Developing Applications Using Java Web Frameworks

Session - 5

Introduction to JavaServer Faces

Objectives

- ❑ Explain JSF framework
- ❑ Explain the components of JSF framework
- ❑ Explain the different types of configuration files used in JSF applications
- ❑ Describe JSF architecture and JSF lifecycle
- ❑ Explain the process of developing JSF application
- ❑ Explain UI components and the renderers in JSF
- ❑ Explain the concept of managed beans in JSF
- ❑ Explain basic tag, converter tag, and validator tag in JSF

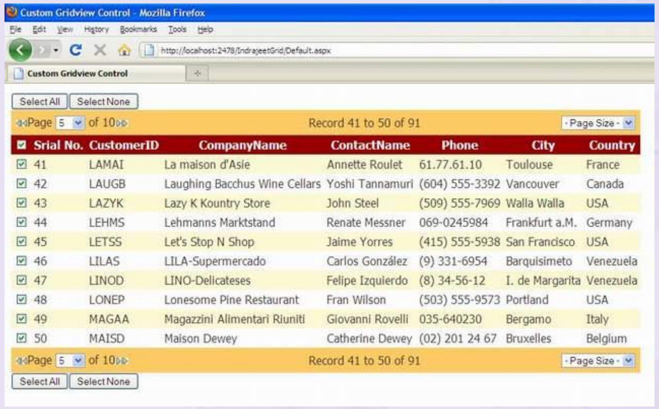
Introduction 1-3

- Sun Microsystems provided two specifications for developing server-side components namely,
  - Servlet - Helps to generate dynamic contents
  - JSP pages - Separate presentation from the application logic

- **Limitations of Servlet and JSP**
  - Both the technologies fail to provide separation of User Interface (UI) components from the model.
  - Developers are dependent on client specific UI for designing the Web pages.
  - For example, for a Web client, HTML Web pages are designed.
  - Apart from static nature of HTML controls, there is a lack of event handling on the controls.
  - For example, a View component in MVC architecture should generate events on the controls that can be handled.

Introduction 2-3

- Some of the challenges faced by the developer, while designing UI controls for the clients of the Web application are as follows:
  - **Rich client controls**
    - Based on the ability to query the model and updated the data with event-handling mechanism.
  - **Support for custom controls**
    - Web applications need to be customized with custom components, such as a grid viewer.


Custom Gridview Control - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:2478/IndrajeetGrid/Default.aspx
Custom Gridview Control
Select All
Select None
Page 5 of 10
- Page Size -

Record 41 to 50 of 91

☑	Srial No.	CustomerID	CompanyName	ContactName	Phone	City	Country
☑	41	LAMAI	La maison d'Asie	Annette Roulet	61.77.61.10	Toulouse	France
☑	42	LAUGB	Laughing Bacchus Wine Cellars	Yoshi Tannamuri	(604) 555-3392	Vancouver	Canada
☑	43	LAZYK	Lazy K Kountry Store	John Steel	(509) 555-7969	Walla Walla	USA
☑	44	LEHMS	Lehmanns Marktstand	Renate Messner	069-0245984	Frankfurt a.M.	Germany
☑	45	LETSS	Let's Stop N Shop	Jaime Yorres	(415) 555-5938	San Francisco	USA
☑	46	LILAS	LILA-Supermercado	Carlos González	(9) 331-6954	Barquisimeto	Venezuela
☑	47	LINOD	LINO-Delicateses	Felipe Izquierdo	(8) 34-56-12	I. de Margarita	Venezuela
☑	48	LONEP	Lonesome Pine Restaurant	Fran Wilson	(503) 555-9573	Portland	USA
☑	49	MAGAA	Magazzini Alimentari Riuniti	Giovanni Rovelli	035-640230	Bergamo	Italy
☑	50	MAISD	Maison Dewey	Catherine Dewey	(02) 201 24 67	Bruxelles	Belgium

