
- The data are accessed and manipulated through methods that are independent of the GUI.
- Simply saying the model represents enterprise data and the business rules that govern access and update of the data.

View in MVC Architecture

– The view object queries the model.

– It uses the query methods of the model to obtain data from the model and then displays the information.

– A view renders the contents of a model. It accesses enterprise data through the model and specifies how that data should be presented.

– It is the view's responsibility to maintain consistency in its presentation when the model changes.

Controller-MVC Architecture

- The controller object knows about all the resources used in the project
- A controller translates interactions with the view into actions to be performed by the model.
- The actions performed by the model include activating business processes or changing the state of the model.
- Based on the user interactions and outcome of the model actions, the controller responds by selecting an appropriate view.

Java MVC Frameworks

Under Java Community Process (JCP) several frameworks came into existence with their unique implementation of Ideas . Some of the popular frameworks are

Component Based Frameworks

- » **Java Server Faces**
- » **Apache MyFaces**
- » **IceFaces**

Action Based Frameworks

- » **Struts**
- » **Spring**

Component Based Frameworks

Web framework based on component driven UI design model, using XML files called view templates . The components are the part of view. Any event on the component is handled by the back-end logic layer

Action Based Frameworks

Action Based Framework give us the ability to map URLs to activities and code on the back end. The URL can be directed to a Java Action Class (Business logic) which handles the URL and return an appropriate view

Java Server Faces (JSF)

- A Component Based Java Web Framework Implementing MVC Design Pattern

Java Server Faces (JSF)..

Introduction

Java Server Faces is a framework that simplifies development of sophisticated web application user interfaces, primarily by defining a user interface component model tied to a well defined request processing life cycle.

Java Server Faces - Objectives

- To provide event-driven and component based technology for developing web applications.

- To separate the user interface from the model objects that encapsulate the data and application logic.

- To provide the server side validation and data conversion.
- To retain the state of the components.

JSF Includes Features like..

- ✔Page navigation control
- ✔Standard user interface components like input fields, buttons, and links, checkboxes etc.
- ✔Type conversion
- ✔User input validation and error handling
- ✔Java bean management
- ✔Event handling
- ✔Internationalization support
- ✔Templating and Custom Components

JSF Buiding Blocks

- Model -CDI Beans, Entity Set, Facades, Enterprise Java Beans , POJO, POJI etc.

- View –Views, Facelets, HTML, JSP, EL JavaFX

- Control – Implicit Annotation-based and Explicit Controller. (faces-config.xml)

Facelets

The New Avatar of JSF

Facelets – Why New ?

•Because it takes Java Completely out of the Web Page .

•While the earlier versions of JSF tags were written with in a JSP and were subjected to the process of servlet generation – compilation-loading, the facelets gave complete freedom from JSP

•That means you don,t require a JSP for writing JSF

So what do we do Now ....

• We write Facelets in XHTML a improved form of HTML

JSF pre 2.0–What earlier Versions did ?

Let us look at the following piece of code written in earlier JSF – **HelloWorld.jsp**

Still A JSP

Including Tag libraries just like JSTL

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<html>
   <head><title>Managed Bean Example(User
Page)</title></head>
   <body>
     <f:view>
       <f:form>
         <h:outputText value="Hello World"></h:outputText>
   </f:form> </f:view>
  </body>
</html>
```

Embedding JSF Tags in JSP

Subject to translation into .java , compiled to .class and then loaded by servlet engine on request

Facelets–What is New ?

No longer a JSP it is xhtml

It is simple xml with some namespaces
To add tags, no taglibs

HelloWorld.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>Text Input Demo</title>
    </h:head>
    <h:body>
        <h1> <h:outputText value="Hello World”/> </h1>
</h:body>
</html>
```

No Embedding JSF Tags in JSP as
It is not a JSP

No translation or compilation. Only a
View tree is maintained which is supplied
To browsers for rendring

The CDI Beans..- *Part of Model*

- In a JSF application, CDI beans (Backing Beans) are used that can be associated with UI components. Backing Beans are nothing but Java Beans which defines properties and methods. These properties are bound to component values or component instances. setX() and getX() methods are defined and implemented for all properties.
- One can also define methods that can be used for different purpose like event handling, navigation, validation etc. that are associated with component.

CDI Bean are used to

- Bind value of the component to the property of the bean or to refer method of the bean from component tag, Expression Language (EL) syntax is used.
- For example, #{BeanName.propertyName}can be used to bind the value of the component to the property "propertyName" of the bean
- "#{BeanName.methodName} can be used to refer method "methodName()" of the bean "BeanName".
- The return value of bean method decides the next View by the controller
- The Beans can be be scoped to request, session and application

A CDI Bean looks like

Step 1: Create a JavaBean file "MyBean.java"

```
package mscit;
@RequestScoped
public class MyBean{;
    private String userName;

  public void setUserName(String userNAame){
    this.userName=userName;}

  public String getUserName(){
     return userName;}

  public String submit {
     return “ShowMe”; }
}
```

Scope of Bean

Property of Bean

Getting and Setting the Property of Bean

A method of Bean, it's Return value is processed by Controller for navigation

Facelet using MyBean

EnterName.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<Doctype tag>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head><title>Text Input Demo</title></h:head>  <h:body>
<h:form>
<h:outputText value="Enter User Name”/>

