Chapter 11
Java server Faces

11.1 Introduction

JSF is nothing but an abstraction over the existing Web Framework. JSF is implemented as a Servlet which is called the Faces Servlet. Before the advent of JSF, Servlets and JSP are the predominant ones that form the core components in the development of a Web Application. Let us see the traditional interaction that takes place in a Web Application is developed only with the Servlet and the JSP components that follows the MVC-2 Architecture. A client who is normally a HTML Browser sends a request to the server. The Web Server receives the request, encapsulates the request and then populates this request object with the various parameter values from the client and will send it to the Servlet. The Servlet which acts as a Controller, analyses the request, then will interact with the Model (Java Beans) that executes the various application business logic and then chooses which View to be shown to the User.

JavaServer Faces (JSF) technology simplifies building user interfaces for JavaServer applications. Developers can quickly and easily build web applications by assembling reusable UI components in a page, connecting these components to an application data source / model, and wiring client-generated events to server-side event handlers. The user interface code runs on the server, responding to events generated on the client. This allows the application developer to focus on application code.

11.2 Need of MVC

MVC is the Java-BluePrints-recommended architectural design pattern for interactive applications. MVC separates design concerns, thereby decreasing code duplication, centralizing control, and making the application more extensible. MVC also helps developers with different skill sets focus on their core skills and collaborate through clearly defined interfaces. MVC is the architectural design pattern for the presentation tier.

What is MVC Design Pattern?

MVC design pattern designs an application using three separate modules:

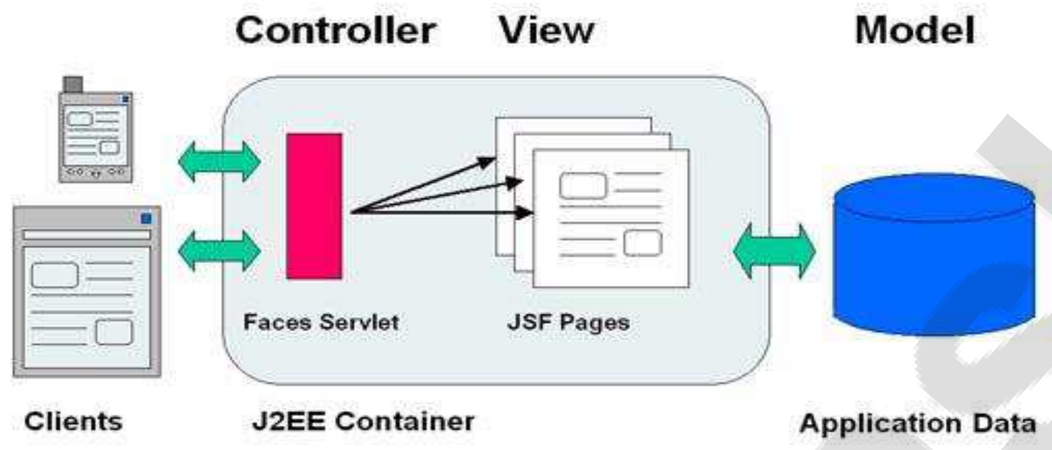
Module	Description
Model	Carries Data and login
View	Shows User Interface
Controller	Handles processing of an application.

Purpose of MVC design pattern is to separate model and presentation to enable developers to set focus on their core skills and collaborate more clearly.

Web Designers have to concentrate only on view layer rather than model and controller layer. Developers can change the code for model and typically need not to change view layer.Controllers are used to process user actions. In this process layer model and views may be changed.

The MVC pattern's purpose is to decouple Model (or data) from the presentation of the data (View). If your application has more than one presentation, you can can replace only the view layer and reuse code for the controller and model. Similarly, if you need to change a model, the view layer remains largely unaffected. Controller handles user actions that might result in changes in the model and updates to the views. When a user requests a JSF page, the request goes to FacesServlet. FacesServlet is the front controller servlet used in JSF. Like many other Web application frameworks, JSF uses the MVC pattern to decouple the view and the model. To handle user requests centrally, the controller servlet makes changes to the model and navigates users to views.

FacesServlet is the controller element in the JSF framework which all user requests go through. FacesServlet examines user requests and calls various actions on the model using managed beans. Backing, or managed, beans are an example of the model. JSF user interface (UI) components represent the view layer. The MVC pattern helps to divide tasks among developers who have different skill sets so tasks can be carried out in parallel; that is, GUI designers can create JSF pages with rich UI components while back-end developers can create managed beans to write business-logic specific code.



11.3 what is JSF?

Java Server Faces which provides a Component-Based Architecture for developing reusable User Interface Components hides most of the complex stuffs that are happening in the View portion of the MVC-2 Architecture. The framework is not only limited to developing customized User Interface Components but also provides support for various Advanced Features like Event handling Mechanism, Validating User Inputs that are sent by the clients, Easy Page Navigation Mechanism etc. The good thing about Java Server Faces is that the degree of coupling between the UI Components that represent the various behaviour/properties and its Rendering is very low. In fact it is almost nill. So, HTML browsers are not the only target client applications. JSF Applications works even well with WML Browsers.

JavaServer Faces (JSF) is a new standard Java framework for building Web applications. It simplifies development by providing a **component-centric** approach to developing Java Web user interfaces. JavaServer Faces also appeals to a diverse audience of Java/Web developers. "Corporate developers" and Web designers will find that JSF development can be as simple as dragging and dropping user interface (UI) components onto a page, while "systems developers" will find that the rich and robust JSF API offers them unsurpassed power and programming flexibility. JSF also ensures that applications are well designed with greater maintainability by integrating the well established Model-View-Controller (MVC) design pattern into it's architecture. Finally, since JSF is a Java standard developed through Java Community Process (JCP), development tools vendors are fully empowered to provide easy to use, visual, and productive develop environments for JavaServer Faces.

JavaServer Faces (JSF) is an application framework for creating Web-based user interfaces. If you are familiar with Swing (the standard Java user interface framework for desktop applications), think of JavaServer Faces as a combination of those frameworks. Like Swing, JSF provides a rich component model complete with event handling and component rendering.

In a nutshell, JSF eases Web-based application development because it:

- Lets you create user interfaces from a set of standard, reusable server-side components
- Provides a set of JSP tags to access those components
- Transparently saves state information and repopulates forms when they redisplay
- Provides a framework for implementing custom components
- Encapsulates event handling and component rendering so you can use standard JSF components or custom components to support markup languages other than HTML
- Lets tool vendors develop IDEs for a standard Web application framework

Currently, JSF is an early access (EA) release, and, as a result, is somewhat immature. The specification leaves some functionality unspecified, and the specification and reference implementation are currently out of sync, with the former specifying new syntaxes and functionality not yet implemented in the latter. On the other hand, JSF is mature enough for you to write code against—although much of that code is guaranteed to be obsolete (see the disclaimer below)—and the reference implementation is fairly complete and relatively bug-free.

11.4 components of JSF

In JSF, a component is a group of interacting classes that together provide a reusable piece of web-based user interface code. A component is made up of three classes that work closely together.

11.4.1 What is Components?

- Components in JSF are elements like text box, button, table etc.. that are used to create UI of JSF Applications. These are objects that manage interaction with a user.
- Simple components like text box, button .
- Compound components like table, data grid
- A component containing many components inside it is called a compound component.

- Components help developers to create UIs by assembling a number of components , associating them with

11.4.2 Components Types

There are two types of Components in JSF

- Standard UI Components
- Custom UI Components

  Standard UI Components

  It contains basic set of UI components like text fields, check boxes , list boxes, panel , label, radio button etc. These are called standard components

  Custom UI Components

  JSF let you create and use your own set of reusable components . These components are called custom components.

11.4.3 JSF–Basic Tags

JSF provides a standard HTML tag library. These tags get rendered into corresponding html output.

For these tags you need to use the following namespaces of URI in html node.

```
<html

xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://java.sun.com/jsf/html" >
```

Following are important *Basic Tags* in JSF:

S.N.	Tag & Description
1	h:inputText Renders a HTML input of type="text", text box.
2	h:inputSecret Renders a HTML input of type="password", text box.
3	h:inputTextarea Renders a HTML textarea field.
4	h:inputHidden Renders a HTML input of type="hidden".

5	h:selectBooleanCheckbox Renders a single HTML check box.
6	h:selectManyCheckbox Renders a group of HTML check boxes.
7	h:selectOneRadio Renders a single HTML radio button.
8	h:selectOneListbox Renders a HTML single list box.
9	h:selectManyListbox Renders a HTML multiple list box.
10	h:selectOneMenu Renders a HTML combo box.
11	h:outputText Renders a HTML text.
12	h:outputFormat Renders a HTML text. It accepts parameters.
13	h:graphicImage Renders an image.
14	h:outputStylesheet Includes a CSS style sheet in HTML output.
15	h:outputScript Includes a script in HTML output.
16	h:commandButton Renders a HTML input of type="submit" button.
17	h:Link Renders a HTML anchor.
18	h:commandLink Renders a HTML anchor.
19	h:outputLink Renders a HTML anchor.
20	h:panelGrid Renders an HTML Table in form of grid.
21	h:message Renders message for a JSF UI Component.
22	h:messages Renders all message for JSF UI Components.
23	f:param Pass parameters to JSF UI Component.
24	f:attribute Pass attribute to a JSF UI Component.
25	f:setPropertyActionListener Sets value of a managed bean's property

11.4.4 JSF – Facelets Tags

JSF provides special tags to create common layout for a web application called facelets tags. These tags gives flexibility to manage common parts of a multiple pages at one place.

For these tags you need to use the following namespaces of URI in html node.

```
<html

xmlns="http://www.w3.org/1999/xhtml"

xmlns:ui="http://java.sun.com/jsf/facelets" >
```

Following are important *Facelets Tags* in JSF :

S.N.	**Tag & Description**

1	Templates We'll demonstrate how to use templates using following tags
	` <ui:insert>`
	` <ui:define>`
	` <ui:include>`
	` <ui:define>`

2	Parameters We'll demonstrate how to pass parameters to a template file using following tag ` <ui:param>`
3	Custom We'll demonstrate how to create custom tags.
4	Remove We'll demonstrate capability to remove JSF code from generated HTML page.

11.4.5 JSF–Convertor Tags

JSF provides inbuilt convertors to convert its UI component's data to object used in a managed bean and vice versa. For example, these tags can convert a text into date object and can validate the format of input as well.

For these tags you need to use the following namespaces of URI in html node.

```
<html

xmlns="http://www.w3.org/1999/xhtml"

xmlns:f="http://java.sun.com/jsf/core" >
```

Following are important *Convertor Tags* in JSF :

S.N.	Tag & Description
1	f:convertNumber Converts a String into a Number of desired format
2	f:convertDateTime Converts a String into a Date of desired format
3	Custom Convertor Creating a custom convertor

11.4.6 JSF–Validator Tags

JSF provides inbuilt validators to validate its UI components. These tags can validates length of field, type of input which can be a custom object.

For these tags you need to use the following namespaces of URI in html node.

```
<html

xmlns="http://www.w3.org/1999/xhtml"

xmlns:f="http://java.sun.com/jsf/core" >
```

Following are important *Validator Tags* in JSF :

S.N.	Tag & Description
1	f:validateLength Validates length of a string
2	f:validateLongRange Validates range of numeric value
3	f:validateDoubleRange Validates range of float value
4	f:validateRegex Validate JSF component with a given regular expression.
5	Custom Validator Creating a custom validator

11.4.7 JSF–Composite Components

JSF provides developer a powerful capability to define own custom components which can be used to render custom contents.

S.N.	tag & Description
1	**composite:interface** Declare configurable values to be used in composite:implementation
2	**composite:attribute** Configuration values are declared using this tag
3	**composite:implementation** Declares JSF component. Can access the configurable values defined in composite:interface using #{cc.attrs.attribute-name} expression.

11.5 JSF as an application

Let us see how to create a simple application using JavaServer Faces or JSF framework in Eclipse IDE. First let us see what are the tools required to create our hello world JSF application.

- JDK 1.5 above
- Tomcat 5.x above or any other container (Glassfish, JBoss, Websphere, Weblogic etc)
- Eclipse 3.2.x above

Following are the list of JAR files required for this application.

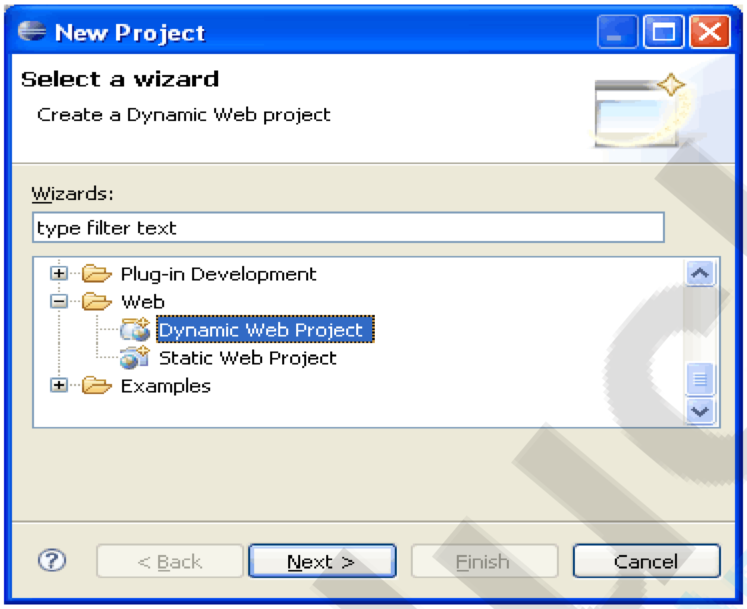
  - jsf-impl.jar
  - jsf-api.jar
  - jstl.jar
  - common-logging.jar
  - common-beanutils.jar
  - common-collections.jar
  - common-chain.jar
  - common-digester.jar

We will implement a JSF application with an Add User screen with two fields, ID and User Name. Once user enter these values and press submit, user will be redirected to a welcome page displaying the user name.