Record 41 to 50 of 91

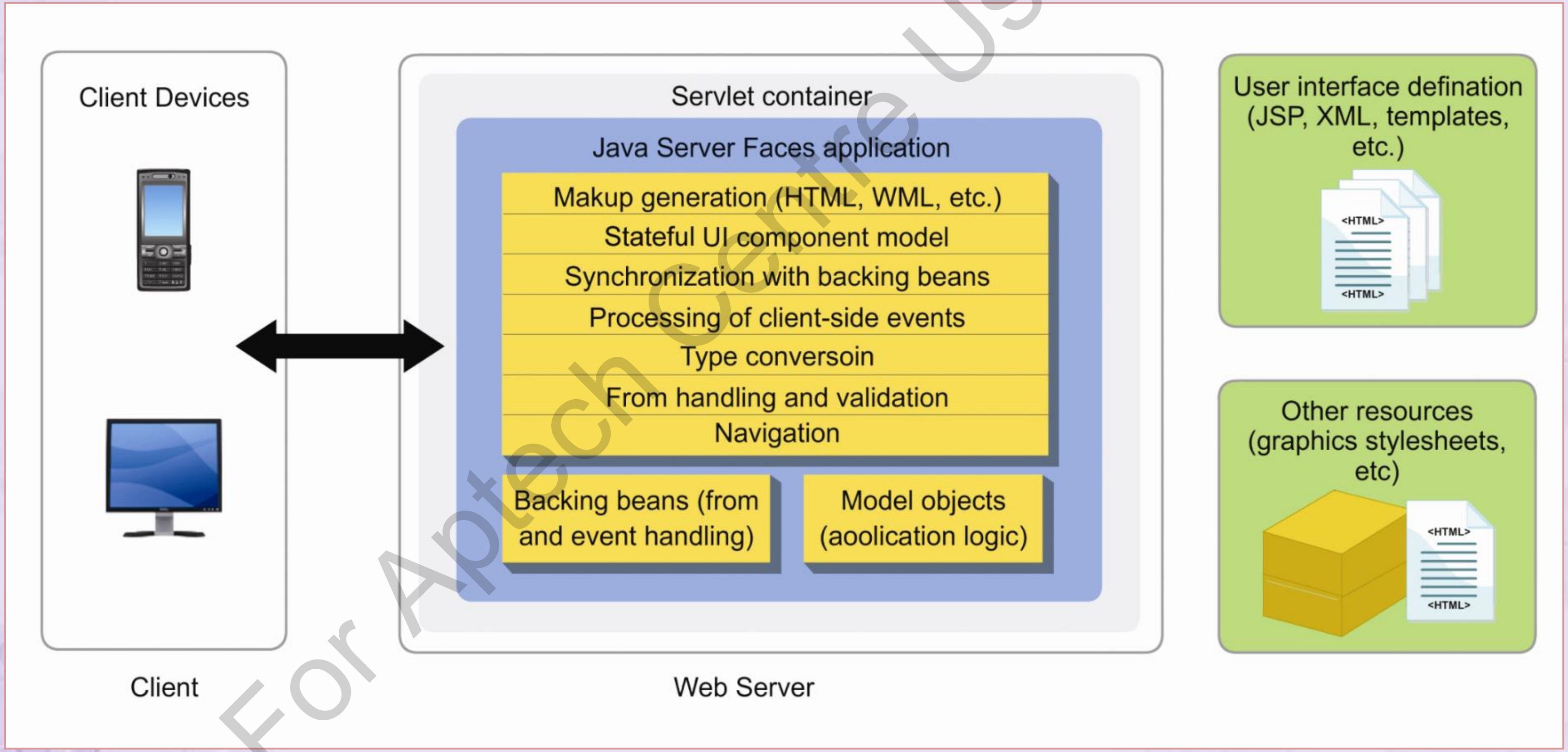
Introduction 3-3

- Thus, a need was felt by the developers for a development environment that would allow them to:
  - Implement custom UI components and add or remove them dynamically.
  - Provide UI support to various clients such as Web browser, PDA, mobile phones, and so on.
  - Use an event model such as Swing to trap events fired by components or controls in Web pages.

JSF is an open source, component based, event driven framework for building user interface for Web applications.

JSF Framework 1-3

- Following figure shows high-level architecture of JSF application:

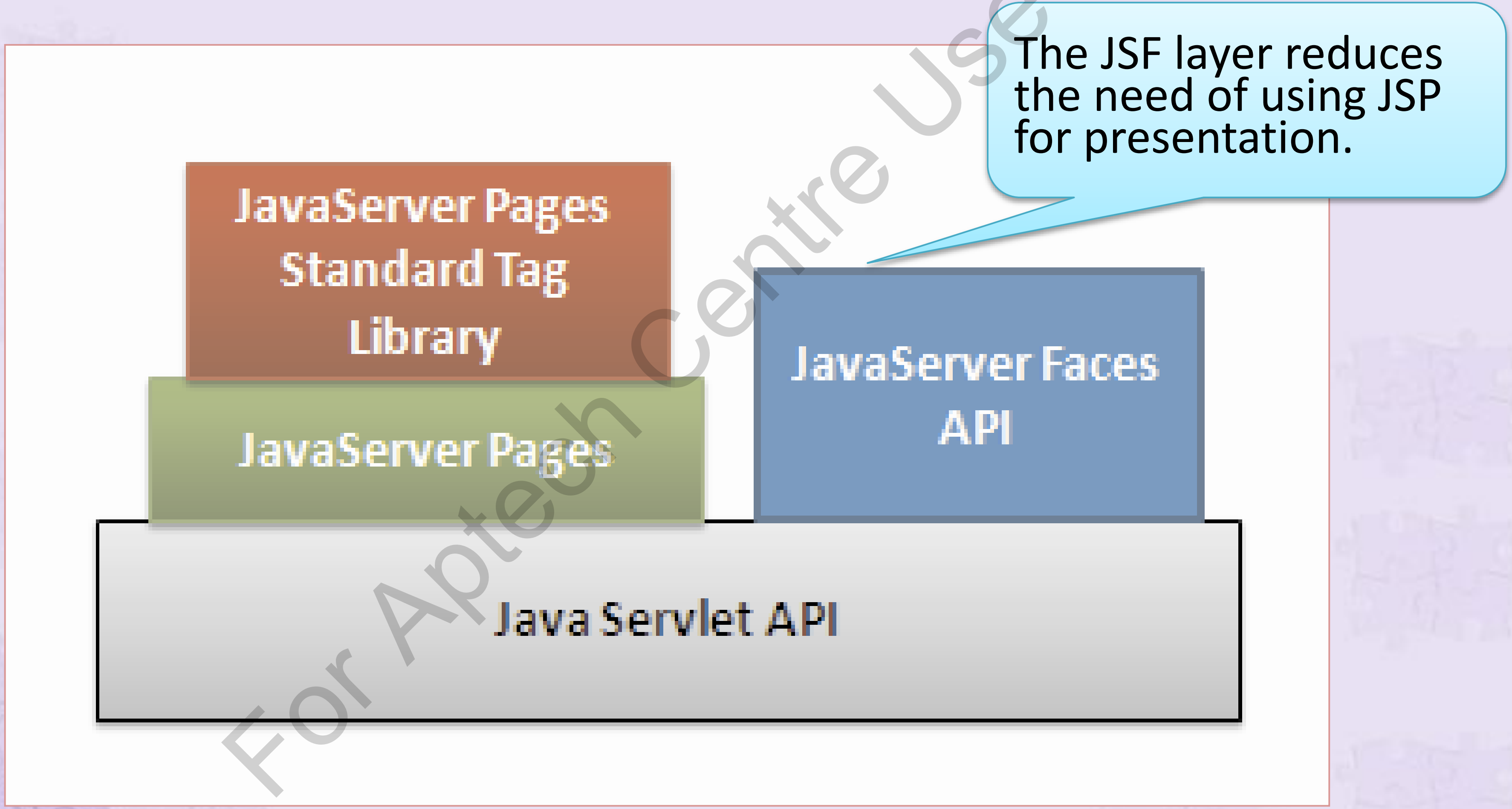

Client Devices
Client
Servlet container
Java Server Faces application
Makup generation (HTML, WML, etc.)
Stateful UI component model
Synchronization with backing beans
Processing of client-side events
Type conversoin
From handling and validation
Navigation
Backing beans (from and event handling)
Model objects (aoolication logic)
Web Server
User interface defination (JSP, XML, templates, etc.)
<HTML>
<HTML>
Other resources (graphics stylesheets, etc)
<HTML>
<HTML>

JSF Framework 2-3

- Component-based framework of JSF defines:
  - Support for standard UI components such as button, text fields, checkboxes, and so on.
  - Supports third-party components.
  - Supports the architecture for displaying same components to different types of clients such as Web browsers, PDAs, and mobiles.
  - Supports deployment of the JSF application at the server-side.
  - Supports execution within a Web container.

JSF Framework 3-3

- Following figure shows the JSF APIs built on top of Servlet API:


The JSF layer reduces the need of using JSP for presentation.
JavaServer Pages Standard Tag Library
JavaServer Pages
JavaServer Faces API
Java Servlet API

Components of JSF 1-5

- Components of JSF are as follows:
  - Managed Beans
  - UI Components
  - Converter
  - Validator
  - Renderers
  - Other core components:
    - Events and listeners
    - Messages
    - Navigation

Components of JSF 2-5

- **Managed Beans**
  - Associates UI component with a JavaBean which defines UI component's properties.
  - Defines methods that perform functions such as event handling, validation, and navigation processing, associated with the component.
  - Gets instantiated at runtime using managed bean creation facility.
- **UI Components**
  - Focuses on interacting with the end user, also called controls or components.
  - Can remember their state automatically.
  - JSF `UIComponent` classes specify the UI component's functionality.