<h:inputText value="#{MyBean.userName}”/> </br>
<h:commandButton  value="Submit” action= ”#{MyBean.submit}”/>
</h:form>
</h:body>
</html>
```

Binding a component with Bean Property

Calling Bean Method

ShowMe Facelet

ShowMe.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<Doctype tag>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head><title>Text Input Demo</title></h:head>  <h:body>
  <h1>Welcome ${MyBean.userName} </h1>
</h:body>
</html>
```

Using EL to get userName Property of MyBean. The State is maintained as Bean is Request Scoped

The Controller

The Controller is implemented by ***"faces-config.xml"***

A Special XML file which is used to register all the application resources like

- CDI Beans and their scope and properties
- Web Application Listeners
- Navigational Rules
- Custom Converters
- Custom Validators
- Resource Bundles

Example of Controller-faces-config.xml

```
<faces-config version="2.0"    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">
<managed-bean>
   <managed-bean-name>MyBean</managed-bean-name>
   <managed-bean-class>mscit.MyBean</managed-bean-class>
   <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<navigation-rule>
    <from-view-id>EnterName.xhtml</from-view-id>
        <navigation-case>
            <from-outcome>ShowMe</from-outcome>
            <to-view-id>ShowMe.jsf</to-view-id>
        </navigation-case> </navigation-rule></faces-config>
```

Registering Beans

Setting Navigation Rule.
What about jsf extention?
The answer is in web.xml

Deployment Descriptor web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app >
   <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>*.jsf</url-pattern>     </servlet-mapping>
    <session-config> <session-timeout> 30 </session-timeout> </session-config>
    <welcome-file-list>  <welcome-file>index.jsf</welcome-file></welcome-file-list>
</web-app>
```

Registering Java Facelet

All facelets will be requested With .jsf extension

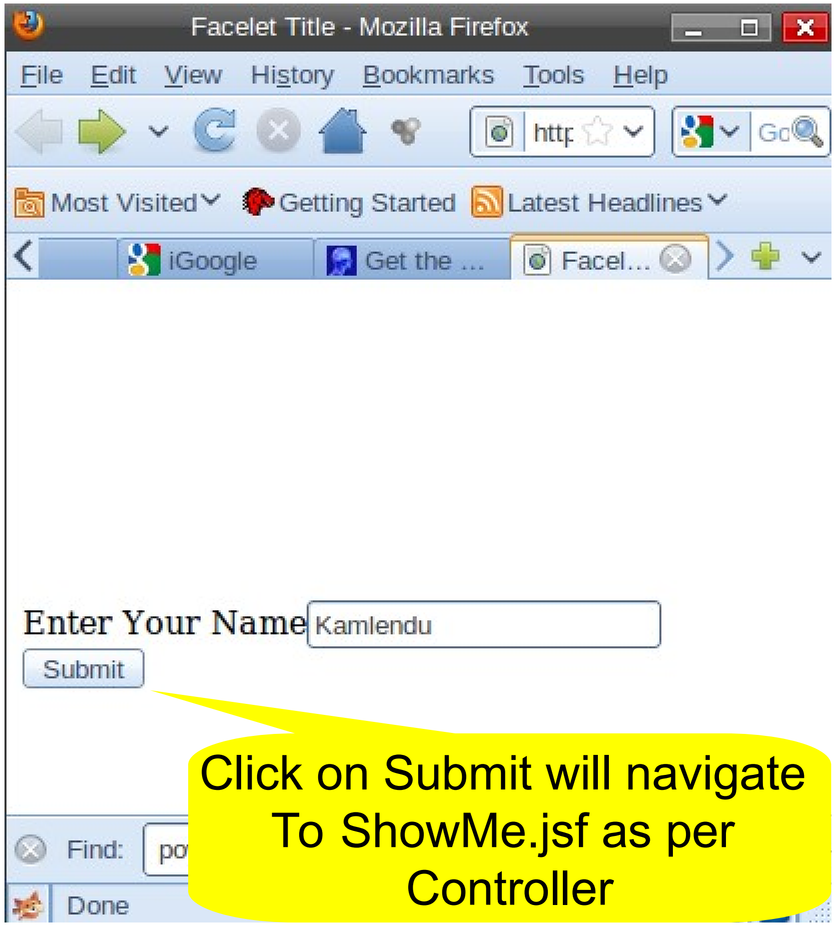
Steps to Run the App

- Build or Compile the Application see that all the necessary Java EE jar files are in the class path. Most of the IDEs like Netbeans will include all the necessary files in the classpath

- Deploy your Application on your favourite web or application server. All the Java web servers are supporting Java Server Faces

- Open your browser and type the following URL
  http://localhost:8080/MyJSFApp/EnterName.jsf

The Output

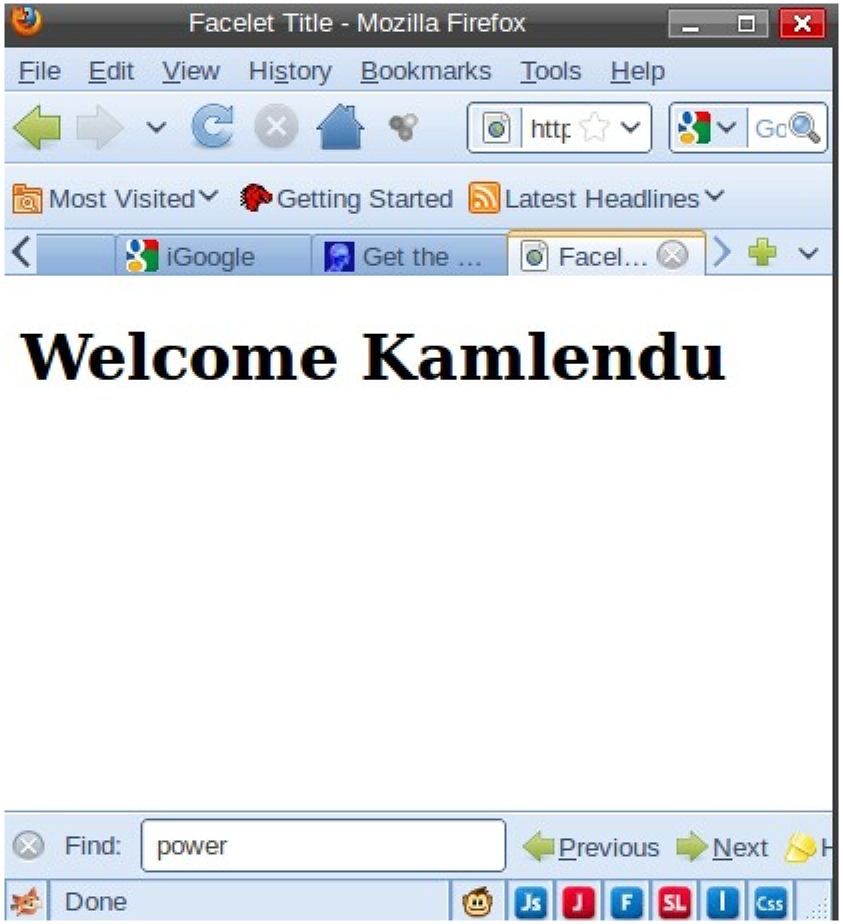

Facelet Title - Mozilla Firefox

Enter Your Name Kamlendu

Submit

File Edit View History Bookmarks Tools Help
Most Visited Getting Started Latest Headlines
iGoogle Get the ... Facel...
Find: po
Done

Click on Submit will navigate To ShowMe.jsf as per Controller


Facelet Title - Mozilla Firefox

Welcome Kamlendu

File Edit View History Bookmarks Tools Help
Most Visited Getting Started Latest Headlines
iGoogle Get the ... Facel...
Find: power Previous Next
Done

http://localhost:8080/MyJSFApp/EnterName.jsf ShowMe.jsf

Java Server Faces UI Components

•The JSF Facelet Architecture has tag library for almost all User Interface Components of Web Application. Some of the listed are

General Name of Component	JSF Name of Component
Static Text	OutputText
TextBox	InputText
Password Box	SecretText
Button	CommandButton
Anchor	CommandLink
Table	DataTable

Java Server Faces UI Components

General Name of Component	JSF Name of Component
CheckBoxGroup	SelectManyCheckBox
Radio Button	SelectOneRadio
List	SelectOneList
Multiple Selection List	SelectManyList
Image	GraphicImage
TextArea	InputTextArea

UI Components..

<h:inputText> & <h:commandButton>

Example:

```
<h:form>
  <div>
    <h:outputLabel for="name" value="Enter your name: " />
    <h:inputText id="name" value="#{backingBean.name}" />
    <h:commandButton value="Submit" />
  </div>
</h:form>
```

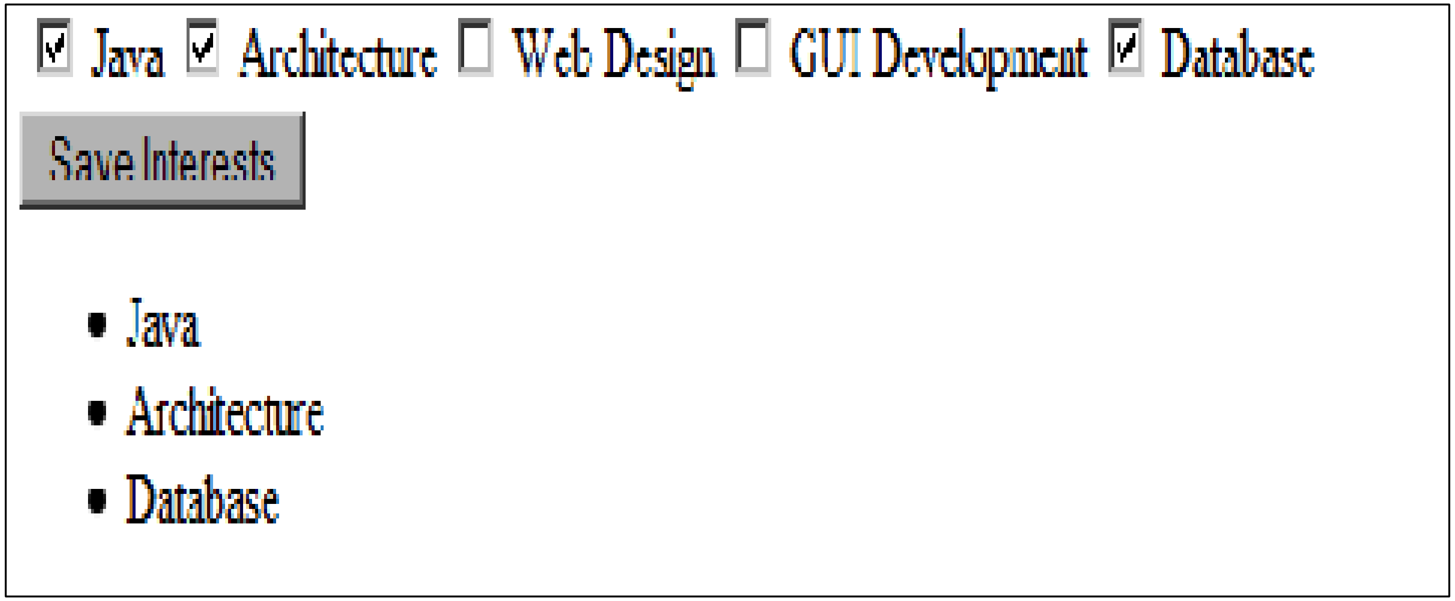
CheckBox Group

Example:

```
<h:selectManyCheckbox id="interests"
value="#{customerBean.interests}" layout="pageDirection">
        <f:selectItem itemLabel="Java" itemValue="Java" />
        <f:selectItem itemLabel="Architecture" itemValue="Architecture" />
        <f:selectItem itemLabel="Web Design" itemValue="Web Design" />
       <f:selectItem itemLabel="GUI Development" itemValue="GUI”    />
        <f:selectItem itemLabel="Database" itemValue="Database" />
</h:selectManyCheckbox>
```

UI Components..

<h:selectManyCheckbox>

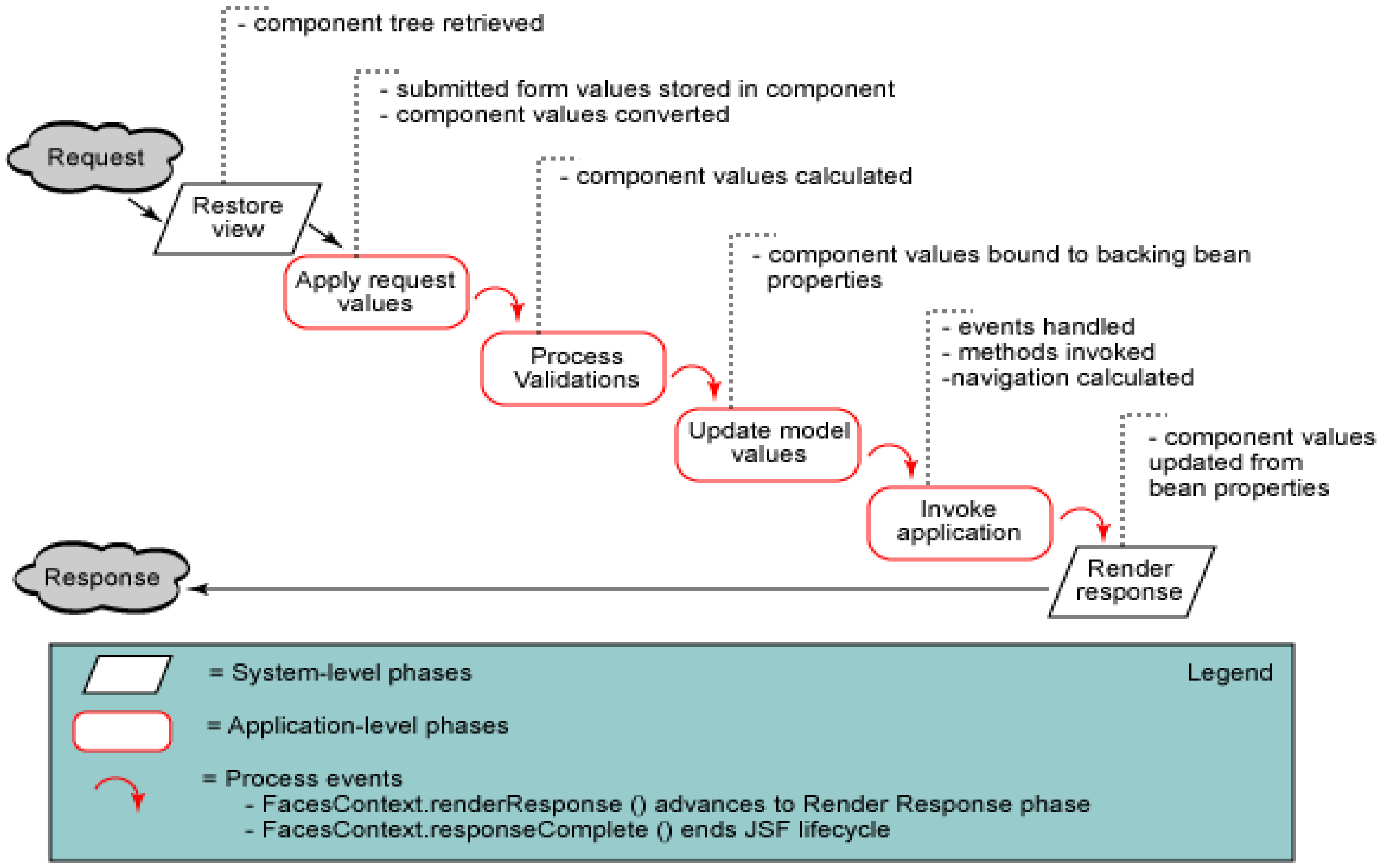


☑ Java ☑ Architecture ☐ Web Design ☐ GUI Development ☑ Database

- Java
- Architecture
- Database

Let us see the usage of other components practically in the applicaton code

JSF Life cycle..



- component tree retrieved
- submitted form values stored in component
- component values converted
- component values calculated
- component values bound to backing bean properties
- events handled
- methods invoked
-navigation calculated
- component values updated from bean properties
Request
Restore view
Apply request values
Process Validations
Update model values
Invoke application
Render response
Response
= System-level phases
Legend
= Application-level phases
= Process events
- FacesContext.renderResponse () advances to Render Response phase
- FacesContext.responseComplete () ends JSF lifecycle

Validators and Converters - *User Input Validation*

- It is important to protect users from sending unexpected , unrelated and irrelevant data from the user interface. Validators and Converters are the best way to achieve it
- **Validators** check the rules of sending the data while **Converters** improve the format of data to be displayed or send
- JSF provides us many in-built validators and Convertors to be used by the programmers
- You can always make your own custom validators and Convertors if you want additional ones
-

Standard/in-built Validators

There are three forms of validation within JSF:

* Built-in validators
* Bean Validators
* Custom Validators

Standard/in-built Validators

The list of Built-in Validators are as given below

- Required Validator – Doesn't allow the blank data to be send
- Range Validator Checks the value is in min and max range
- Pattern Validator Checks the presence of set of characters in input using regular expressions
- Long Range Validators
- Double Range Validator
- Email Validator etc...

Standard/in-built Validators

Using length validator for Text Field

 <h:inputText id="first" value="#{UserRegistration.user.first}">

•<f:validateLength minimum="2" maximum="25" />

</h:inputText>

 If the validation fails, The error message is displayed where-ever the following tag is placed

• <h:message for="first" styleClass="errorMessage" />.

To display stack of all the errors in page use

<h:messages id="hmsg"/>

Custom Validation..