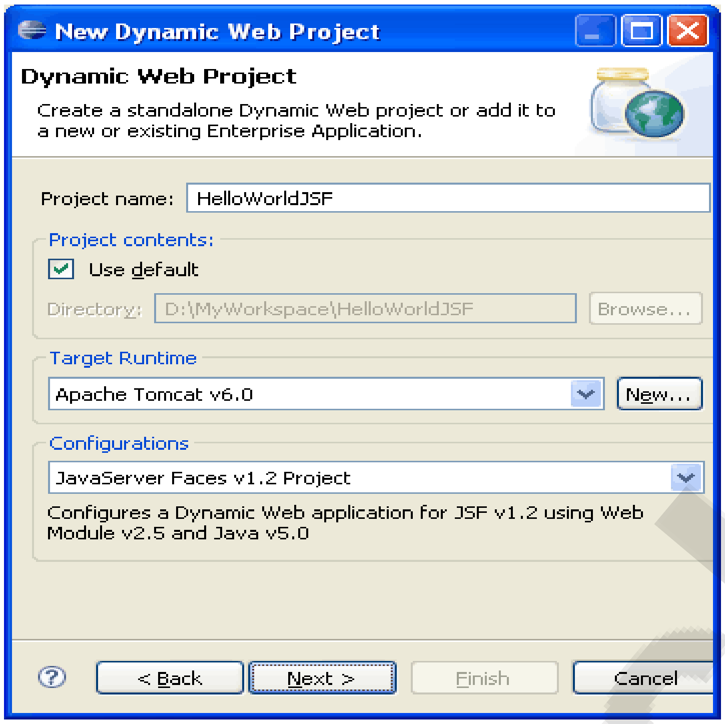
Let us start with our first JSF based web application.

Step 1: Create Dynamic Web project

Open Eclipse and goto File -> New -> Project and select **Dynamic Web Project** in the New Project wizard screen.



Select Dynamic Web application and click **Next**.



Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: HelloWorldJSF

Project contents:

✓ Use default

Directory: D:\MyWorkspace\HelloWorldJSF

Target Runtime

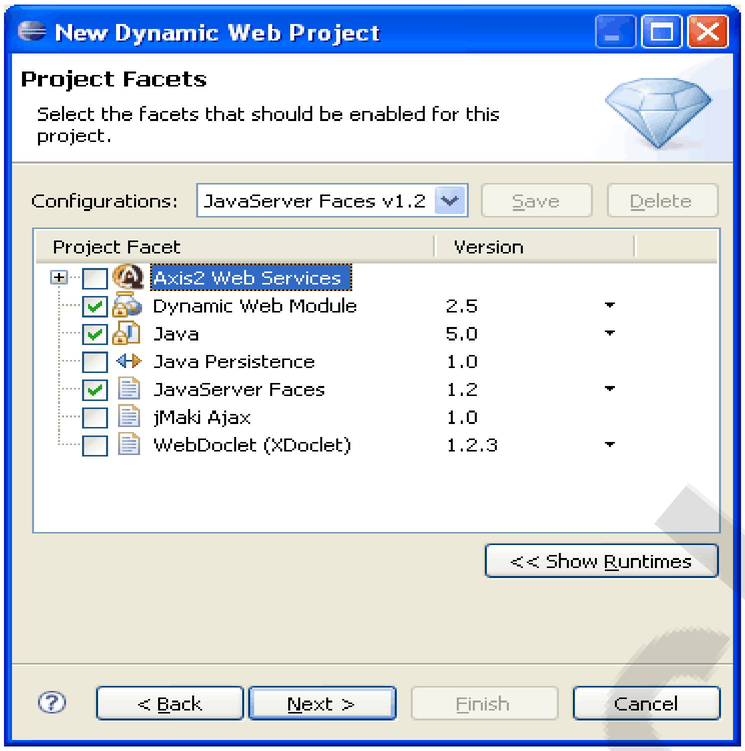
Apache Tomcat v6.0

Configurations

JavaServer Faces v1.2 Project

Configures a Dynamic Web application for JSF v1.2 using Web Module v2.5 and Java v5.0

Write the name of the project **HelloWorldJSF**. Once this is done, select the target runtime environment (e.g. Apache Tomcat v6.0). This is to run the project inside Eclipse environment. In configuration select **JavaServer Faces v1.2 Project** and press **Next**.


New Dynamic Web Project

Project Facets

Select the facets that should be enabled for this project.

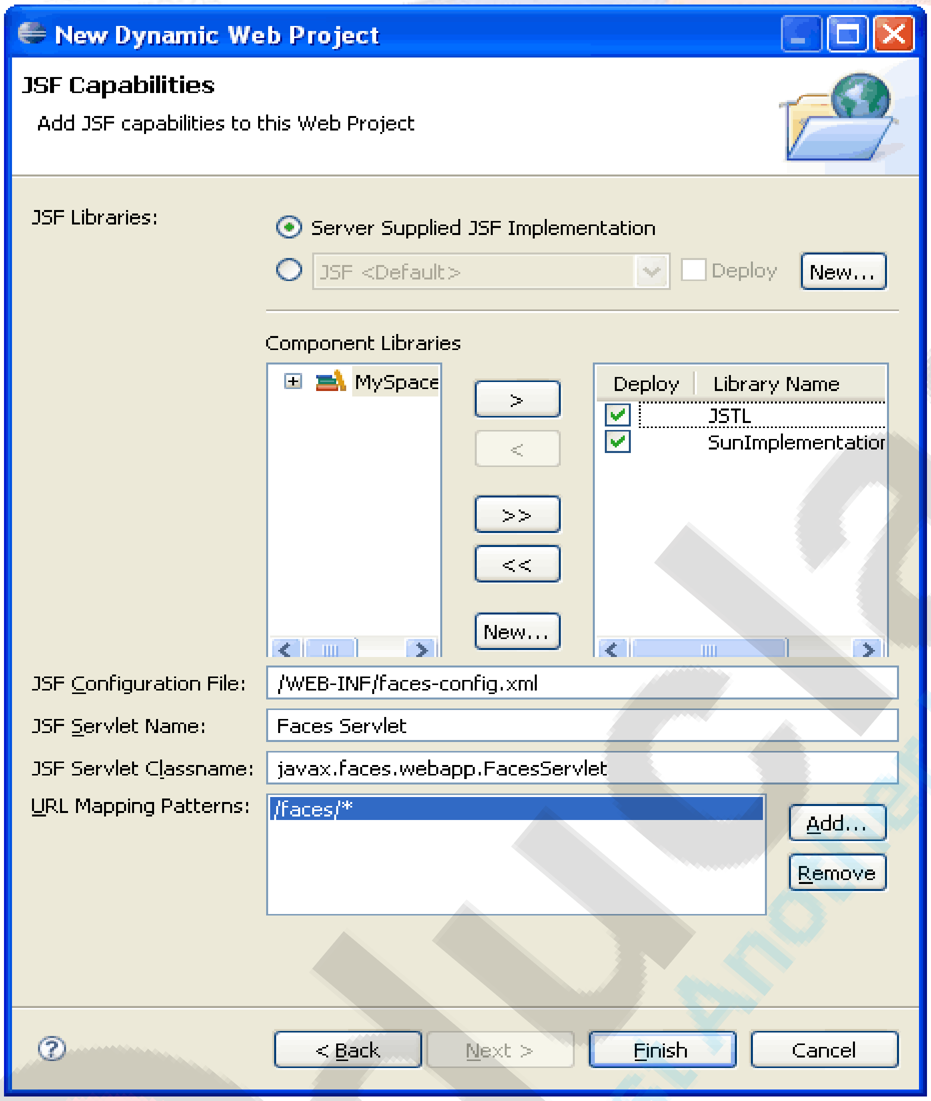
Configurations: JavaServer Faces v1.2

Project Facet	Version
☐ Axis2 Web Services	
✓ Dynamic Web Module	2.5
✓ Java	5.0
☐ Java Persistence	1.0
✓ JavaServer Faces	1.2
☐ jMaki Ajax	1.0
☐ WebDoclet (XDoclet)	1.2.3

On Project Facets window, select Java 5 and JSF 1.2 and press Next.

Skip Web module window and press Next.

Skip Web module window and press Next.


New Dynamic Web Project

JSF Capabilities

Add JSF capabilities to this Web Project

JSF Libraries: [✓ Server Supplied JSF Implementation] JSF <Default> Deploy New...

Component Libraries

MySpace

Deploy	Library Name
✓	JSTL
✓	SunImplementation

JSF Configuration File: /WEB-INF/faces-config.xml

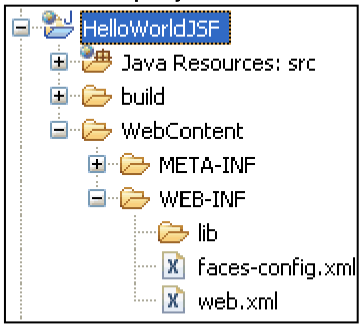
JSF Servlet Name: Faces Servlet

JSF Servlet Classname: javax.faces.webapp.FacesServlet

URL Mapping Patterns: /faces/*

Select JSF component library. Click New in Component Libraries and add `jstl.jar`, `jsf-api.jar` and `jsf-impl.jar`. In URL Mapping Patterns add /**faces**/* and then click **Finish**.

Once the project is created, you can see its structure in Project Explorer.



Step 2: Create Package and Managed bean

Create a package `net.test.jsf.helloworld` in the source folder and create a Java file `UserBean.java.` Copy following content into `UserBean.java.`

```
package net.test.jsf.helloworld;

publicclassUserBean {

    privateintid;

    privateString name;

    //Action method to add user

    publicString addUser() {

        return"success";

    }

    publicintgetId() {

        returnid;

    }

    publicvoidsetId(intid) {

        this.id = id;

    }
    publicString getName() {

        returnname;

    }

    publicvoidsetName(String name) {

        this.name = name;

    }

}
```

Above Java class is a User bean that we will use to store our user's information. This class acts like a form bean and action class. The `addUser()` method will get called when we click Add button on our Add User page.

Step 3: Create JSP files

Create two JSP files: `AddUser.jsp` and `ListUser.jsp` in WebContent folder. Copy following content in each of these files.

AddUser.jsp

```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>

<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>

<f:view>

        <p>

            <h:messageid="errors"for="User_ID"style="color:red">

        </h:message></p>

    <h:form>

        <h:panelgridborder="1"columns="2">

            <h:outputtextvalue="ID"></h:outputtext>


   <h:inputtextid="User_ID"value="#{userBean.id}"required="true">
```

```
                    <f:validatelongrangeminimum="1"maximum="500">

                </f:validatelongrange></h:inputtext>

                <h:outputtextvalue="Name"></h:outputtext>

                <h:inputtextvalue="#{userBean.name}"></h:inputtext>


<h:commandbuttonaction="#{userBean.addUser}"value="Add
   Customer"></h:commandbutton>

        </h:panelgrid>

    </h:form>

</f:view>
```

We have added a validation rule for ID using `f:validateLongRange` tag and `required="true"` attribute. The ID must be in between 1 and 500.

ListUser.jsp

```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>

<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>

<f:view>

    <h:form>

        <h:outputtextvalue="User #{userBean.name} is added
successfully.">

        </h:outputtext>

    </h:form>

</f:view>
```

Step 4: Modify faces-config.xml file

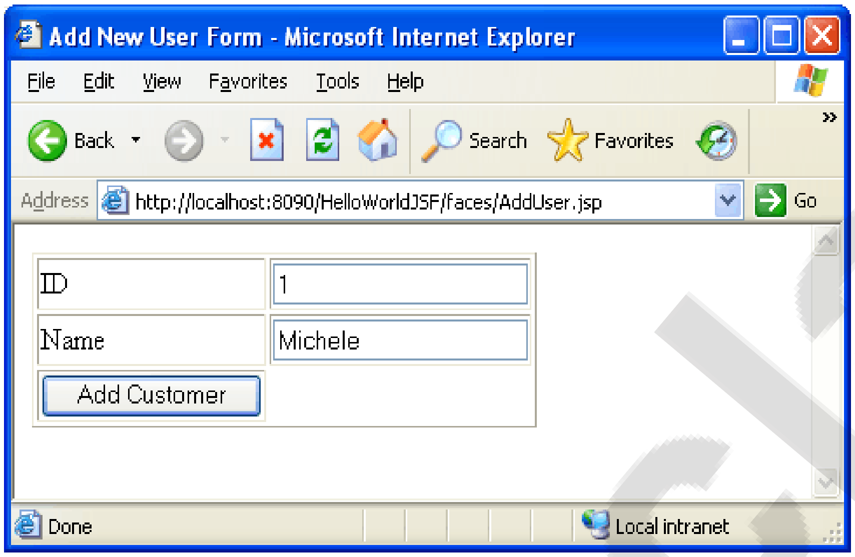
Open `faces-config.xml` from WebContent -> WEB-INF folder and copy following content into it.