Components of JSF 3-5

- **Converter**
  - Converts form (UIComponent) data to Java objects for storing onto the model data and from model's Java object to presentation view.
  - Ensure that the user's input is in a specified format as specified by the converter.
  - Provides a set of standard Converter implementations as well as facility to customize the Converter implementations.
- **Validator**
  - Supports data validation before the form (UIComponent) values update the object data.
  - Defines a collection of standard classes for common data validations and a developer can also define custom validations.

Components of JSF 4-5

- **Renderers**
  - JSF component architecture separates the functional aspect of a UIComponent from the rendering of UI components.
  - Separate classes called renderers handle the rendering process.
  - Collection of renderer classes forms a render kit.
  - The render kit defines mapping of component classes to component tags suitable for a specific client.
- **Events and Listeners**
  - Event are represented by an instance of an event class and there is an event listener interface for every JSF component event.
  - When a component is used by activating it, such as **button clicked**, then an event is fired. JSF implementation then invokes the corresponding listener method to process the event.

Components of JSF 5-5

- **Messages**

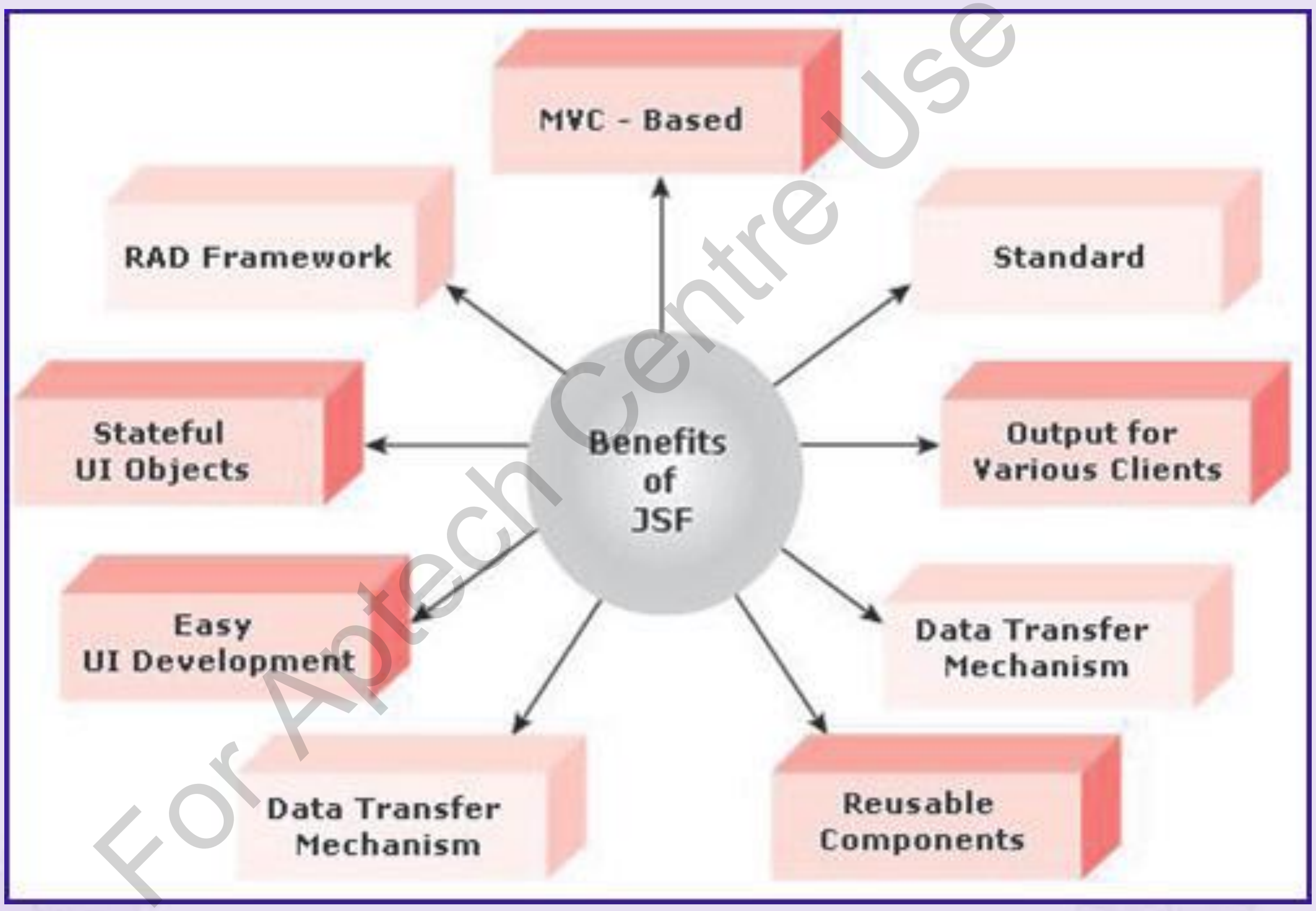
  - Display either information or errors.
  - Two types of error messages are Logical Error messages and Input Error messages.
  - **Logical Error messages** - Can be generated either due to errors in business logic or database errors or connection errors.
  - **Input Error messages** - Can be generated due to user input errors such as empty field or invalid text and so on.

- **Navigation**

  - Allows the developer to use a configuration file to specify and control the page navigation.
  - Takes place when a user clicks either a command button or a command link.
  - Provides a default action listener for such navigational event.

Benefits of JSF

- Following figure depicts the benefits of JSF:



Comparison between JSP and JSF

- Following table shows the comparison between JSP and JSF:

JavaServer Pages (JSP)	JavaServer Faces (JSF)
Java based technology that is utilized to develop and design dynamic Web pages.	Web application, which is used to simplify development and integration of Web based UIs.
Does not support Validator, Conversion, and Ajax support.	Supports Validator, Conversion, and Ajax support.
No features such as managed beans and template-based component system.	Core features such as managed beans and template-based component system.
Not a request-driven MVC. Accessed by the dynamically generated pages such as XML or HTML.	Request-driven MVC.
Compiled within the server, not within a view template.	Interface within a view template.

Introduction to JSF Versions

- Since its first release, JSF have gone through major and minor version enhancements:
- **JSF 1.0**
  - Initial specification released which combines the MVC design pattern with a powerful component-based UI framework.
- **JSF 1.2**
  - Release was introduced with many improvements and used on Java EE 5 platform.
- **JSF 2.0**
  - Was introduced with the main focus of simplifying and building of UI components.
- **JSF 2.1 and 2.2**
  - Facelets Views.
  - File Upload Component.

