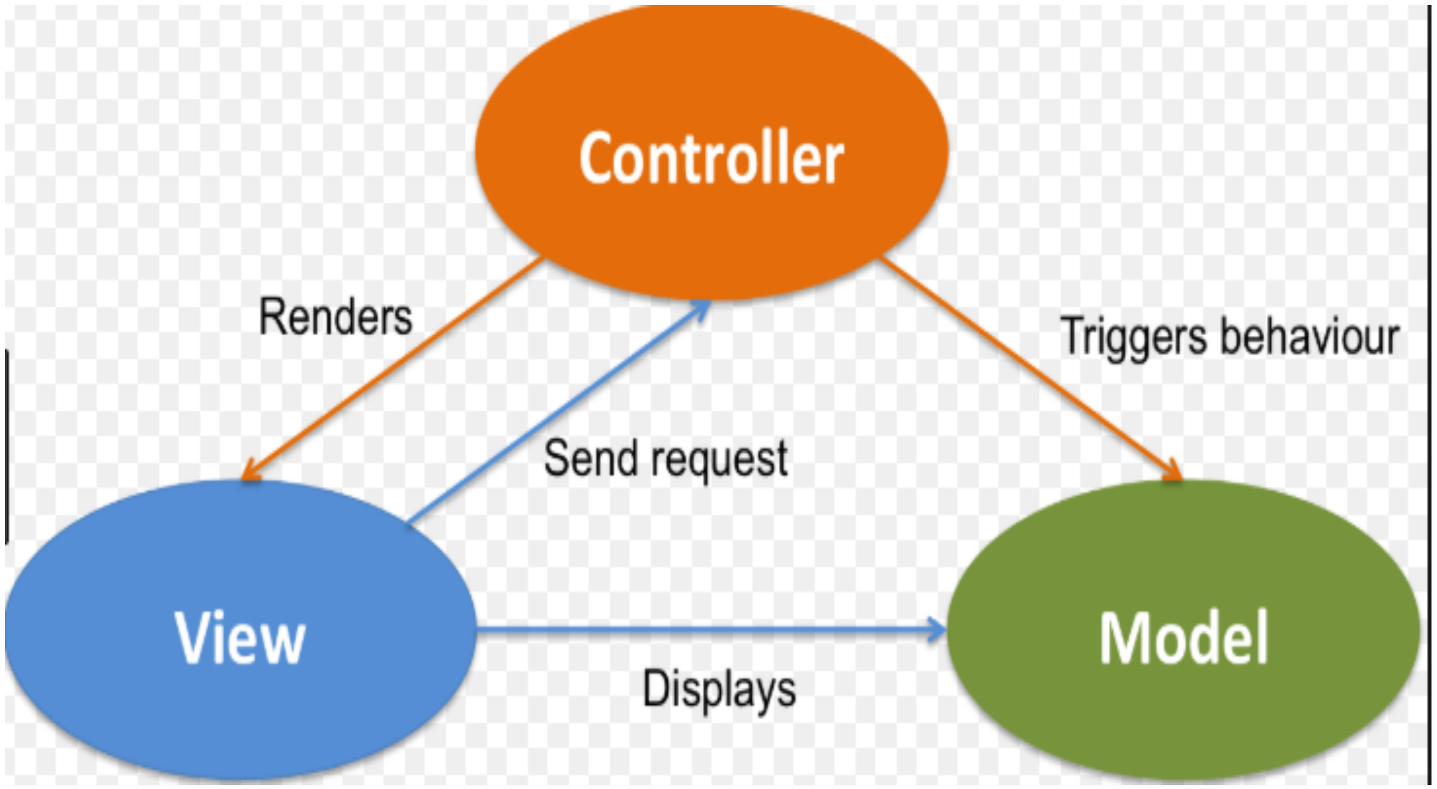
UNIT-9

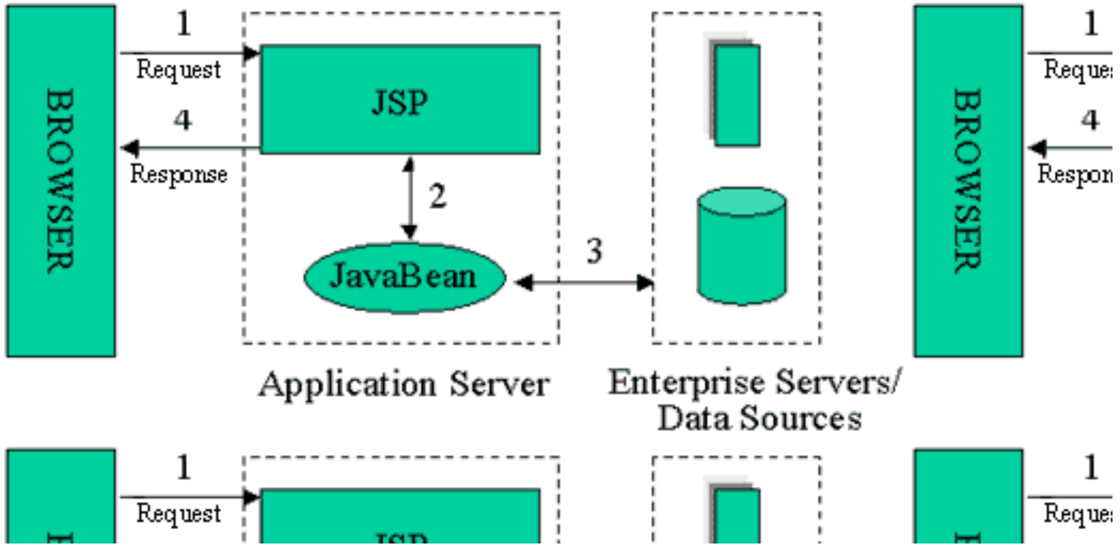
Java Server Faces(JSF)

MVC


Controller
Renders
Triggers behaviour
Send request
View
Model
Displays

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

Model 1



- Web browser directly accessing Web-tier JSP pages
- The JSP pages access Web-tier JavaBeans that represent the application model

What is JavaServer Faces?

- **JavaServer Faces is:**
- Web Application Framework
- Request-driven MVC
- Uses component-based approach
- Uses JSP for its display technology, but is not limited to it

JSF versions

Version	Year	J2EE Version
JSF 1.0	2004	J2EE 1.4
JSF 1.2	2006	J2EE5
JSF 2.0	2009	J2EE6
JSF 2.2	2013	J2EE7
JSF 2.3	EXPECTED IN 2017	-