```
public class EmailValidator implements Validator{
      public void validate(FacesContext context, UIComponent
            component, Object value) throws ValidatorException
             {
                  if (email.indexOf("@") == -1) {
           FacesMessage facesMessage = new FacesMessage("Invalid
Email", "Invalid Email");
           FacesContext.getCurrentInstance().addMessage("Invalid Email",
facesMessage);
           facesMessage.setSeverity(FacesMessage.SEVERITY_ERROR);

           System.out.println("FACES MESSAGE: _ " + facesMessage);
           throw new ValidatorException(facesMessage);} }
```

Using Custom Validator..

```
<!-- Put this tag into faces-config.xml
<validator>
        <validator-id>emailValidator</validator-id>
        <validator-class>CustomValidator.EMailValidator</validator-
class>
    </validator>
```

Using the Custom Validators in Facelets

```
<h:inputText value="#{ValidationBean.emailId}" id="txtEmail" >
<f:validator validatorId="emailValidator"></f:validator>
</h:/inputText>
```

Converters

- Following Built-in Converters are provided by JSF
- BigDecimalConverter
- BigIntegerConverter
- .BooleanConverter
- ByteConverter
- CharacterConverter
- DateTimeConverter
- .DoubleConverter
- FloatConverter
-

Converters - Example

Converting a date value into short

```
<h:outputText value="#{ConverterBean.dateVar}">
<f:convertDateTime type="date" dateStyle="short" />
</h:outputText>
```

output 05/09/10

- Converting a number into percentage

```
<h:outputText value="#{ConverterBean.integerVar}">
              <f:convertNumber type="percent"/></h:outputText>
```

- *Output : 45.33 %*

Custom Converter

//Creating Custom Phone Converter to convert it to String

```
public class PhoneConverter implements Converter {
public String getAsString(FacesContext context,
                                        UIComponent component, Object value) {
            return value.toString();
      }}
```

Registering Custom Converter in faces-config.xml

```
<converter>
<converter-id>PhoneConverter</converter-id>

<converter-class>converters.PhoneConverter</converter-class></converter
>
```

Using the Custom Convertor

```
<h:inputText id="phone" value="#{UserRegistration.user.phone}">
<f:converter  converterId="PhoneConverter" /></h:inputText>
```

The Ajaxian Validation

One of the finest characterisics of JSF is a inherent support for Ajax. The code below shows the Ajaxian Submission and validation at form level