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

<faces-configxmlns="http://java.sun.com/xml/ns/javaee"xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance"xsi:schemalocation="htt
p://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd"version
="1.2">

    <managed-bean>

        <managed-bean-name>userBean</managed-bean-name>

        <managed-bean-class>

            net.test.jsf.helloworld.UserBean

        </managed-bean-class>

        <managed-bean-scope>session</managed-bean-scope>

    </managed-bean>

    <navigation-rule>

        <display-name>AddUser</display-name>

        <from-view-id>/AddUser.jsp</from-view-id>

        <navigation-case>

            <from-outcome>success</from-outcome>

            <to-view-id>/ListUser.jsp</to-view-id>

        </navigation-case>

    </navigation-rule>

</faces-config>
```

In faces config we have defined a managed bean UserBean with scope session and mapping from `AddUser.jsp` to `ListUser.jsp`.

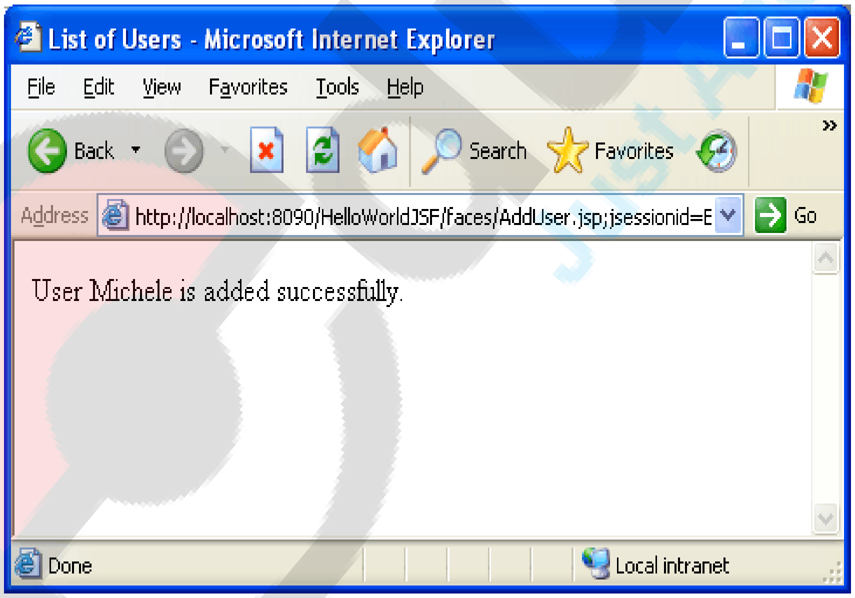
Step 5: Execute and run the project

Final step is to execute the project and view it in browser.
For this, right click on Project Name in Project Explorer -> Run As -> Run on Server (Shortcut Alt+Shift+X, R).


Add New User Form - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8090/HelloWorldJSF/faces/AddUser.jsp
Go
Done
Local intranet

ID	1
Name	Michele
Add Customer	

Once you enter ID and Username and press Add User, following success screen will appear.


List of Users - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8090/HelloWorldJSF/faces/AddUser.jsp;jsessionid=E
Go
Done
Local intranet

User Michele is added successfully.

11.6 JSF lifecycle

Regardless of whether you are using JSF with JSP pages, servlets, or some other web technology, each request/response flow that involves JSF follows a certain life cycle. Several kinds of request/response cycles can occur in a JSF-enabled application. You can have a request that comes from a previously rendered JSF page (a JSF request) and a request that comes from a non-JSF page (a non-JSF request). Likewise, you can have a JSF response or a non-JSF response. We are concerned with these three request/response pairs:

• Non-JSF request generates JSF response
• JSF request generates JSF response
• JSF request generates non-JSF response

Of course, you can also have a non-JSF request that generates a non-JSF response. Because this does not involve JSF in any way, the JSF life cycle does not apply.

When using JSF, the life cycle is more complicated. This is because the core of JSF is the MVC pattern, which has several implications. User actions in JSF-generated views take place in a client that does not have a permanent connection to the server. The delivery of user actions or page events is delayed until a new connection is established. The JSF life cycle must handle this delay between event and event processing. Also, the JSF life cycle must ensure that the view is correct before rendering the view. To ensure that the business state is never invalid, the JSF system includes a phase for validating inputs and another for updating the model only after all inputs pass validation.

The JSF life cycle has six phases as defined by the JSF specification:

• **Restore View:**In this phase, the JSF implementation restores the objects and data structures that represent the view of the request. Of course, if this is the client's first visit to a page, the JSF implementation must create the view. When a JSF implementation creates and renders a JSF-enabled page, it creates UI objects for each view component. The components are stored in a component tree, and the state of the UI view is saved for subsequent requests. If this is a subsequent request, the previously saved UI view is retrieved for the processing of the current request.

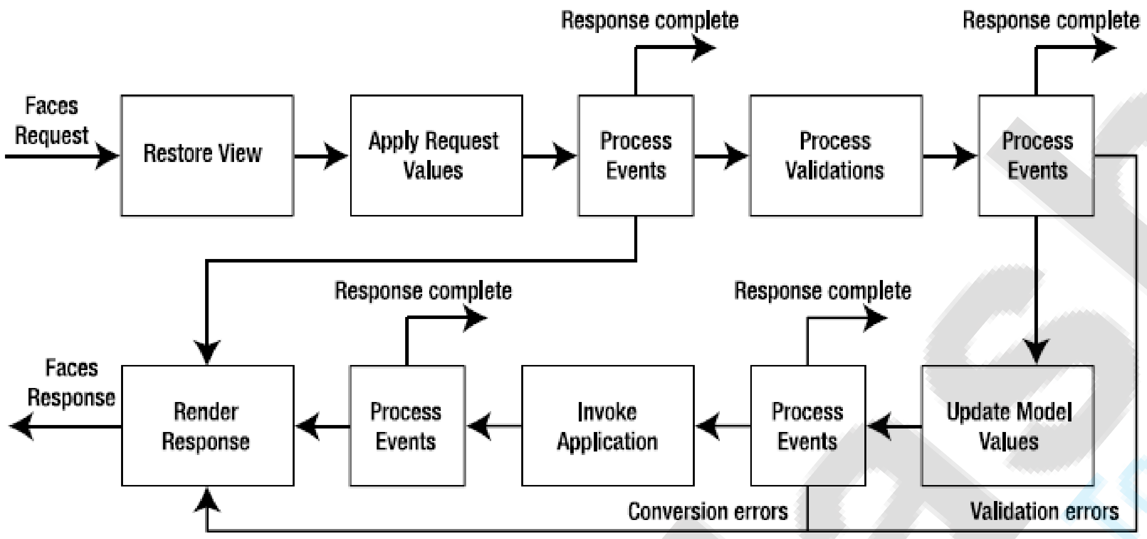
• **Apply Request Values:**Any data that was sent as part of the request is passed to the appropriate UI objects that compose the view. Those objects update their state with the data values. Data can come from input fields in a web form, from cookies sent as part of the request, or from request headers. Data for some components, such as components that create HTML input fields, is validated at this time. Note that this does not yet update the business objects that compose the model. It updates only the UI components with the new data.

• **Process Validations:**The data that was submitted with the form is validated (if it was not validated in the previous phase). As with the previous phase, this does not yet update the business objects in the application. This is because if the JSF implementation began to update the business objects as data was validated, and a piece of data failed validation, the model would be partially updated and in an invalid state.

• **Update Model Values:**After all validations are complete, the business objects that make up the application are updated with the validated data from the request. In addition, if any of the data needs to be converted to a different format to update the model (for example, converting a String to a Date object), the conversion occurs in this phase. Conversion is needed when the data type of a property is not a String or a Java primitive.

• **Invoke Application:**During this phase, the action method of any command button or link that was activated is called. In addition, any events that were generated during previous phases and that have not yet been handled are passed to the web application so that it can complete any other processing of the request that is required.

• **Render Response:**The response UI components are rendered, and the response is sent to the client. The state of the UI components is saved so that the component tree can be restored when the client sends another request. For a JSF-enabled application, the thread of execution for a request/response cycle can flow through each phase, in the order listed here and as shown in Figure below. However, depending on the request, and what happens during the processing and response, not every request will flow through all six phases.



In Figure, you can see a number of optional paths through the life cycle. For example, if errors occur during any of the phases, the flow of execution transfers immediately to the Render Response phase, skipping any remaining phases. One way this might occur is if input data is incorrect or invalid. If data fails validation in either the Apply Request Values or Process Validations phase, information about the error is saved and processing proceeds directly to the Render Response phase. Also, if at any point in the life cycle the request processing is complete and a non-JSF response is to be sent to the client, the flow of execution can exit the life cycle without completing further phases.

11.7 JSF configuration

JSF configuration files

JSF is based on the following configuration files:

web.xml - General web application configuration file

faces-config.xml - Contains the configuration of the JSF application.

11.7.1 web.xml

JSF requires the central configuration list web.xml in the directory WEB-INF of the application. This is similar to other web-applications which are based on servlets.

You must specify in `web.xml` that a "FacesServlet" is responsible for handling JSF applications. "FacesServlet" is the central controller for the JSF application. "FacesServlet" receives all requests for the JSF application and initializes the JSF components before the JSP is displayed.

You can get `web.xml` file from WEB-INF folder of any other application available in TOMCAT by default or you can create yourself with the same name and extention of the file i.e. "`web.xml`". If you are creating this file then take care of mentioning version of xml. For ex. `<?xml version="1.0"?>` at the top of file and after that all elements will be written within `web-app` opening and closing tag i.e.`<web-app>`and `</web-app>`. So the root element of this file is `<web-app>`.

So the initial format of this file will be like this:

```
<?xml version="1.0"?>
 <web-app>
 .................................
 .................................
 .................................
 </web-app>
```

Define a “`javax.faces.webapp.FacesServlet`” mapping, and map to those well-known JSF file extensions (`/faces/*, *.jsf, *.xhtml,*.faces`).

11.7.2 faces-config.xml

Now we come to the second file `faces-config.xml` that will be in the same place where `web.xml` is i.e. WEB-INF folder. Here also you have to take care of mentioning version of xml as we did in `web.xml` file. All tag elements will be within **faces-config** opening and closing tag i.e. `<faces-config>`and `</faces-config>`.So the root element of this file is `<faces-config>` tag.

"`faces-config.xml`" allows to configure the application, managed beans, convertors, validators, and navigation.

So initial format of this file will be like this:

```
<?xml version="1.0"?>
 <faces-config>
 .................................
 .................................
 .................................
 </faces-config>
```

11.8 JSF web applications (login form, JSF pages)

This application will make you understand how `Model (M), View (V), Controller (C)` architecture is implemented in JavaServer Faces. This application will make use of UI components, Validator, Navigation and Bean component available with JSF.

This application will take a user First Name and Last Name. Later these fields will be validated by JSF and using the controller bean and Navigation rule the output will be displayed. This application will also introduce a UI component which is a submit button.

To run this tutorial, as a minimum you will be required to have installed the following prerequisite software:

- Sun JDK 6.0+ (J2SE 1.6)
- Eclipse IDE for Java EE Developers, which is platform specific
- Apache Geronimo Eclipse Plugin 2.1.x
- Apache Geronimo Server 2.1.x

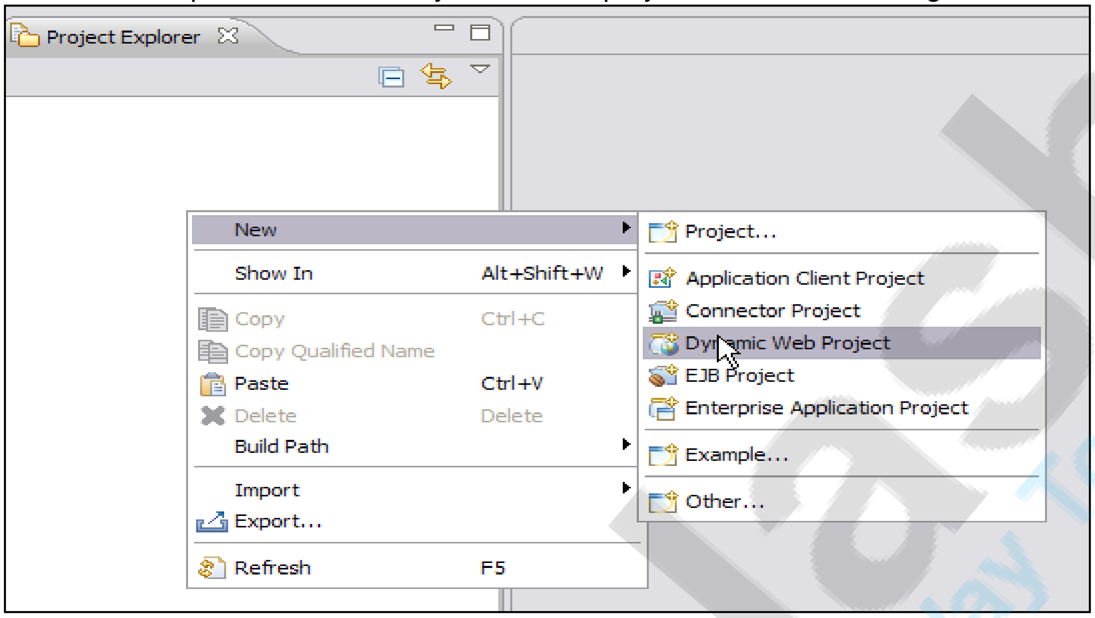
Icon

Geronimo version 2.1.x, Java 1.5 runtime, and Eclipse Ganymede are used is used in this tutorial but other versions can be used instead (e.g., Geronimo version 2.2, Java 1.6, Eclipse Europa)

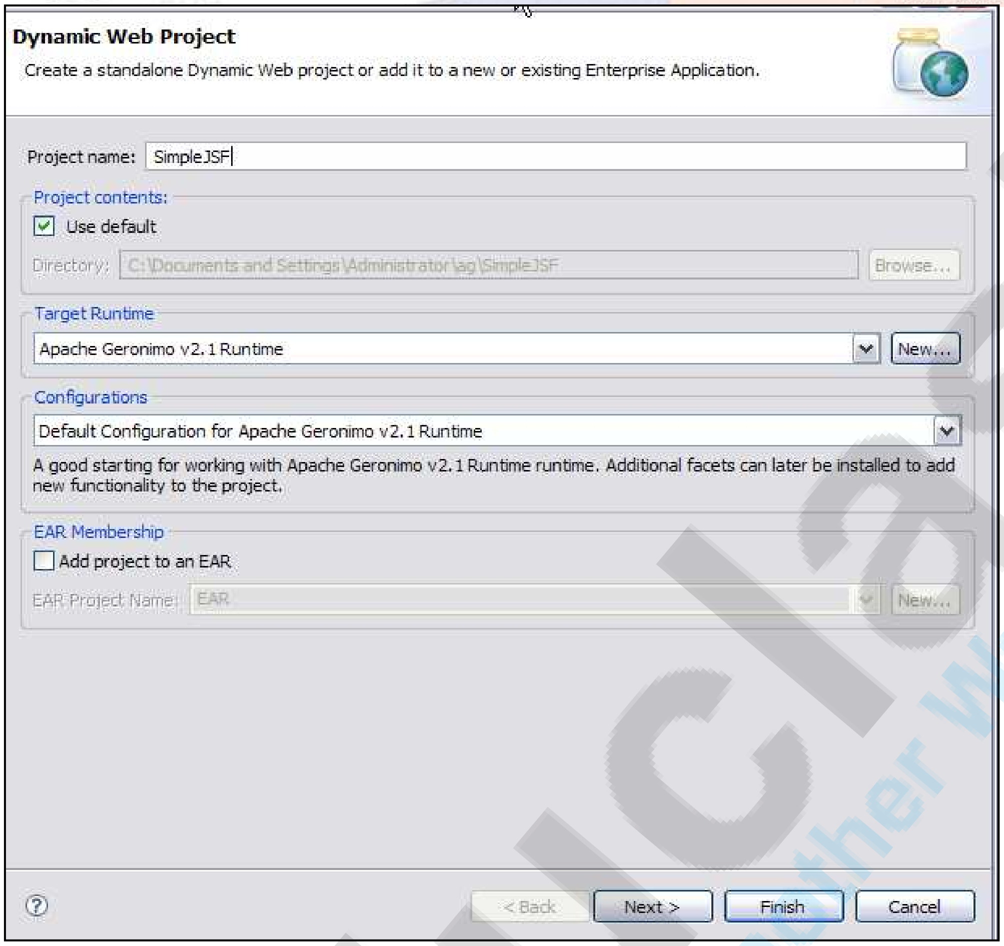
Once you have all the prerequisites installed, follow the following steps to create a project with Eclipse.

11.8.1 Setting Eclipse for application development

1. Launch Eclipse and create a dynamic Web project as shown in the figure.



2. Give the fields for the Web Project as shown in the following figure.


Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: SimpleJSF

Project contents:

[✓] Use default

Directory: C:\Documents and Settings\Administrator\ag\SimpleJSF

Target Runtime: Apache Geronimo v2.1 Runtime

Configurations: Default Configuration for Apache Geronimo v2.1 Runtime

A good starting for working with Apache Geronimo v2.1 Runtime runtime. Additional facets can later be installed to add new functionality to the project.

EAR Membership

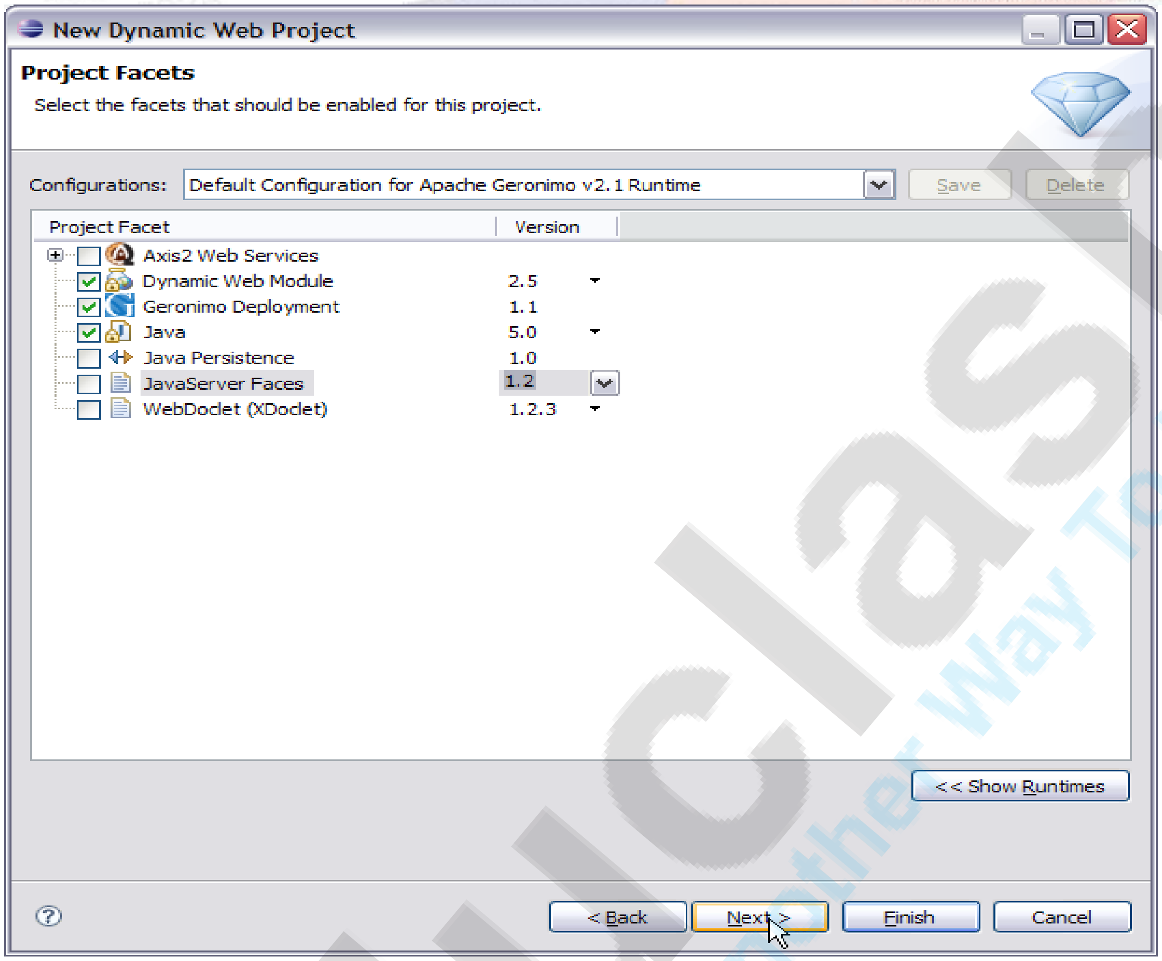
[ ] Add project to an EAR

EAR Project Name: EAR

3.Select **Finish**.

4.Right click on the **SimpleJSF** project and select **Properties**, then select **Project Facets**.

5.Check the box for **JavaServerFaces** and under the *Version* tab select **1.2** as the version. Select the **Further configuration required...** indicator to display the *JSF Capabilities* pane.


New Dynamic Web Project

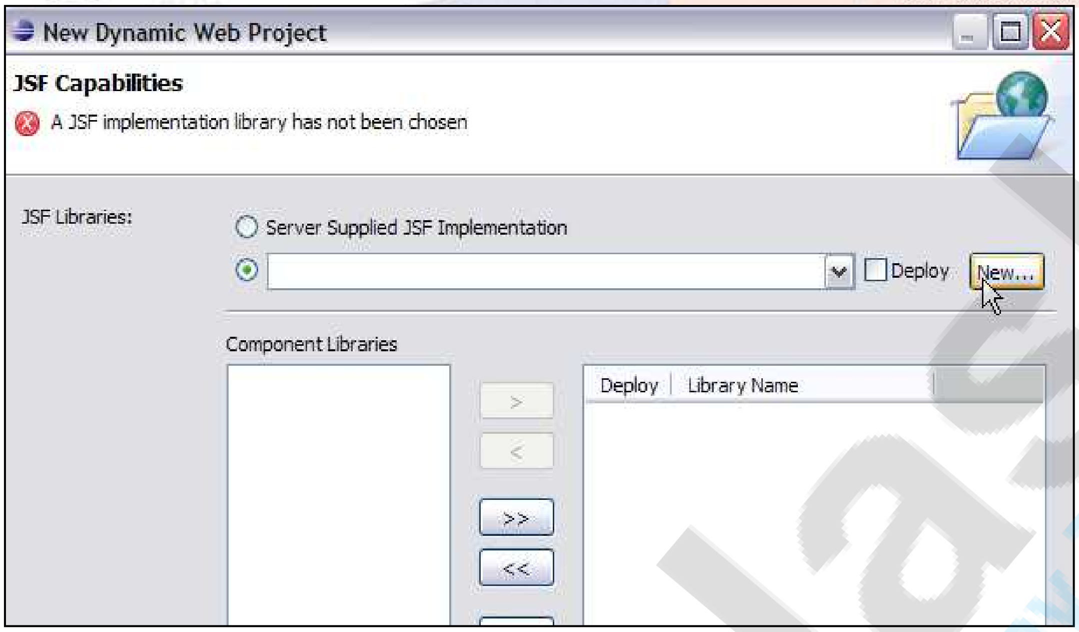
Project Facets

Select the facets that should be enabled for this project.

Configurations: Default Configuration for Apache Geronimo v2.1 Runtime

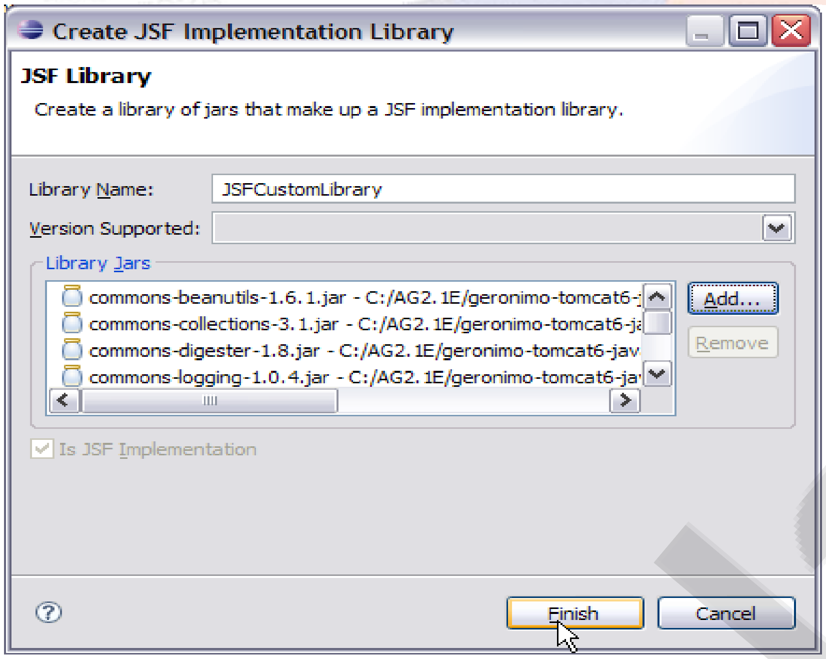
Project Facet	Version
☐ Axis2 Web Services	
✓ Dynamic Web Module	2.5
✓ Geronimo Deployment	1.1
✓ Java	5.0
☐ Java Persistence	1.0
☐ JavaServer Faces	1.2
☐ WebDoclet (XDoclet)	1.2.3

6. On the *JSF Capabilities* window check the box and select **new** as shown in the figure.



7. The next window is used to create a JSF Implementation library. Give the library name as **JSFCustomLibrary** and add the following jars. Select **Finish** once done. See the figure below:

- `<GERONIMO_HOME>\repository\commons-beanutils\commons-beanutils\1.7.0\commons-beanutils-1.7.0.jar`
- `<GERONIMO_HOME>\repository\commons-collections\commons-collections\3.2\commons-collections-3.2.jar`
- `<GERONIMO_HOME>\repository\commons-digester\commons-digester\1.8\commons-digester-1.8.jar`
- `<GERONIMO_HOME>\repository\commons-logging\commons-logging\1.0.4\commons-logging-1.0.4.jar`
- `<GERONIMO_HOME>\repository\org\apache\myfaces\core\myfaces-api\1.2.3\myfaces-api-1.2.3.jar`
- `<GERONIMO_HOME>\repository\org\apache\myfaces\core\myfaces-impl\1.2.3\myfaces-impl-1.2.3.jar`


Create JSF Implementation Library

JSF Library

Create a library of jars that make up a JSF implementation library.

Library Name: JSFCustomLibrary

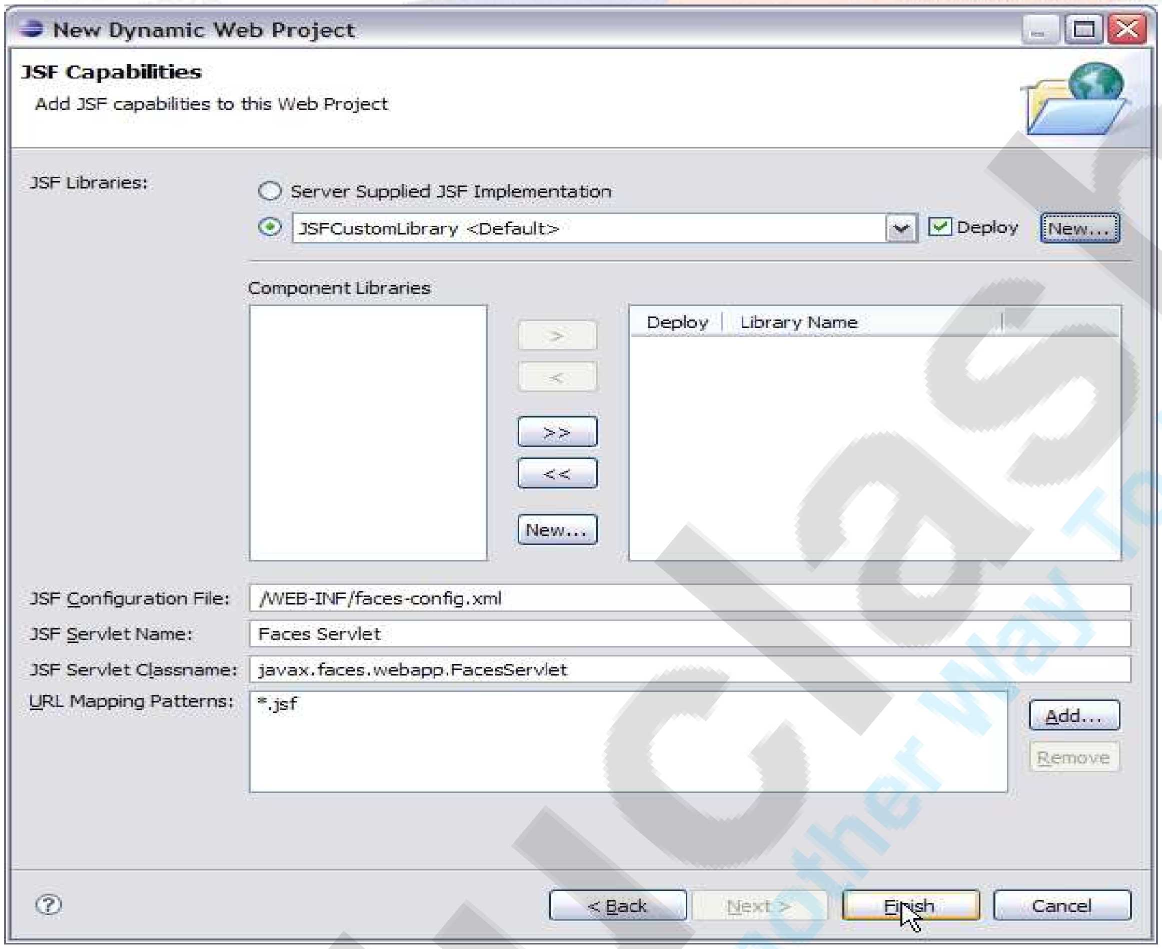
Version Supported:

Library Jars

commons-beanutils-1.6.1.jar - C:/AG2.1E/geronimo-tomcat6-j
commons-collections-3.1.jar - C:/AG2.1E/geronimo-tomcat6-ja
commons-digester-1.8.jar - C:/AG2.1E/geronimo-tomcat6-jav
commons-logging-1.0.4.jar - C:/AG2.1E/geronimo-tomcat6-ja

Is JSF Implementation

8. Check Deploy and modify the URL pattern to ***.jsf** as shown in the figure. Select **Finish**.



New Dynamic Web Project

JSF Capabilities

Add JSF capabilities to this Web Project

JSF Libraries: ( ) Server Supplied JSF Implementation [● JSFCustomLibrary <Default>] [✓ Deploy]

Component Libraries

JSF Configuration File: /WEB-INF/faces-config.xml

JSF Servlet Name: Faces Servlet

JSF Servlet Classname: javax.faces.webapp.FacesServlet

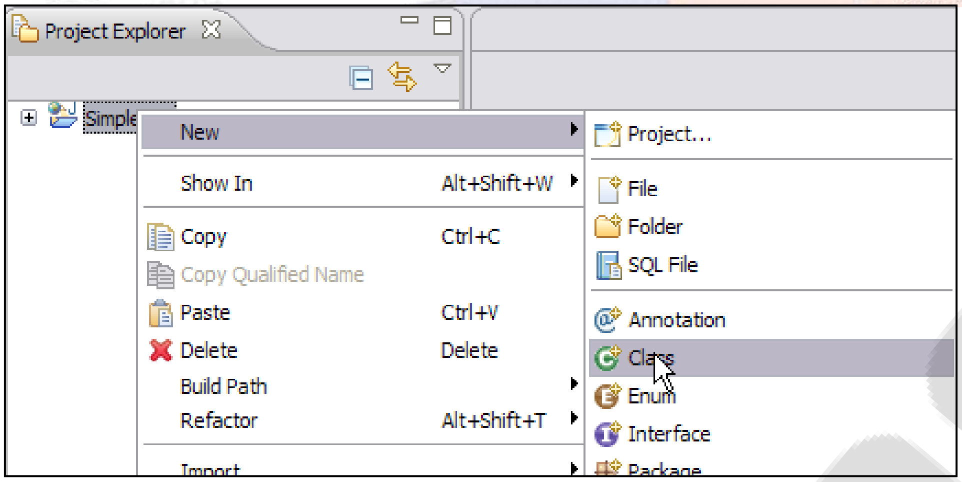
URL Mapping Patterns: *.jsf

This finishes the setting up of the Eclipse IDE for application development.

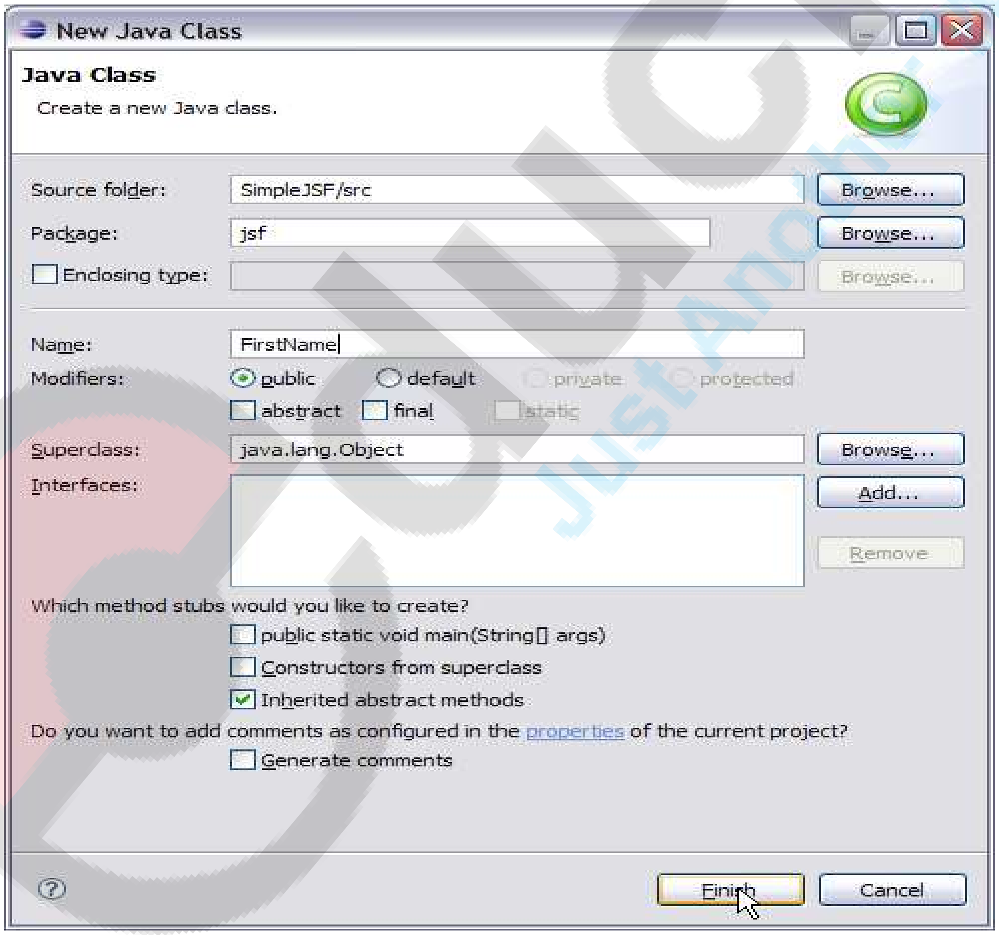
11.8.2 Define and Implement the application Model (M)

The **Model** as suggested by MVC architecture handles data and logic of the application. In an enterprise application. Java Beans are used to represent collection of data and operation on that data. In JSF we use Java Beans to define the Model.

1. Under the project explorer right click on the **SimpleJSF** project and create a new class.



2. Fill the *New Java Class* form with *jsf* as the package name and *FirstName* as the bean class name. Select **Finish** once done.


New Java Class

Java Class

Create a new Java class.

Source folder: SimpleJSF/src

Package: jsf

Enclosing type: [ ]

Name: FirstName

Modifiers: [✓ public] default private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[ ] public static void main(String[] args)

[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?

[ ] Generate comments

3. Add the following code to the **FirstName** bean class:

FirstName.java