Introduction to JSF Configuration Files

- A developer needs to use minimum of two XML configuration files, while working with JSF.
- These two files are as follows:
  - `web.xml`
  - `faces-config.xml`
- The configuration files let the Java code to be easily shared between JSP pages.

web.xml File 1-2

- Located in the `/WEB-INF/` directory of the Web application that is to be deployed, and configures the Web server end of the application.
- Configures `FaceServlet.`
- A `FacesServlet` is:
  - Responsible for handling JSF applications.
  - Central controller of JSF application.
  - Receives all requests for the JSF application.
  - Initializes the JSF components before the JSP is displayed.

web.xml File 2-2

- Following code snippet displays the configuration of `FacesServlet` in `web.xml`:

```
<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 to URL pattern -->
<Servlet-mapping>
  <Servlet-name>Faces Servlet</Servlet-name>
   <url-pattern>*.xhtml</url-pattern>
</Servlet-mapping>
<Servlet-mapping>
    <Servlet-name>Faces Servlet</Servlet-name>
    <url-pattern>*.jsf</url-pattern>
</Servlet-mapping>
<Servlet-mapping>
     <Servlet-name>Faces Servlet</Servlet-name>
     <url-pattern>*.faces</url-pattern>
</Servlet-mapping>
</web-app>
```

- In JSF 2.0, the `FacesServlet` can map to different URL patterns, such as `*.xhtml`, `*.jsf`, and `*.faces`.
- If `web.xml` is not present, then the `FacesServlet` is automatically mapped to the common URLs such as `/faces/*`, `*.jsf`, `*.faces`, and so on.

faces-config.xml 1-2

- Contains the configuration of the JSF application.
- Allows the configuration of the application, converters, validators, managed beans, and navigation and also defines the behavior of the Faces Servlet.
- Tends to be more specific to a certain JSF application, whereas `web.xml` generally comprises common configuration options.

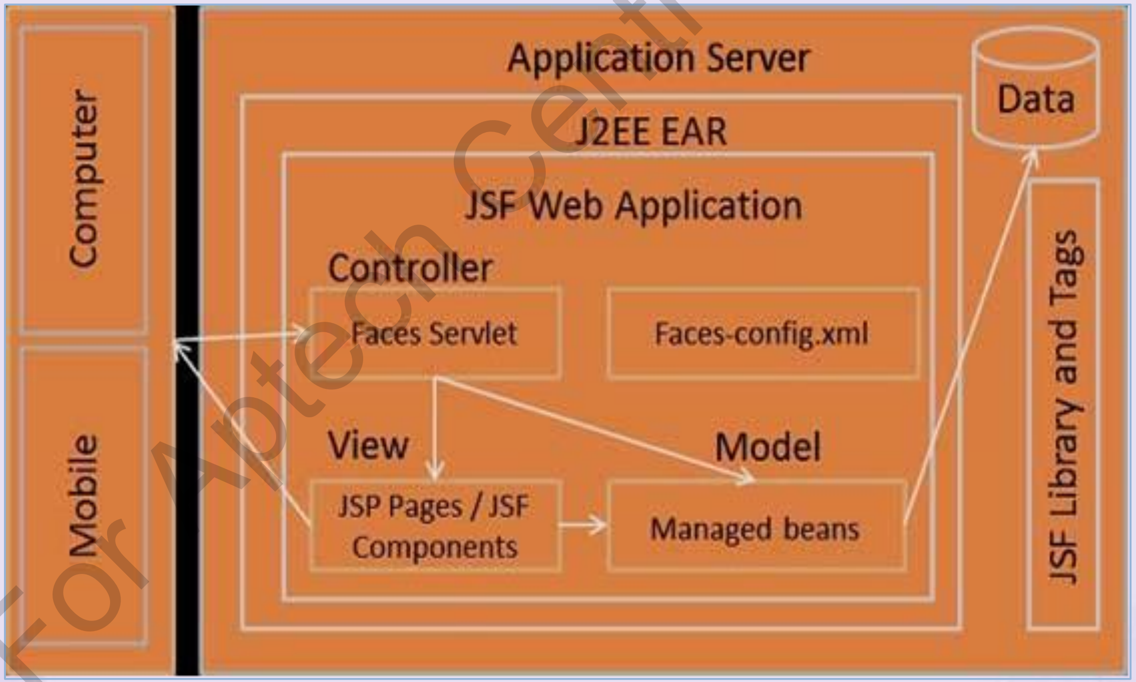
faces-config.xml 2-2

- Following code snippet demonstrates the creation of a file called `/WEB-INF/faces-config.xml`:

```
<faces-config  . . .
<!-- Configuring managed bean -->
<managed-bean>
     <managed-bean-name>messageBean</managed-bean-name>
     <managed-bean-class>JSF.SimpleController</managed-bean-class>
     <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<!-- Configuring navigation -->
<navigation-rule>
<from-view-id>/starting-page.xhtml</from-view-id>
<navigation-case>
     <from-outcome>return-value-1</from-outcome>
     <to-view-id>/result-page-1.xhtml</to-view-id>
</navigation-case>
</to-view-id>
<navigation-case>  .   . .</navigation-case>
</navigation-rule>
</faces-config>
```

JSF Architecture 1-2

- JSF is an MVC-based application framework, and it provides a rich architecture for defining user interface components, managing their state on the server, and handling client-generated events.
- Following figure shows the main components of the JSF architecture and depicts the processing flow of a client request:

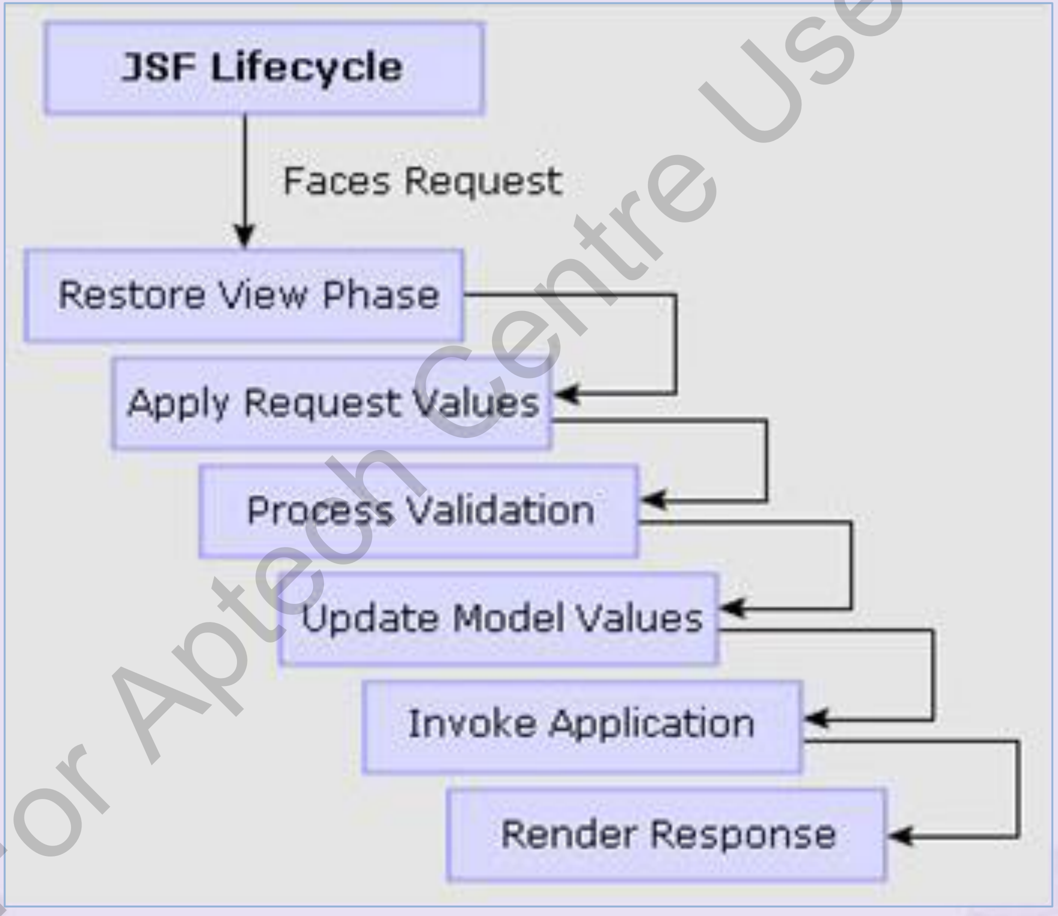


JSF Architecture 2-2

- **Controller**
  - `FacesServlet` is the entry point for all requests for a JSF page.
  - Initializes the resources needed by the framework to control the page's lifecycle and subsequently, invokes the page to process the request.
- **View**
  - The user interface components in the JSF page are represented on the server by a component tree.
  - When a request is processed by the framework, this tree is either created for an initial page request or restored from its saved state for subsequent page requests.
- **Model**
  - A Backing Bean is a JavaBean that holds the data for a JSF Page's user interface components.
  - Implements methods that support their behavior.
  - Includes logic to perform event handling, validation, and navigation control.

Introduction to JSF Life Cycle 1-4

- Following figure depicts the JSF life cycle:



Introduction to JSF Life Cycle 2-4

- **Restore View Phase**
  - This phase starts when the application responds to a request coming from `FacesServlet` controller.
  - When a request is received initially, the JavaServer Faces implementation creates a view tree for the page.
  - A new view is stored in the `FacesContext` parent object.
- **Apply Request Values Phase**
  - In this phase, each component's state is retrieved, that is, after restoring the view tree, request processing is performed.
  - Each component in the view tree extracts the new value of requested parameter by using the component name-value pair of information.
  - The value is stored locally on the component.

Introduction to JSF Life Cycle 3-4

- **Process Validations Phase**
  - In this phase, values associated with each component is validated against the validators registered for this component, as per predefined validation rules.
  - If the local value is found invalid, JSF adds an error message to the `FacesContext` instance, and the component is marked invalid.
- **Update Model Values**
  - Post validation and conversion, the actual values of server-side model classes get updated to the component's local value.
  - This involves converting the component's string data to model property type.
  - If the component's values cannot be converted, then JSF advances to Render Response phase and flag messages against the respective components.

Introduction to JSF Life Cycle 4-4

- **Invoke Application**
  - In this phase, all those application-specific events which have not yet been handled are handled by invoking the Web application.
  - All the form data that has passed through conversions and validations will be now processed as per the business logic.
- **Render Response**
  - In the final phase of JSF life cycle, the response is rendered to the client.
  - JSF implementation transfers the task of rendering the page to the JSP container, if the application is using a JSP page.
  - If this rendering is in response to an initial request, then the components represented on the page are added to the view tree.
  - After rendering, the state of the responses sent is saved for later restoration during the Restore View phase.

Developing JSF Applications

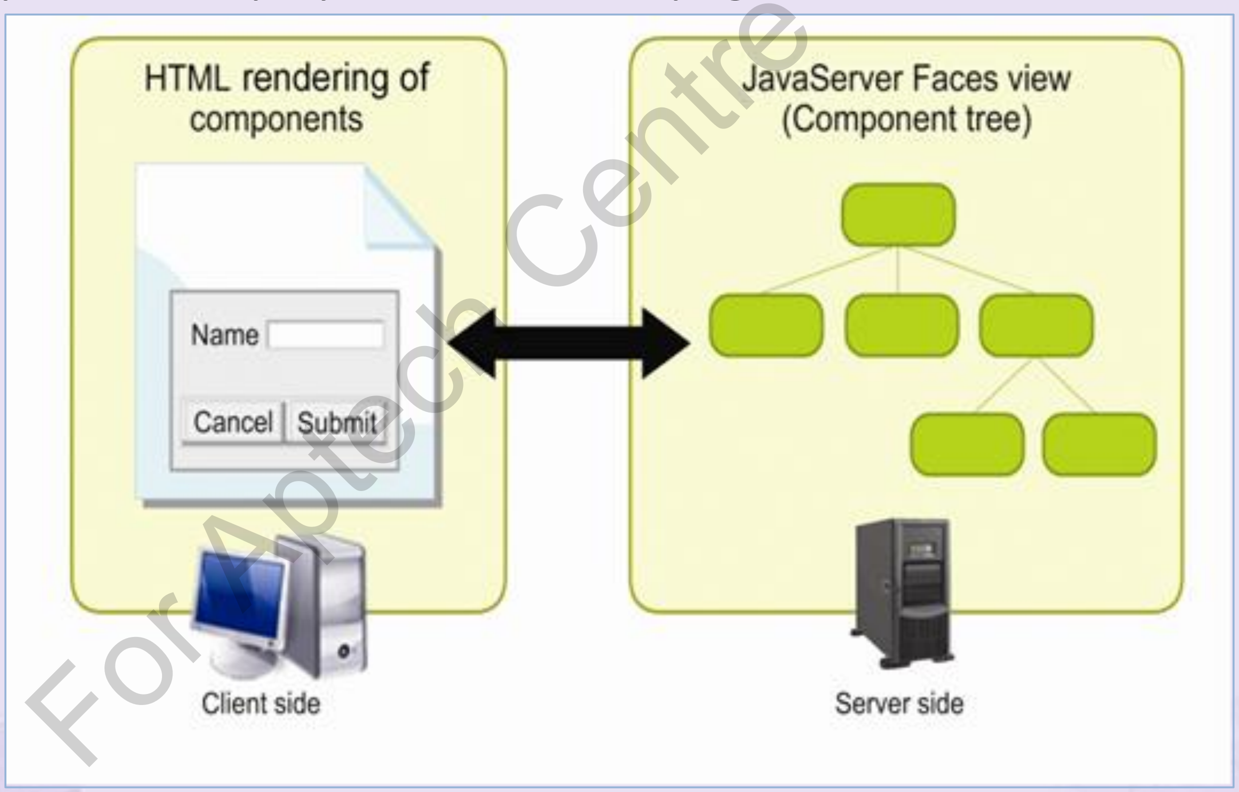
- Some of the functionalities performed by the components are as follows:
  - Custom tag library
  - JavaBean components
  - Custom tag libraries for managing event listeners, validators, converters, and other core actions
  - Server-side classes, also known as helper classes for obtaining the functionalities
  - Configuration file, `faces-config.xml`, for configuring application resources