```
<h:form>
  . . . . // Input Components
 <h:commandButton id="cmdLogin" value="Login"
action="#{indexBean.login}">
  <f:ajax event="action" execute="@form" render="hmsg" />
        </h:commandButton>
</h:form>
<h:messages id="hmsg"/>
```

This is going to validate the form without refreshing the page

JSF Listeners

JSF UI Components emit Events to Listeners if they are registered. The Event Handling Model in Java Server Faces is very similar to Java Beans Event Handling Model. The types of Events emitted will fall either in Action Event, Value-Change Event or Phase Event.

Listener Tags are identified for handling the various types of Events we discussed just before. They are mentioned below. Let us cover in detail in the subsequent sections.

* Action Listener Tag
* Value Change Listener Tag
* Phase Listener Tag

JSF Listener – Action Event

```
// The Listener Tag used in facelets
<h:commandButton value = "Submit">
    <f:actionListerner type = "mylisteners.SubmitAction"/>
</h:commandButton>
// The Class Behind it
package mylisteners;
import javax.faces.event.*;
public class SubmitAction implements ActionListener{

    public void processAction(ActionEvent ActionEvent){
        // Actual Logic Here.
    }
}
```

JSF Listener - ValueChanged

```
// The Listener Tag used in facelets
<h:inputTextField>
<f:valueChangeListener
type="listeners.MyValueChangeListener" /></h:inputTextField>
// The Class Behind it
package listeners;import javax.faces.event.*;
public class MyValueChangeListener{
   public void processValueChange(ValueChangeEvent vcEvent)
{
      String oldValue = (String)vcEvent.getOldValue();
      String newValue = (String)vcEvent.getNewValue();
      // Do Something with the old and the new value.} }
}
```

JSF Listener – Phase Event

```
// The Listener Tag used in facelets
<f:view>
   <f:PhaseListener type = "mylisteners.MyPhaseListener"/>
</f:view>
// The Class Behind it
package mylisteners;import javax.faces.event.*;
public MyPhaseListener implements PhaseListener{
public void beforePhase(PhaseEvent PhaseEvent){ ....   }
public void afterPhase(PhaseEvent PhaseEvent){ .....   }
public PhaseId getPhaseId(){
      return PhaseId.ANY_PHASE;
   }}
```

Registering the Listeners