```
packagejsf;

publicclassFirstName {

    String username;

    publicString getName() {

        returnusername;

    }

    publicvoidsetName(String name) {

        username = name;}

   }
```

4. Create a second Bean class **LastName** and add the following code to the class:

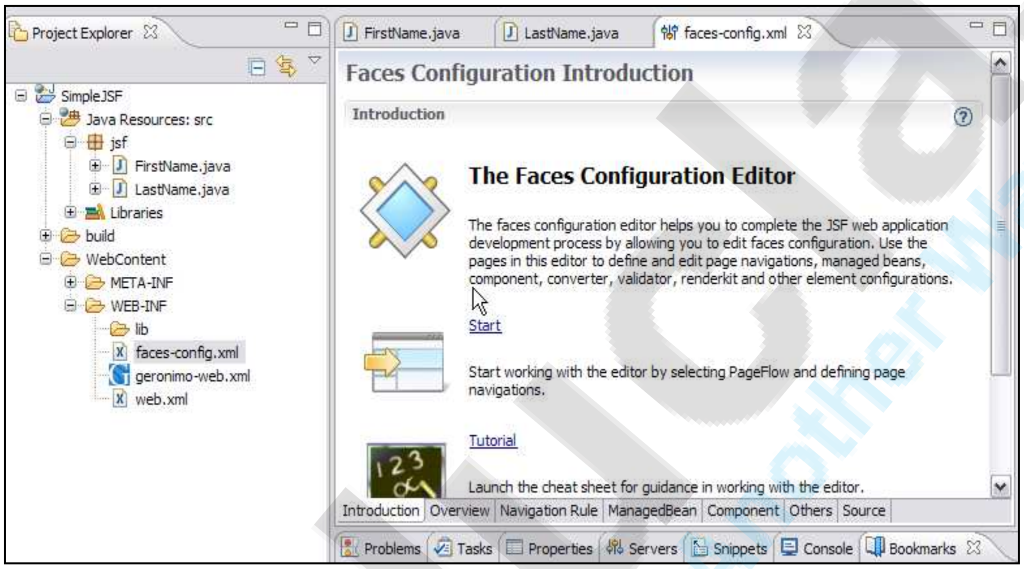
LastName.java

```
packagejsf;

publicclassLastName {

    String lastname;

    publicString getLName()    {

        returnlastname;

    }

    publicvoidsetLName(String lname)

    {

        lastname = lname;