User Interface Component Model 1-2

- **UI Components:**
  - Are simple and reusable elements that are used to design the user interface of JSP applications.
  - Can be simple such as text field or button as well as complex such as trees or data table.
  - User interface controls are represented by `UIComponent` classes on the server-side.
  - Base class for all the UI components is `UIComponentBase` class are defined in `javax.faces.component` package.
- **`UIComponentBase` class**:
  - Provides the default state and behavior for all UI components.

User Interface Component Model 2-2

- JSF framework creates a tree of UI components at server-side, while displaying them on the page. The component tree is also called as **View**.
- Following figure displays the component tree managed on the server for the UI components displayed on the Web page:


HTML rendering of components
Name
Cancel
Submit
Client side
JavaServer Faces view (Component tree)
Server side

Renderers 1-2

- The JSF API includes a standard HTML kit for rendering the component as HTML object to be sent to HTML client.
- Following table describes some of the UI component classes present in the JSF technology:

Component Class	Component Tag	Functionality	Rendered as HTML	Appearance on the Page
UIColumn	`<h:column>`	Represents a single column of data and is used with UIData component	A column of data in an HTML table	Column in a table
UICommand	`<h:commandButton>/<h:commandLink>`	Represents controls which submits the form data to an application or links to another location when activated	An HTML tag <input type="XXX" /> where XXX is a type value that can be a submit, reset, or image	A button or a hyperlink

Renderers 2-2

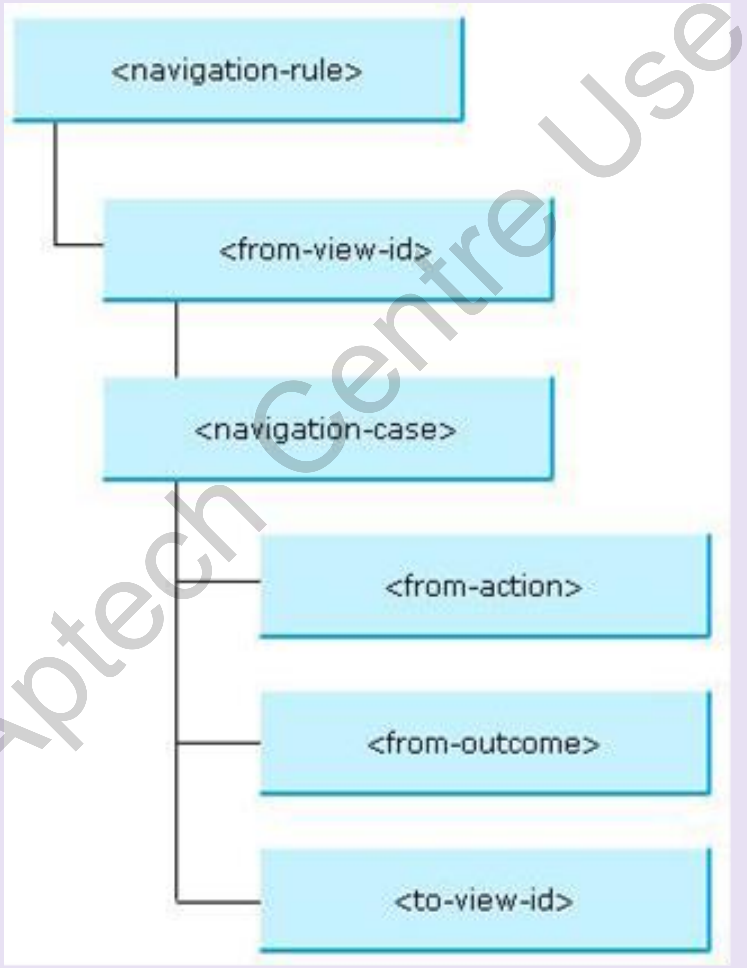
Component Class	Component Tag	Functionality	Rendered as HTML	Appearance on the Page
UIForm	`<h:form>`	Collection of controls whose data is submitted to the application for processing. This tag resembles the <form> tag in HTML	An HTML <form> element	No appearance
UIGraphics	`<h:graphicImage>`	Used to display an image	An HTML <img> tag	An image
UIPanel	`<h:panelGrid>`	Manages the layouts of child components embedded in it	An HTML <table> element with <tr> <td> elements	A table

Navigation Model 1-4

- Is an easy way to declare a set of rules that define the next view for user based on user actions.
- **Rules**
  - Are specified using XML elements in the application's configuration resource file, often named as `faces-config.xml`.
  - Uses action event invoked by clicking of button or hyperlink.
  - Also handle additional processing required to select the correct sequence in which pages are to be loaded.

Navigation Model 2-4

- Following figure depicts a navigation model:



Navigation Model 3-4

- To facilitate navigation, the Web developer performs the following steps:
  - **Define set of navigation rules**
    - `<navigation-rule>` - defines which pages should be chosen from a set of pages.
    - `<navigation-case>` - represents each path to a page.
    - A navigation rule can optionally specify the page to which it is bound with the `<from-outcome>` tag.
    - Each rule must have the `<to-view-id>` to define which page to load next.
  - **Bind these rules to the UI component** - The navigation occurs when the `UIComponent` within the page trigger an action event.