```
<!-- Put this tag into faces-config.xml
<faces-config>
   <lifecycle>
      <phase-listener>PhaseTracker</phase-listener>
   </lifecycle>
</faces-config>
```

Lets Go Local or International

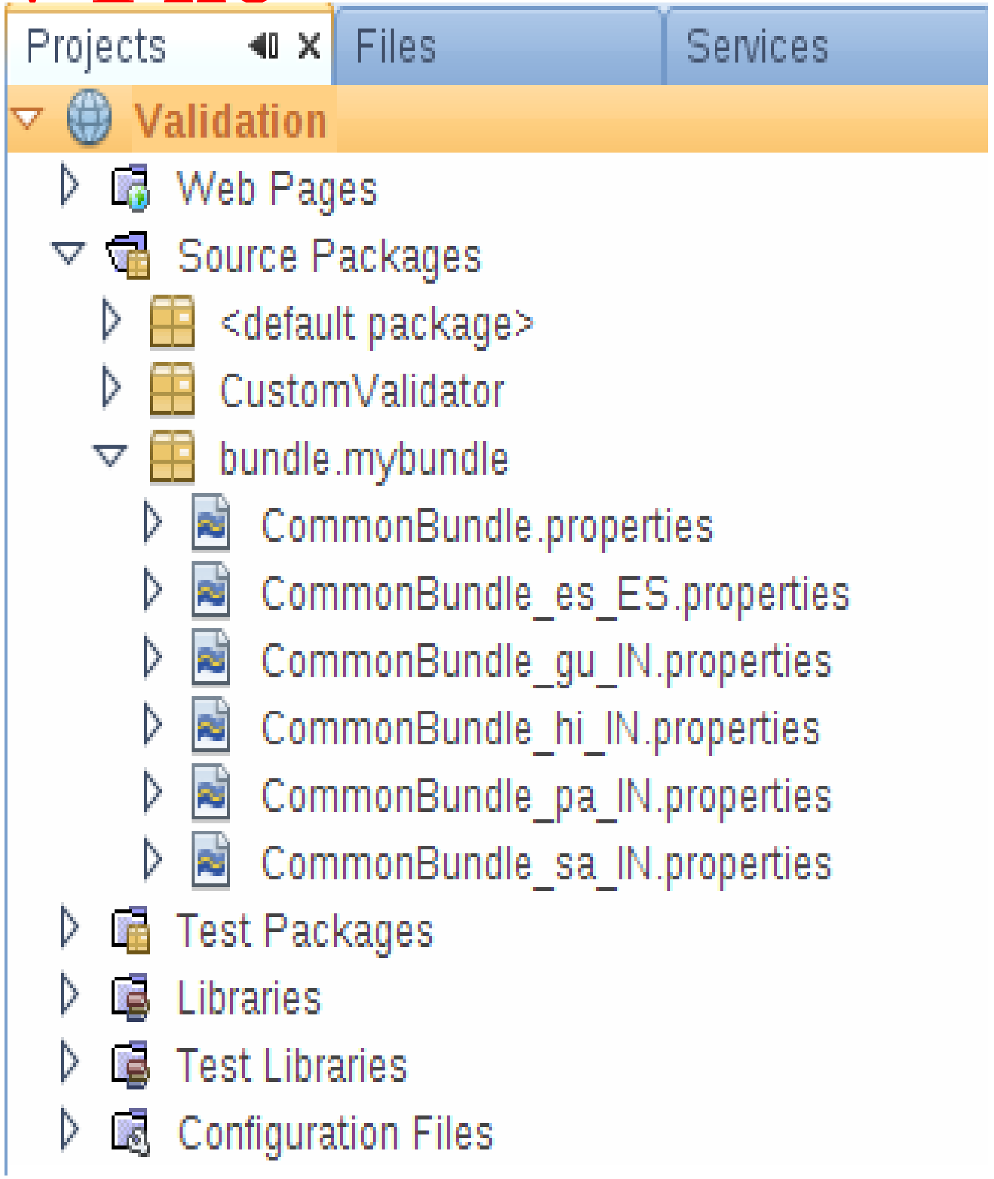
JSF has got all the techniques to support local and international Languages. The global demand of the products and info is forcin companies to adopt internationalised (French, Chinese, Spanish) or localised version (Hindi Punjabi etc) of the app. Two things play a major role in it .

➔The Property file to store the Resource bundle and locale information
➔The JSF tag
➔

The Property File

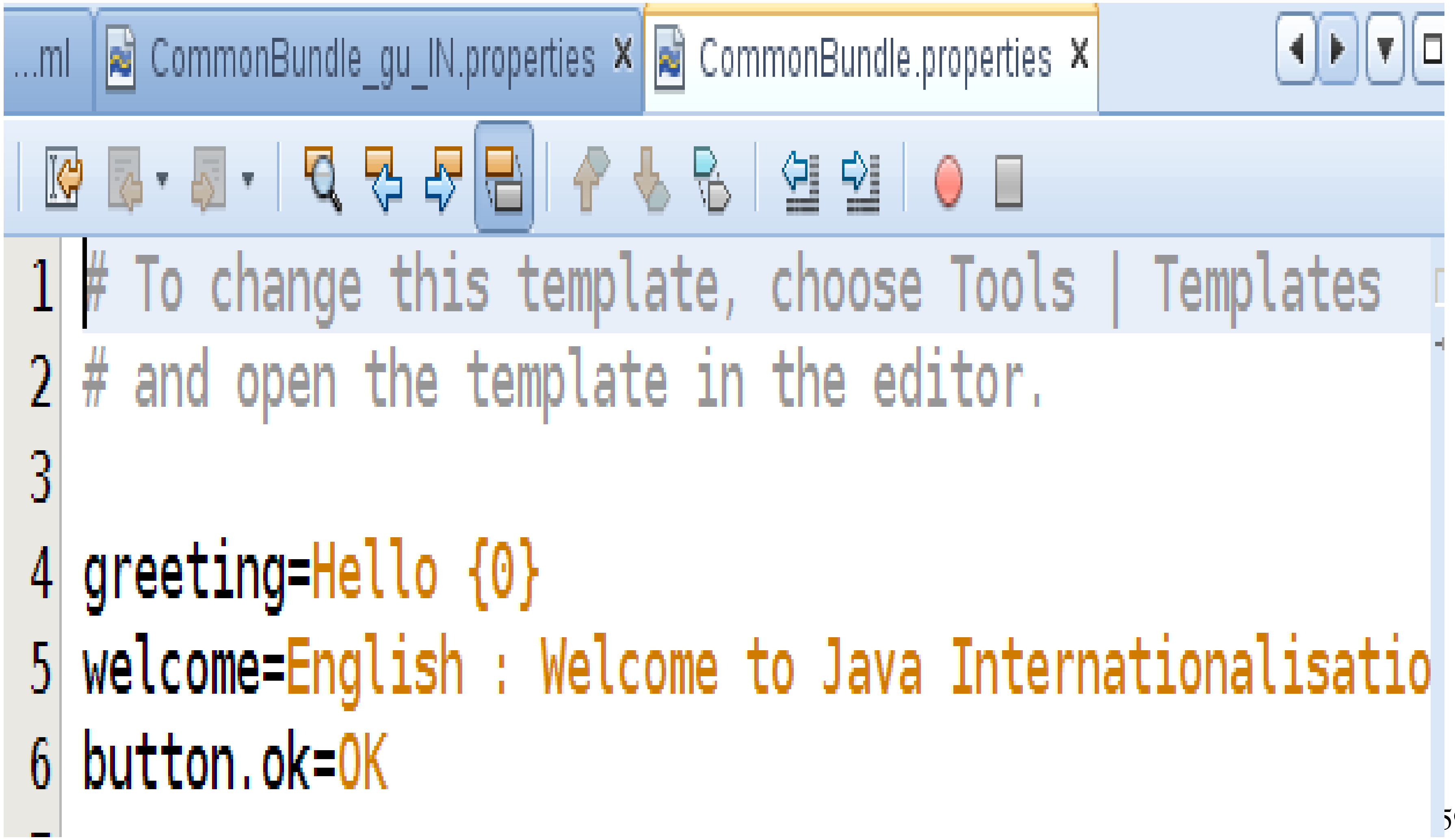
The Property file are just like any other file except they are tagged with the locale and have name value pairs in it.

The property files are just treated like any other java class as they represent the resource bundle. Like the file stored in /bundle/mybundle/common is refered as package bundle.mybundle.common