}
}
```

This completes the Model definition and implementation of the bean classes.

11.8.3 Define and implement Model (M) objects to Controller

1. In a JSF application the **Controller** is implemented by a configuration file called `WebContent/WEB-INF/faces-config.xml`. Double click on the file. This will open the *Faces Configuration Editor*.



The Faces Configuration Editor

The faces configuration editor helps you to complete the JSF web application development process by allowing you to edit faces configuration. Use the pages in this editor to define and edit page navigations, managed beans, component, converter, validator, renderkit and other element configurations.

Start

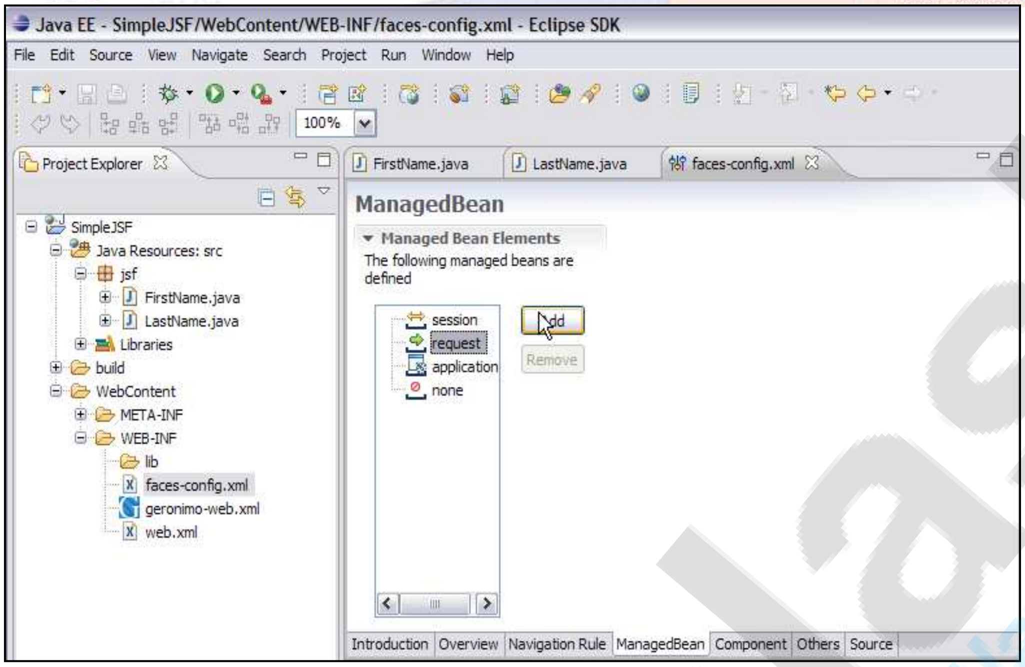
Start working with the editor by selecting PageFlow and defining page navigations.

Tutorial

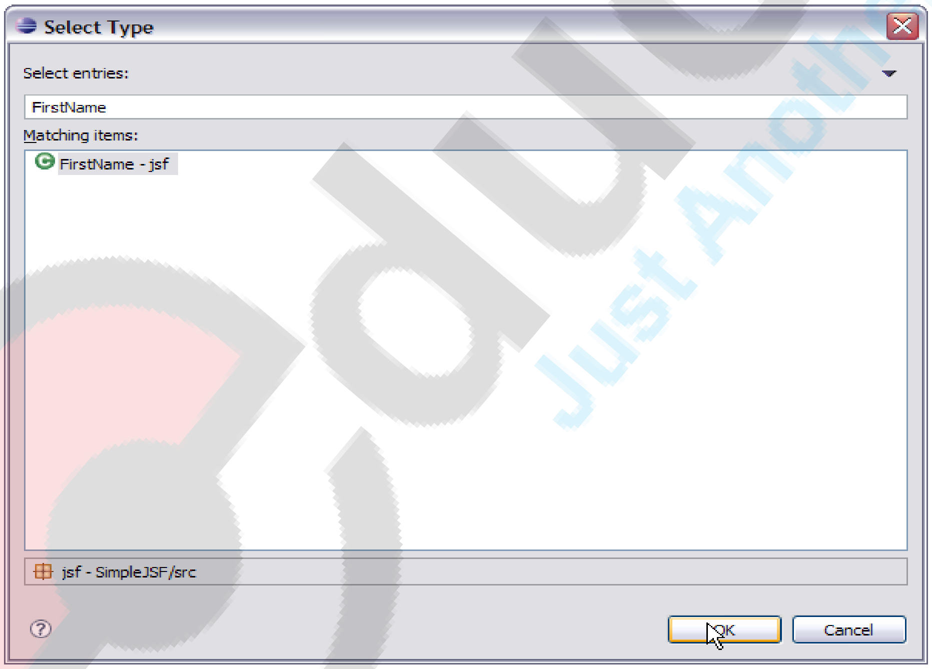
Launch the cheat sheet for guidance in working with the editor.

Introduction | Overview | Navigation Rule | ManagedBean | Component | Others | Source
Problems | Tasks | Properties | Servers | Snippets | Console | Bookmarks

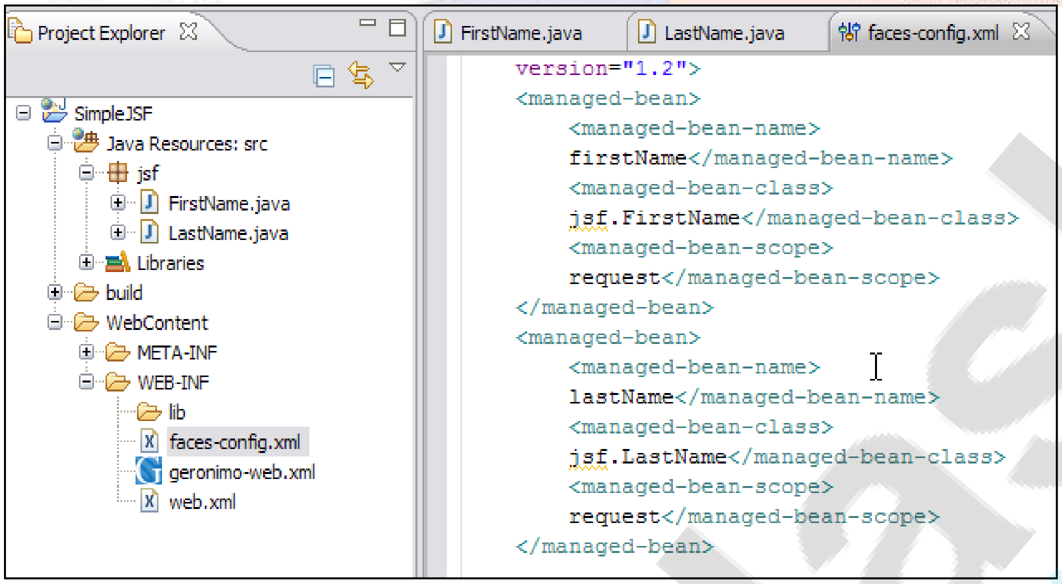
2. Select the *ManagedBean* tab in the editor. Select the **request** Managed Bean Element and select **Add**.



3. Choose the **Using an existing Java class** option, select **Browse**. Give the search element as *FirstName* and select **OK**.



4. Select **Finish** on the next window. Similarly add the other bean **LastName**. Now select the *Source* tab in the Faces configuration Editor. It displays the bean components (i.e., the Model) in the controller.

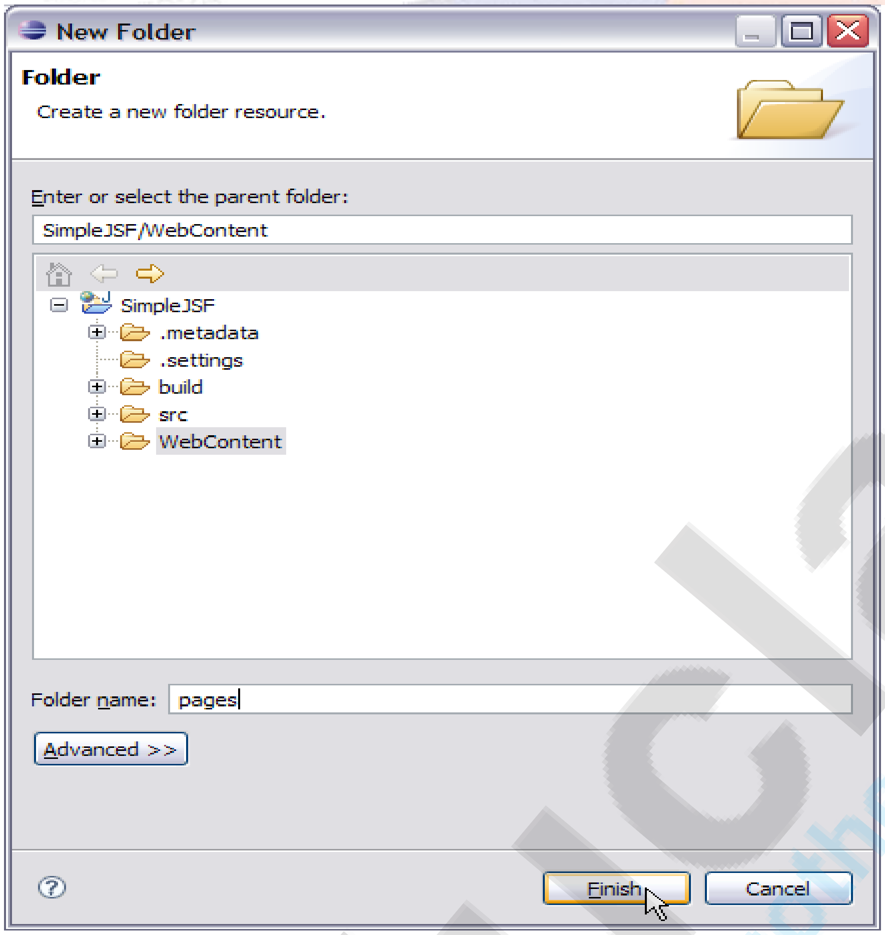


```
    version="1.2">
    <managed-bean>
        <managed-bean-name>
        firstName</managed-bean-name>
        <managed-bean-class>
        jsf.FirstName</managed-bean-class>
        <managed-bean-scope>
        request</managed-bean-scope>
    </managed-bean>
    <managed-bean>
        <managed-bean-name>
        lastName</managed-bean-name>
        <managed-bean-class>
        jsf.LastName</managed-bean-class>
        <managed-bean-scope>
        request</managed-bean-scope>
    </managed-bean>
```

This completes the description of Model to Controller.

11.8.4 Define and implement View (V) in application

1. Right click on **WebContent** and create a new folder with the name *pages*.


New Folder

Folder

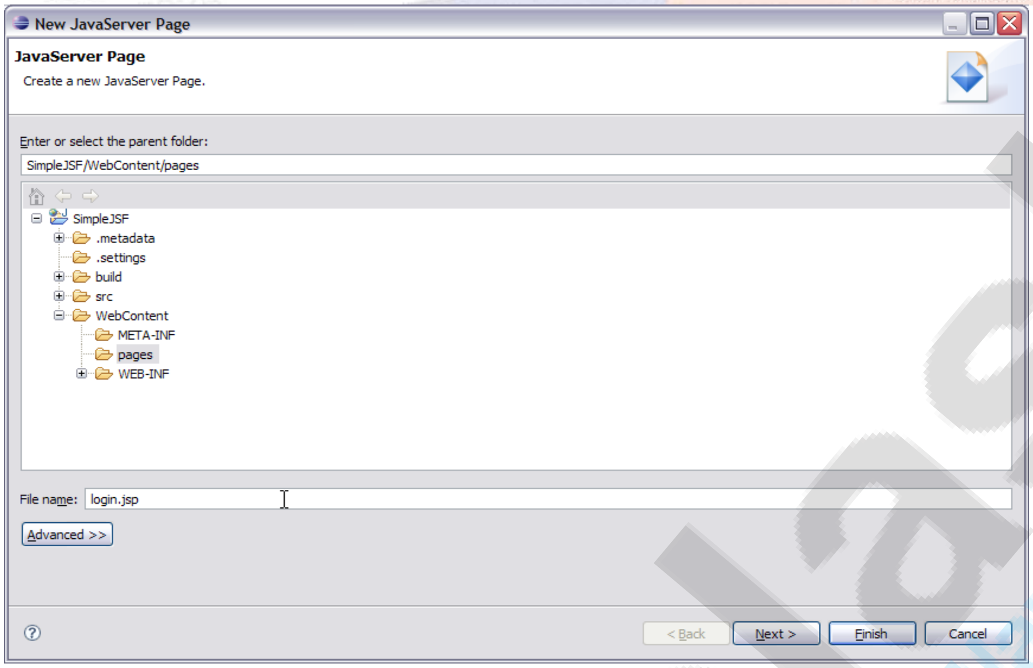
Create a new folder resource.

Enter or select the parent folder: SimpleJSF/WebContent

- SimpleJSF
  - .metadata
  - .settings
  - build
  - src
  - WebContent

Folder name: pages

2. Right click on **pages** folder and create a JSP called `login.jsp`. Select **Finish**.



3. Similarly create another JSP page called `welcome.jsp`.

4. Now we have to include the `Tag Library Descriptors` (TLD) in our application. Geronimo comes packaged with the required TLD's, which can be found in:

Location of TLD

```
<GERONIMO_HOME>\repository\org\apache\myfaces\core\myfaces-impl\1
.2.3\myfaces-impl-1.2.3.jar\META-INF\myfaces-html.tld
```

and

```
<GERONIMO_HOME>\repository\org\apache\myfaces\core\myfaces-impl\1
.2.3\myfaces-impl-1.2.3.jar\META-INF\myfaces_core.tld
```

5. To add these two TLD's in the application, in Eclipse under the Project Explorer right click on **WEB-INF**. Create a folder called **tld**. Copy `myfaces-html.tld` and `myfaces_core.tld` to this folder.

6. The next step is to populate login.jsp and welcome.jsp with data

Login.jsp

```
<%@ taglib uri="/WEB-INF/tld/myfaces-html.tld" prefix="h" %>

<%@ taglib uri="/WEB-INF/tld/myfaces_core.tld" prefix="f" %>

<html>

<head>

<metahttp-equiv="Content-Type"
content="text/html;charset=ISO-8859-1">

<title>Welcome to Apache Geronimo</title>

</head>

<body>

<f:view>

    <h1><h:outputTextvalue="Welcome to Apache Geronimo" /></h1>

    <h:form>

        <h:messagefor="firstName" style="color: red;" />

        <h:messagefor="lastName" style="color: red;" />

        <br>

        <h:outputTextvalue="Enter your first name" />

        <br>

        <h:inputTextid="firstName" value="#{firstName.name}"
             required="true">

            <f:validateLengthminimum="4" maximum="10" />

        </h:inputText>

        <br>

        <h:outputTextvalue="Enter your last name" />
```

```
         <br>

         <h:inputTextid="lastName" value="#{lastName.LName}"
           required="true">

             <f:validateLengthminimum="3" maximum="10" />

         </h:inputText>

         <br>

         <h:commandButtonid="submit" action="validated"
value="Enter" />

    </h:form>

</f:view>

</body>

</html>
```

Welcome.jsp

```
<%@ taglib uri="/WEB-INF/tld/myfaces-html.tld" prefix="h"%>

<%@ taglib uri="/WEB-INF/tld/myfaces_core.tld" prefix="f"%>

<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<metahttp-equiv="Content-Type"
content="text/html;charset=ISO-8859-1">
```

```
<title>Welcome</title>

</head>

<body>

<f:view>

<h3><h:outputTextvalue="You have successfully logged in: " />
<h:outputTextvalue="#{firstName.name} " />
<h:outputTextvalue="#{lastName.LName}" />

<h:outputTextvalue="!" /></h3>

</f:view>

</body>

</html>
```

The first two lines in login.jsp defines two tag libraries

Code Snippet from login.jsp

```
<%@ taglib uri="/WEB-INF/tld/myfaces-html.tld" prefix="h" %>
```

and

```
<%@ taglib uri="/WEB-INF/tld/myfaces_core.tld" prefix="f" %>
```

These two sets of tags are defined by JSF. The first one with the namespace "h" is used to generate html views. The second one with the namespace "f" handles the core functionalities of JSF like type conversions, validations and listeners for input from user.

The next few lines contains the usual html tags

Code Snippet from login.jsp