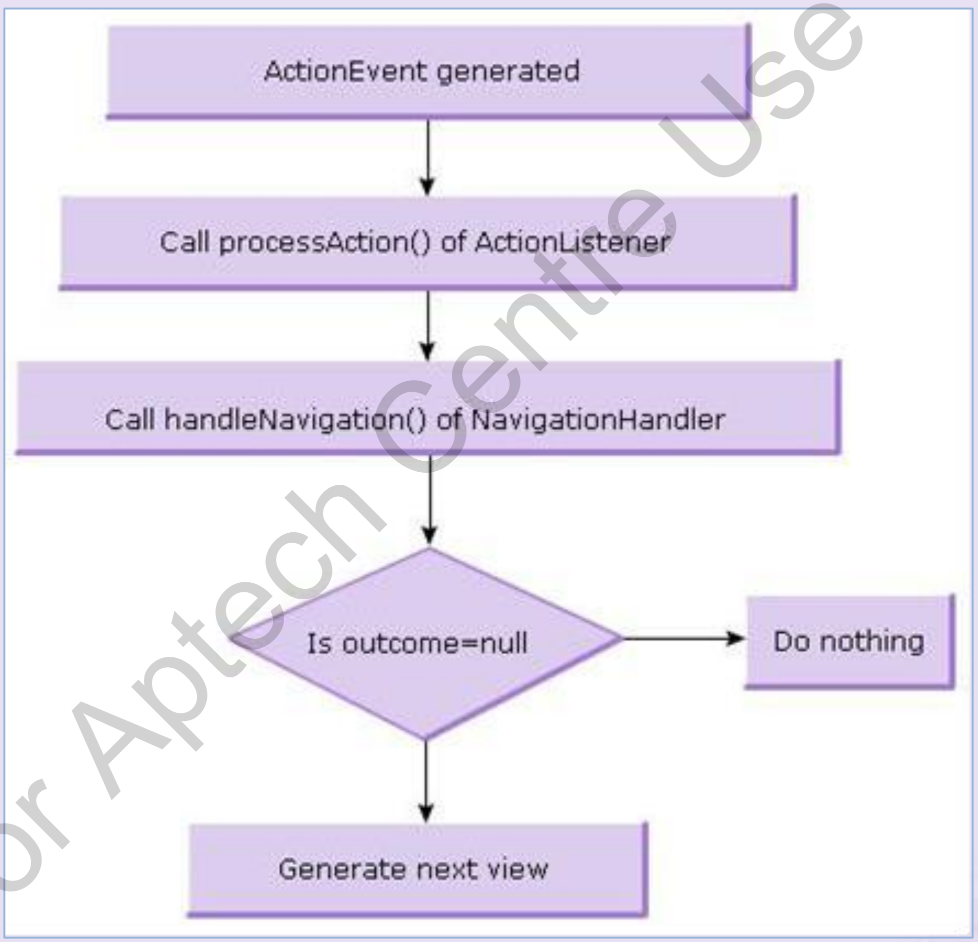
Navigation Model 4-4

- Following code snippet shows the code to handle navigation for the current view ‘`/pages/register.jsp`’:

```
…
<navigation-rule>
<from-view-id>/pages/register.jsp</from-view-id>
<navigation-case>
   <from-outcome>success</from-outcome>
   <to-view-id>/pages/welcome.jsp</to-view-id>
</navigation-case>
<navigation-case>
   <from-outcome>failure</from-outcome>
   <to-view-id>/pages/loginError.jsp</to-view-id>
</navigation-case>
</navigation-rule>
```

Handling a Navigation Event

- Following figure depicts handling of a navigation event:


ActionEvent generated
Call processAction() of ActionListener
Call handleNavigation() of NavigationHandler
Is outcome=null
Do nothing
Generate next view

Concept of Backing Bean 1-3

- **Backing Bean**:
  - Defines the objects that hold business data and perform application-specific processing on that data.
  - Contains properties and methods of UIComponents appearing in the page are defined using the backing beans.
- Some of the functions performed by managed bean methods comprise the following:
  - Handle an event activated by a component.
  - Execute the procedure of locating the next page where the application must navigate to.
  - Validate the data of a component.

Concept of Backing Bean 2-3

- Following code snippet demonstrates the creation of a backing bean:

```
. . .
public class Product {
String name;
float price;
public Product() { }
 public void setName(String name) { this.name = name; }
public String getName() { return name; }
public void setPrice (float price) { this.price = price;  }
public float getPrice() { return price; }
}
 . . .
```

- The code creates a Java bean named ‘`Product`’ with ‘`name`’ and ‘`price`’ as its properties that store the name and price of the product.

After development of backing beans, it needs to be registered with application’s configuration resource file, so that JSF implementation can create their instances automatically whenever required.

Concept of Backing Bean 3-3

- Following code snippet registers this ProductBean to `faces-config.xml` file:

```
. . .
<managed-bean>
   <managed-bean-name>ProductBean</managed-bean-name>
   <managed-bean-class>com.aptech.Product</managed-bean-class>
   <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

- The `<managed-bean-class>` tag specifies fully qualified name of class implementing the business logic.
- The `<managed-bean-scope>` tag defines the scope of **ProductBean** to be within the request object.

Binding Backing Bean

- The JSF framework supports Expression Language (EL) that can be used to access the application data stored in backing bean.
- The UIComponent can be bound to either the property or methods of backing bean.
- This is achieved by setting the value attribute of component tag to the EL expression augmented with ‘#’.
- For example,
  - `<h:inputText value=“#{Productbean.name}”/>`
  - `<h:inputText value=“#{Productbean.price}”/>`

Methods of Backing Beans

- The various functions performed by the methods of backing bean are as follows:
  - **Handling Navigation**
  - **Handling Action Event**
  - **Performing Validation**
  - **Handling ValueChange Event**

Annotations of Managed Beans 1-2

- In JSF 2.0, managed beans can be registered through annotations.
- **@ManagedBean annotation**
  - Marks the bean as a managed bean.
  - Takes the `name` attribute to specify the bean name.
- **Scope annotations**
  - Define the scope in which the bean will be stored.
  - Some of the scope annotations are as follows:
    - `@RequestScoped`
    - `@NoneScoped`
    - `@ViewScoped`
    - `@SessionScoped`
    - `@ApplicationScoped`

Annotations of Managed Beans 2-2

- Following code snippet demonstrates the creation of a backing bean:

```
. . .
@ManagedBean(name = "ProductBean", eager = "true)
@RequestScoped
public class Product {
String name;
float price;
public Product() { }
public void setName(String name) {  this.name = name;  }
public String getName() { return name;  }
public void setPrice (float price) {  this.price = price; }
public float getPrice() {  return price;  }
}
```

- The attribute `eager="true"` specifies that the managed bean is created before it is requested for the first time.
- The life of the bean is as long as the HTTP request and response exists.

JSF Tag Libraries

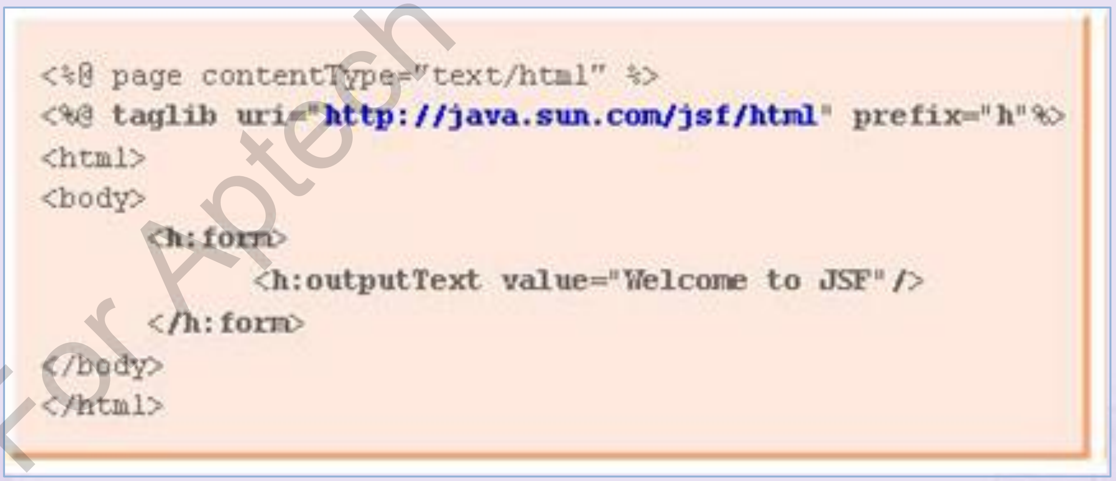
- A JSF tag library is a collection of related tags called **Component Tags**.
- Component tags are custom tags that define actions for the associated UI Component.
- JSF framework supports two types of libraries:
  - Core Tag library
  - HTML Tag library
- **HTML Tag library** - Consists of all the component tags that are specific to HTML clients.
- **Core Tag library** - Consists of all the component tags that are independent of any page markup language.

Elements of JSF Core Tag Library

- A Core Tag library also contains tags for views and sub-views, loading resource bundle, and adding arbitrary text to a page.
- Some of the tags defined in this library are as follows:
  - `<f:actionListener>`
  - `<f:attribute>`
  - `<f:converDateTime>`
  - `<f:convertNumber>`

Basic Tags 1-3

- The HTML tag library must be imported before use by specifying `taglib` directive at the top of your JSP file.
  - The value of `uri` attribute refers to the path on Sun Website from where one can access this library.
  - The value of `prefix` attribute denotes that prefix ‘`h`’ will be added to define each tag under this library.
- Following figure shows the use of HTML tag library:



```
<%@ page contentType="text/html" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<html>
<body>
      <h:form>
            <h:outputText value="Welcome to JSF"/>
      </h:form>
</body>
</html>
```

Basic Tags 2-3

- Following is the general syntax of specifying the html tag as a JSF tag:

```
<htmlprefixvalue:tagName attribute 1="value"
attribute n="value">

 </htmlprefixvalue:tagName attribute 1="value"
attribute n="value">
```

- Based upon the user interface components that are rendered, the html tags can be divided into following categories:
  - **Inputs** - The tags under this category render HTML input elements that accepts an input from the user.
  - **Outputs** - The tags under this category render HTML output elements that are used to output results to the user.

Basic Tags 3-3

- **Commands** - The tags under this category render Submit buttons that perform user actions. These buttons can be associated with beans.
- **Selections** - The tags under this category render HTML selection components such as radio buttons, list boxes, and menu.
- **Layouts** - The tags under this category are used to define appearance of data to be displayed on the Web page.
- **Data Table** - The tags under this category render HTML tables.
- **Errors and Messages** - The tags under this category render customized error and informatory messages associated with the `UIComponent`.

Converter Tag 1-4

- The Converter tags convert data into appropriate types before the application processes the data.
- For example, data such as age, salary, and year can be represented by Integer, Double, and Date objects.
- Following are different types of converter tags:
  - `<f:convertNumber>`
    - Is used to convert numbers to different representations such as currency, percentage, and so on.
    - Contains attributes to format the display information.
    - For example, **Currency Code** and **Currency Symbol** are used to control the display of currency information.

Converter Tag 2-4

- Following code snippet demonstrates the `<f:convertNumber>` tag on the JSP page:

```
<f:convertNumber pattern = "patternFormat"
  minIntegerDigits = "minDigits"
  maxIntegerDigits = "maxDigits" minFractionDigits =
"minDigits"
  maxFractionDigits = "maxDigits" groupingUsed =
"trueOrfalse" integerOnly = "trueOrfalse" type =
"numberOrCurrencyOrPercent"
  currencyCode = "currencyCodeValue" currencySymbol =
"currencySymbol" locale = "locale">
</f:convertNumber>
```

Converter Tag 3-4

- <f:convertDateTime>
  - Converts the String to a Date/Time object. It has a set of attribute values to format both Date and Time values.
  - **Syntax:**

```
<f:convertDateTime dateStyle =
"default|short|medium|long|full"

timeStyle = "default|short|medium|long|full"

pattern = "pattern" type = "time|date|both">

</f:convertDateTime>
```

Converter Tag 4-4

- `<f:converter>`
  - Converts the user input.
  - For example, a string value that has to be converted into an object before being set as a property in the registered managed bean.
  - **Syntax:**

```
<f:converter converterId =
"ClassNameOfTheConverter"/>
```

Validator Tag 1-4

- ❑ Validator tags help in validating the data from the clients, before being processed by the Server Web Application.
- ❑ Some of these tags are as follows:
  - **<f:validateLength>**
    - Is identified by `<f:validateLength>` tag.
    - Specifies the maximum and the minimum characters, a JSF UI Component can accept.
    - **Syntax:**

```
<f:validateLength minimum = "minRange" maximum
= "maxRange"> </f:validateLength>
```

Validator Tag 2-4

- **<f:validateLongRange>**
  - Validates on JSF UI Components whose value is expected to fall between certain integer (long) values.
  - **Syntax:**

```
<f:validateLongRange minimum = "minLongValue"
 maximum = "maxLongValue">
</f:validateLongRange>
```

Validator Tag 3-4

- **`<f:validateDoubleRange>`**
  - Operates on floating data.
  - Executes range validations on UI Components that accepts floating values.
  - **Syntax:**

```
<f:validateDoubleRange minimum = "minDoubleValue"
maximum = "maxDoubleValue">
</f:validateDoubleRange>
```

Validator Tag 4-4

- <f:validator>
  - Performs customized validations on UI Components.
  - For example, validating whether the entered user id is in the appropriate format or whether the stock symbol is available in the database.
  - **Syntax:**

```
<f:validator validatorId = "IdForValidator">
</f:validator>
```

Summary

- JSF is an open source, component based, event driven framework for building user interface for Web applications.
- JSF components are based on event-driven programming model in which events are handled at the server-side.
- The components of JSF are namely, Managed Beans, UI components, convertor, validator, renderers, events and listeners, and navigation.
- The faces-config.xml file contains the configuration of the JSF application. The FacesServlet is the entry point for all requests for a JSF page.
- The JSF life cycle phases manage the input data, so that the Web developer does not need to write the code for processing the request manually.
- User interface components are simple and reusable elements that are used to design the user interface of JSP applications.
- The rendering model defines the way the components are displayed to the clients.
- The JSF Navigation model is an easy way to declare a set of rules that define the next view for user based on his actions.
- JSF tag library is a collection of tags that assist Web developer in performing common tasks such as creating user interface components, performing formatting of displayed data.