```
Validation
  Web Pages
  Source Packages
    <default package>
    CustomValidator
    bundle.mybundle
      CommonBundle.properties
      CommonBundle_es_ES.properties
      CommonBundle_gu_IN.properties
      CommonBundle_hi_IN.properties
      CommonBundle_pa_IN.properties
      CommonBundle_sa_IN.properties
  Test Packages
  Libraries
  Test Libraries
  Configuration Files
```

The Property File



```
# To change this template, choose Tools | Templates
# and open the template in the editor.

greeting=Hello {0}
welcome=English : Welcome to Java Internationalisatio
button.ok=OK
```

Entry in faces-config.xml

```
<application>

    <resource-bundle>
      <base-name>bundle.mybundle.CommonBundle
</base-name>
      <var>msgs</var>
      </resource-bundle>
</application
```

The JSF Tag to Show Data

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

Displaying value contained
In the resource bundle

Templating in JSF

Templates are powerful tool to standardize our view of web pages. JSF uses ui:insert and ui:define tags to achieve this. The view of the entire application can be changed in a very short time . It helps the company to be in line with its competitors for look and feel of pages

Templating in JSF

Defining Template as xhtml file

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE --- >
<html xmlns="http://www.w3.org/1999/xhtml" ...>
<h:head> <link href="./resources/Layout.css"  rel="stylesheet"
type="text/css" /> <title>Facelets Template</title></h:head>
<h:body>
        <div id="top" class="top">
            <ui:insert name="top"> </ui:insert></div>
           <div id="left">  <ui:insert name="left"></ui:insert></div>
            <div id="content"> <ui:insert name="content"> </ui:insert>
            </div>
        </div>
    </h:body> </html>
```

Using the Template in JSF

Using Template in any facelets

```
<?xml..<--starting tags -->
<ui:composition template="newTemplate.xhtml">
    <ui:define name="top"> Great Things may not look
good</ui:define>
    <ui:define name="left"> <ul>
                    <li> One </li>
                    </ul></ui:define>
     <ui:define name="content">
        <h:form> <br></br>
            <h:outputText value="Book Name :- "></h:outputText>
<h:inputText value="#{BookMaster.bookName}"> </h:inputText>
 <h:commandButton value=”Submit” action=”#BookMaster.go”/>
</html>
```

Writing Custom Components

- It is now very easy to develop Custom or Composite Components in Post JSF 2.0 . Earlier it was really a time taking laborious process. The UI composition play a great role in this. It involves following steps.
-

1) Writing a <tag-name> facelet for new Components in resources/<tagname-prefix> directory, its interface and rendering rules in implementation

2) Using the new tag in any Facelets by importing its namespace

Interface for New Component

```
<composite:interface>
<composite:actionSource            name="loginButton"
targets="form:loginButton"/>
<composite:attribute name="loginButtonText" default="Log In"
required="true"/>
    <composite:attribute name="loginPrompt"/>
    <composite:attribute name="namePrompt"/>
    <composite:attribute name="passwordPrompt"/>
    <composite:attribute name="loginAction"
     method-signature="java.lang.String action()"/>
    <composite:attribute name="managedBean"/>
  </composite:interface>
```

Implementing New Component

```
 <composite:implementation>
<h:form  id="form" prependId="false ">
           <h1>  #{cc.attrs.loginPrompt} </h1>
          <h:panelGrid columns="2">
              #{cc.attrs.namePrompt}
             <h:inputText id="name" value="#{cc.attrs.managedBean.name}"
            valueChangeListener="#{cc.attrs.managedBean.validateName}"/>
                #{cc.attrs.passwordPrompt}
              <h:inputSecret id="password"
                value="#{cc.attrs.managedBean.password}"/> </h:panelGrid> <p>
<h:commandButton  id="loginButton"    value="#{cc.attrs.loginButtonText}"
action="#{cc.attrs.loginAction}"/> </p></h:form>
 <h:messages layout="table"/> </composite:implementazon>
```

Using the custom components

```
<?xml ..... />
<html> ......
<util:login loginPrompt="#{msgs.loginPrompt}"
              namePrompt="#{msgs.namePrompt}"
          passwordPrompt="#{msgs.passwordPrompt}"
         loginButtonText="#{msgs.loginButtonText}"
             loginAction="#{user.login}"
             managedBean="#{user}">

  </util:login>
</html>
```

Summary

- *We have learnt the following*
- Evolution of Java Web Architecture
- JSTL/EL Apps
- JSF with Facelets
- Managed Beans
- Controllers
- Validators and Converters
- Event Listens
- Localisation and Internationalisation
- Templates- Standardising the view

The Future

It is fast becoming a industry standard. Oracle, now the owner of Oracle Corporation is promoting JSF and thus it is seen as the tough competitor to other Framework and Languages.

The Future

It is fast becoming a industry standard. Oracle, now the owner of Java is promoting JSF and thus it is seen as the tough competitor to other Framework and Languages.