```
<html>

<head>
```

```
<metahttp-equiv="Content-Type"
content="text/html;charset=ISO-8859-1">

<title>Welcome to Apache Geronimo</title>

</head>

<body>
```

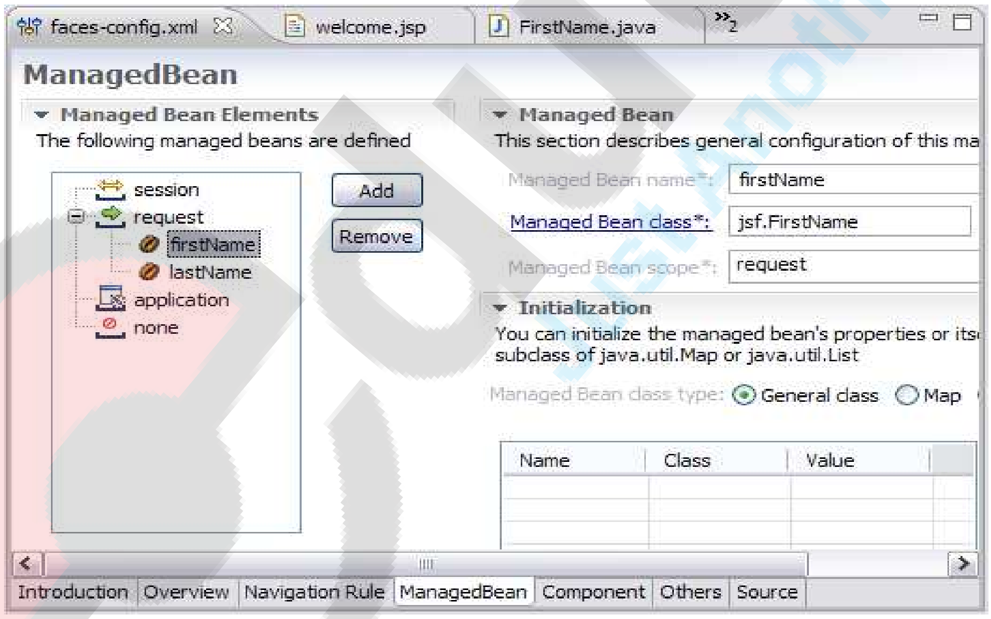
The tag **<f:view>** represents the start of JSF code.

This line of code Represents the input tag. The **id="firstName"** and **value="firstName.name"** comes from the Managed Bean.

Code Snippet from login.jsp

```
<h:inputTextid="firstName" value="#{firstName.name}"
required="true">
```

7. Using the Faces Configuration Editor, select **firstName** bean under *Managed Bean* tab. The Managed Bean Name is **firstName**. See the figure below.


faces-config.xml
welcome.jsp
FirstName.java
ManagedBean
Managed Bean Elements
The following managed beans are defined
session
request
firstName
lastName
application
none
Add
Remove
Managed Bean
This section describes general configuration of this ma
Managed Bean name*: firstName
Managed Bean class*: jsf.FirstName
Managed Bean scope*: request
Initialization
You can initialize the managed bean's properties or its
subclass of java.util.Map or java.util.List
Managed Bean class type: General class Map
Name
Class
Value
Introduction
Overview
Navigation Rule
ManagedBean
Component
Others
Source

This completes the implementation of View (V) in the application.

11.8.5 Define the Validator Component

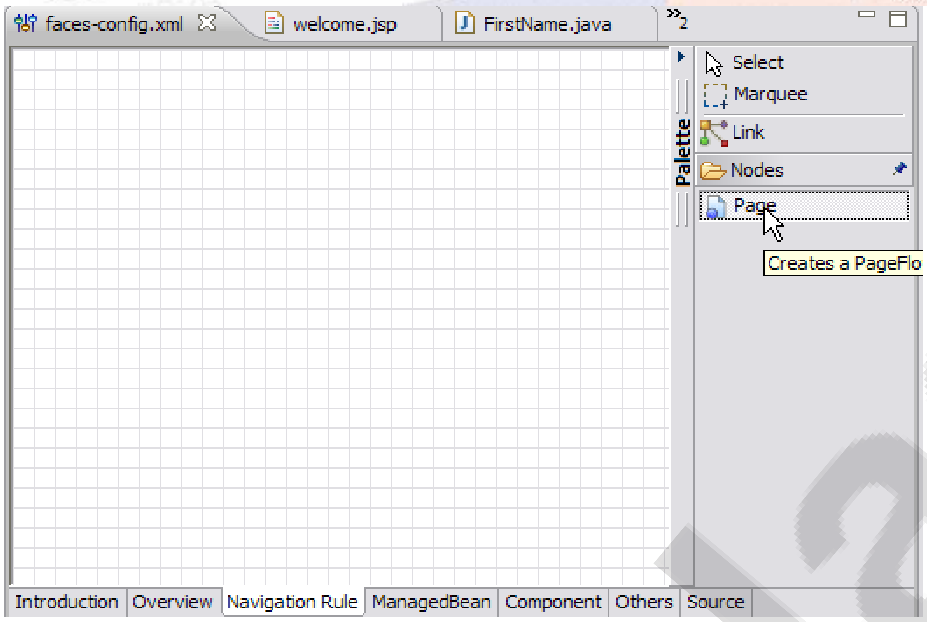
The code `<f:validateLength minimum="4" maximum="10"/>` defines the input text length to be minimum of 4 characters and maximum of 10 characters. This is the standard validation provided by core tag libraries. Other examples of validators are `Validate Long Range` tag, `Validate Double Range` tag , and so on. JSF also provides a Validator interface which can be implemented to create custom validators.

The code `<h:message for="" style="color: red;"/>` defines the error message. When the user inputs the controller validates each of the inputs. If the inputs are invalid Controller displays the same page again with an error message for the errors. The **color:red** suggests that the error message will be displayed in red color.

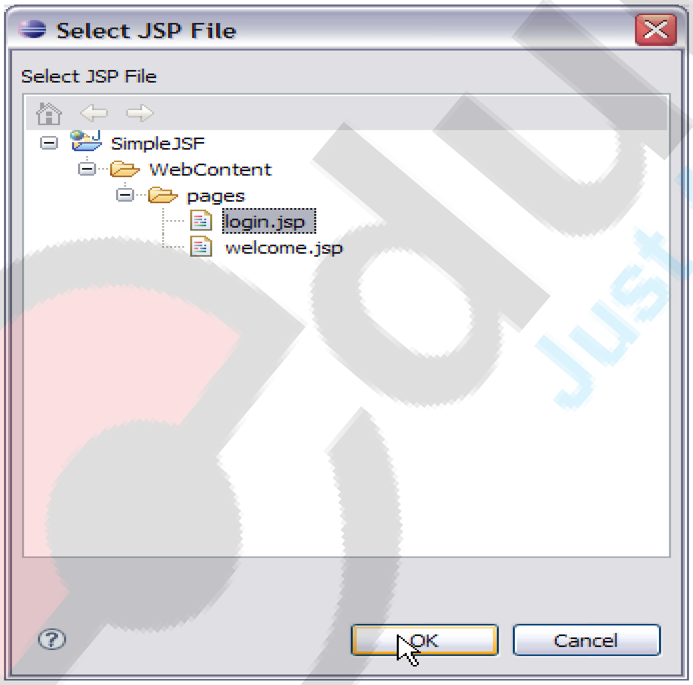
11.8.6 Define and implement the View navigation by Controller (C)

This step uses the JSP page navigation in the order of user inputs and validation by controller. If all the inputs are valid than the controller performs the action as suggested by the HTML form. This action is submitted by the HTML form as a command button.
The code in the input.jsp `<h:commandButton id="submit" action="validated" value="Enter" />` checks to determine if all the inputs are valid. This is the button which submits the form to controller if all inputs are valid. In this case the **commandButton** tells the controller to execute the validated action if all the inputs are valid.
The pages navigation in a JSF applicaiton is defined by `faces-config.xml`. Follow the steps before to define the pages navigation.

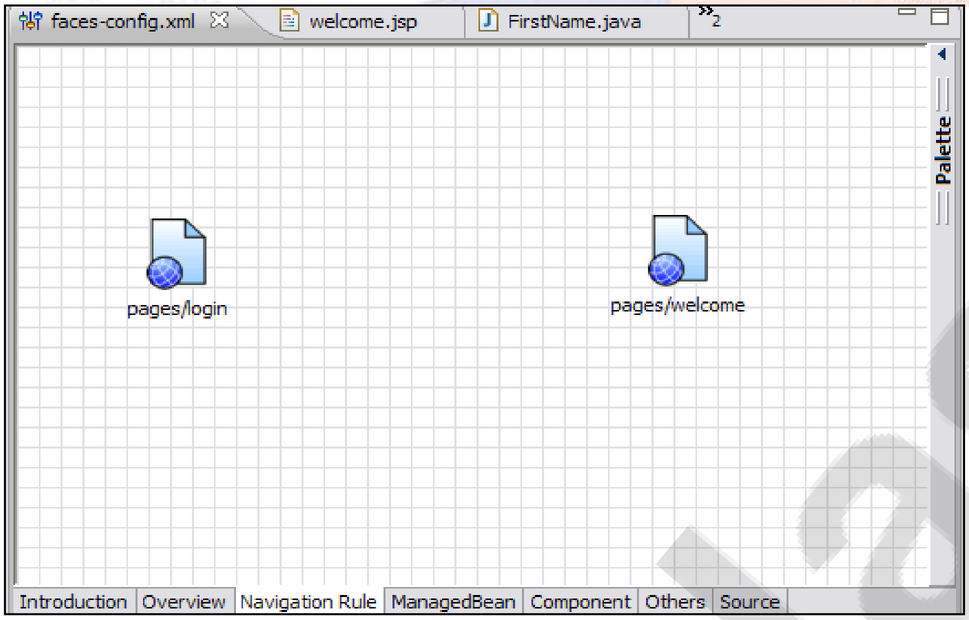
1.Launch the Faces Configuration Editor by double clicking on `faces-config.xml`
2. Select the *Navigation Rule* tab in the Configuration Editor. Under the Palette window select **Page**. This will select a `PageFlow Page` GUI object.



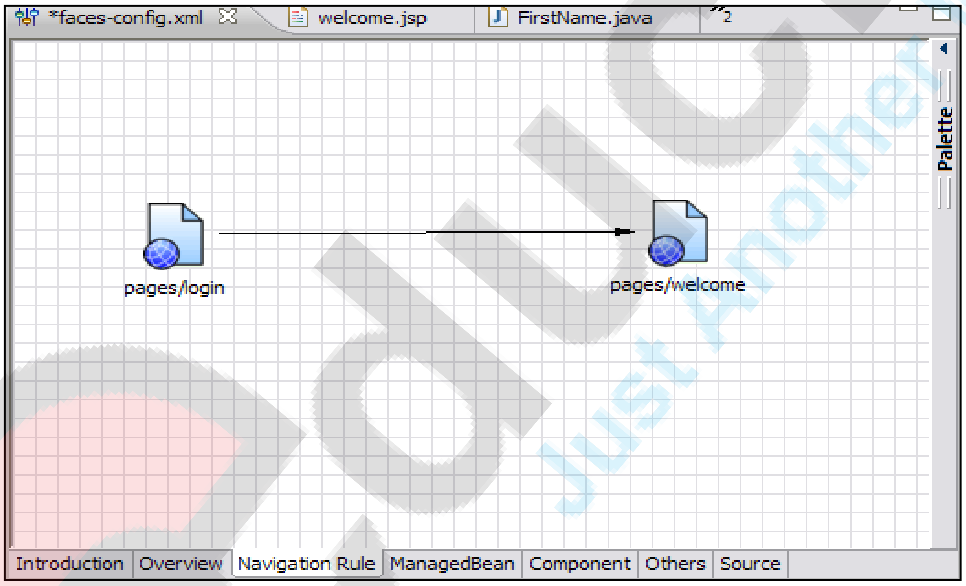
3. Drag the mouse over the Navigation Rule Window and click on the window. This will give a *Select JSP File* window. Select the `login.jsp` as shown in the figure and select **OK**.



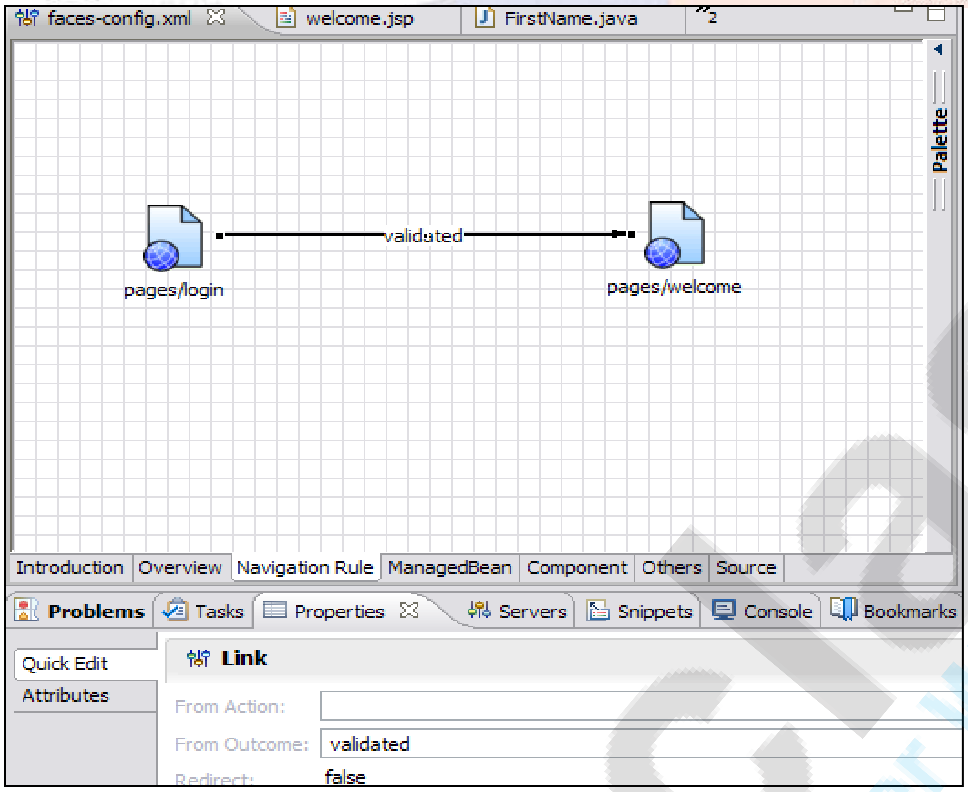
4. Similarly add the `welcome.jsp` page on the Navigation Rule window. See the figure below:



5. Select **Link** from the Palette window and join the two pages as shown in the figure:

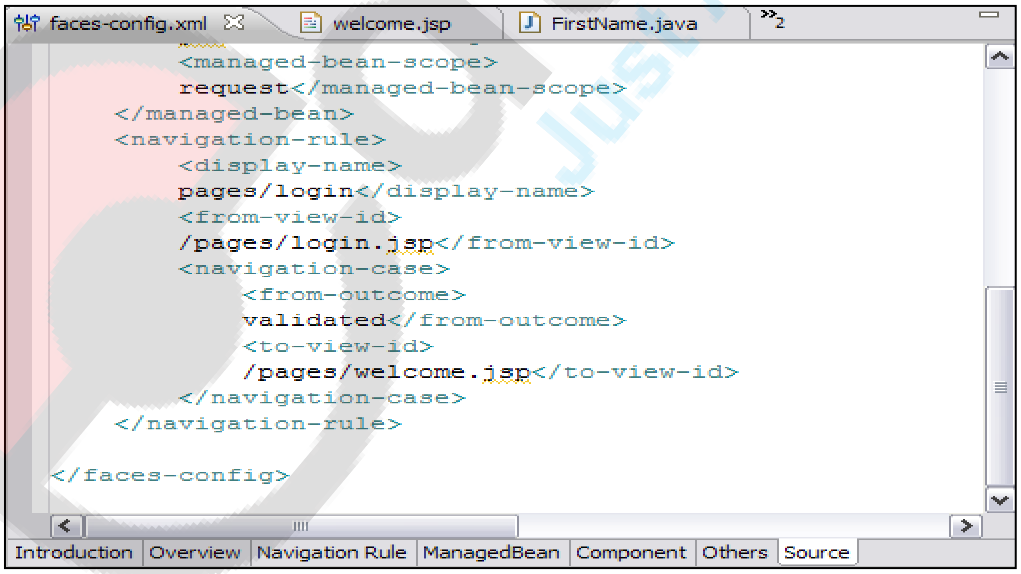


6. Select the link between the two pages and go to properties view and set the value for *From Outcome* field as **validated**. This is because of the tag **<h:commandButton id="submit" action="validated" value="Enter"** />. Once all the inputs are valid the action taken is **validated**. See the figure.



From Action:
From Outcome: validated
Redirect: false

7. Once done have a look the *Source* tab in the Faces Navigation Editor. A `<navigation-rule>` tag has been introduced into the `faces-config.xml`. This rule instructs the Controller that if all the inputs are valid from a form in the `/pages/login.jsp`, and the action is **validated**, then go to page `/pages/welcome.jsp`.



```
        <managed-bean-scope>
        request</managed-bean-scope>
    </managed-bean>
    <navigation-rule>
        <display-name>
        pages/login</display-name>
        <from-view-id>
        /pages/login.jsp</from-view-id>
        <navigation-case>
            <from-outcome>
            validated</from-outcome>
            <to-view-id>
            /pages/welcome.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>

</faces-config>
```

8. Now lets add a index.jsp under **WebContent** as follows:

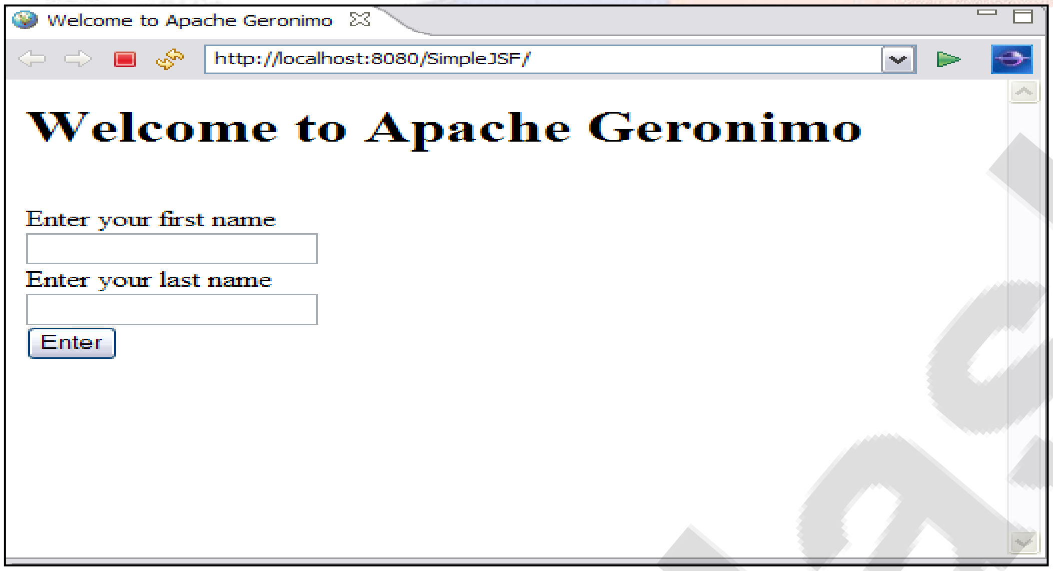
Index.jsp

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"pageEncoding="ISO-8859-1"%>

<\!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<body>

<jsp:forwardpage="/pages/login.jsf" />

</body>

</html>
```

What is the `login.jsf` in the **forward path** tag. If you look at the web.xml, ***.jsf** is used as the URL pattern to suggest that forwarded page be taken care by Java Server Faces Servlet.
This completes the Application Development process. The next step is to deploy and test the application.

11.8.7 Deploy and Test the application

Right click on the project `SimpleJSF` and select **Run As -> Run On Server**. This will deploy the sample on the Apache Geronimo Server and a Login page will be launched.


Welcome to Apache Geronimo

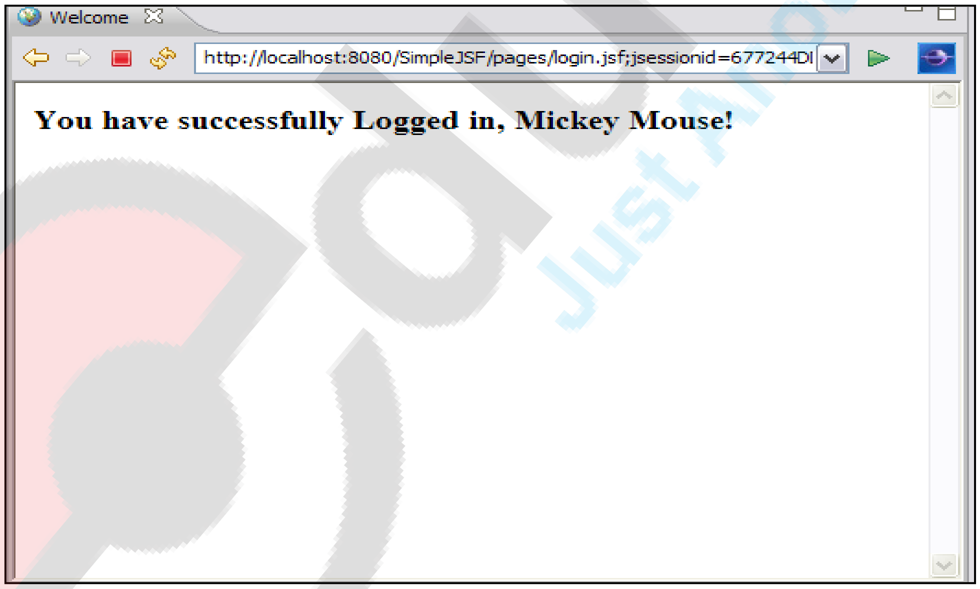
http://localhost:8080/SimpleJSF/

Welcome to Apache Geronimo

Enter your first name

Enter your last name

Lets give some sample inputs:
Sample Input #1:
First Name: Mickey
Last Name: Mouse
Both the First Name as well as Last Name fulfills the validation rules, so this form will be submitted to controller and according to the navigation rule controller will launch a `welcome.jsp` page.

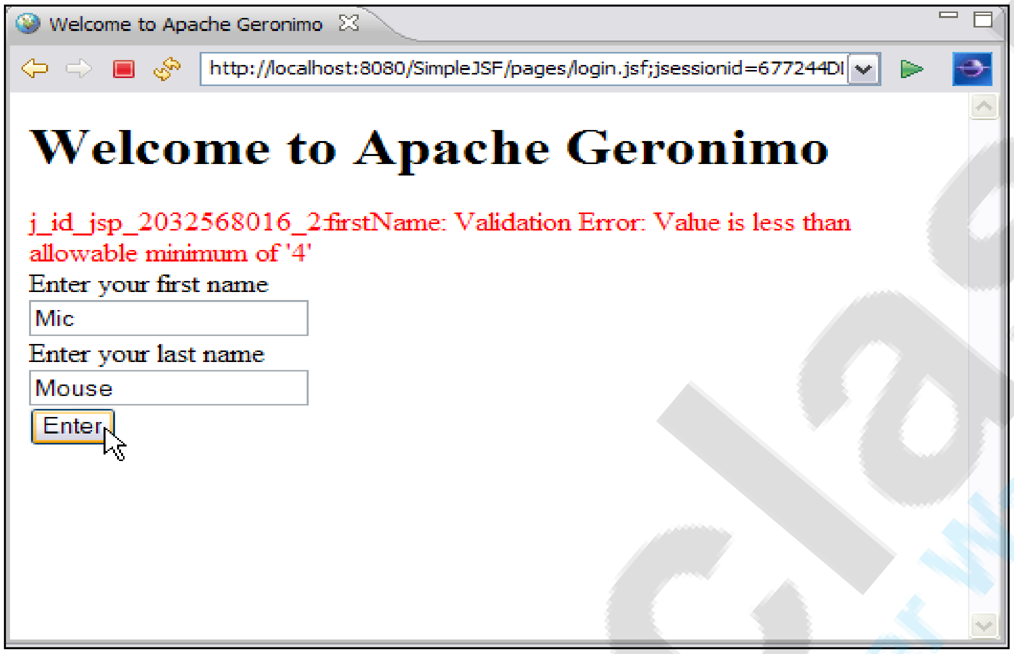

Welcome

http://localhost:8080/SimpleJSF/pages/login.jsf;jsessionid=677244DI

You have successfully Logged in, Mickey Mouse!

Sample Input #2:
First Name: Mic
Last Name: Mouse

First Name should be minimum of length=4 but in this case First Name is of length=3. In this case validation will fail and an error message will be generated by controller for First Name field.


Welcome to Apache Geronimo
http://localhost:8080/SimpleJSF/pages/login.jsf;jsessionid=677244DI

Welcome to Apache Geronimo

j_id_jsp_2032568016_2:firstName: Validation Error: Value is less than allowable minimum of '4'

Enter your first name

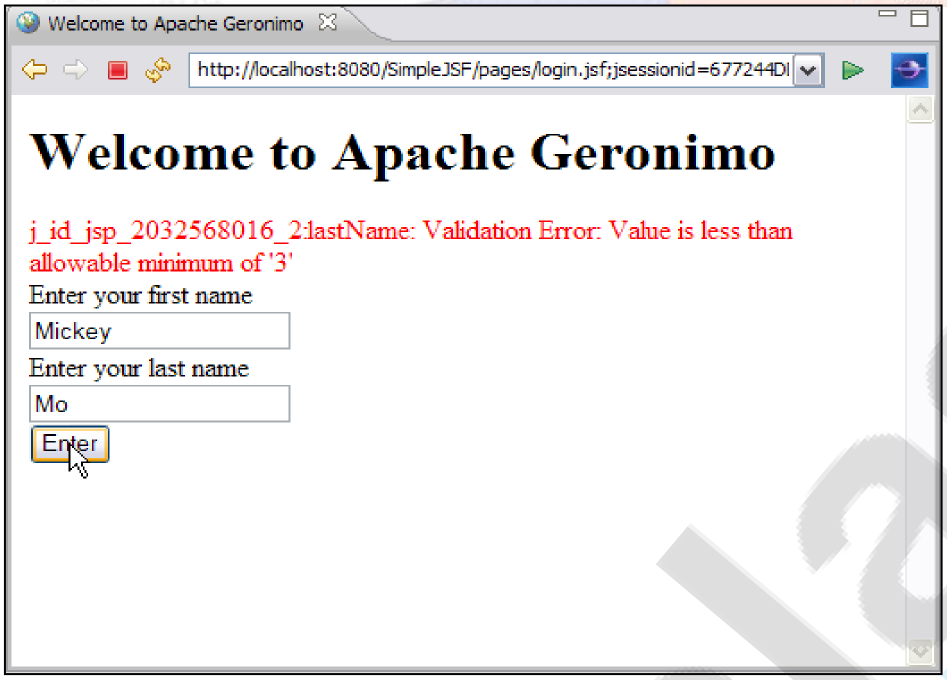
Mic

Enter your last name

Mouse

Enter

Sample Input #3:
First Name: Mickey
Last Name: Mo
Last Name should be minimum of length=3 but in this case Last Name is of length=2. In this case validation will fail and an error message will be generated by controller for Last Name field.


Welcome to Apache Geronimo
http://localhost:8080/SimpleJSF/pages/login.jsf;jsessionid=677244DI
Enter

Welcome to Apache Geronimo

j_id_jsp_2032568016_2:lastName: Validation Error: Value is less than allowable minimum of '3'

Enter your first name

Mickey

Enter your last name

Mo