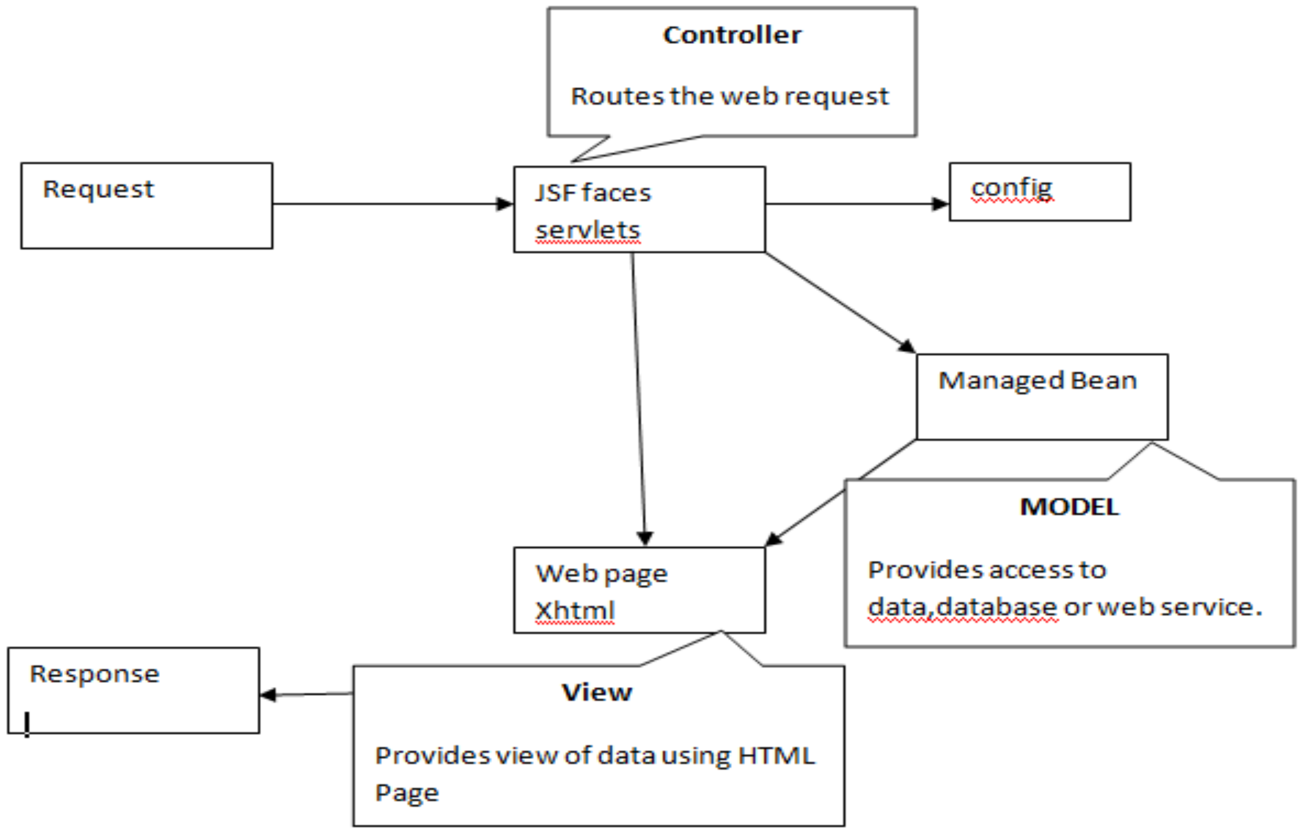
Advantages of JSF

- Component Based Framework
- Implements Facelets Technology
- Integration with Expression Language
- Support HTML5
- Ease and Rapid web Development.
- Support Internationalization
- Bean Annotations
- Default Exception Handling
- Templating
- [illegible]

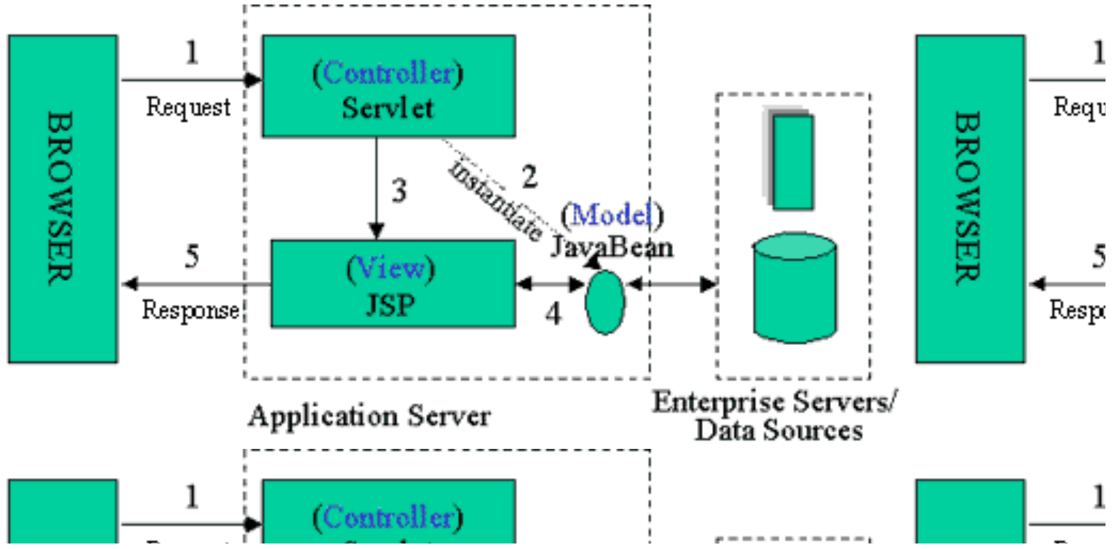
Disadvantages of JSF

- This framework is not suitable for high performance applications.
- It allows very little control over generated HTML/CSS or JavaScript pages.

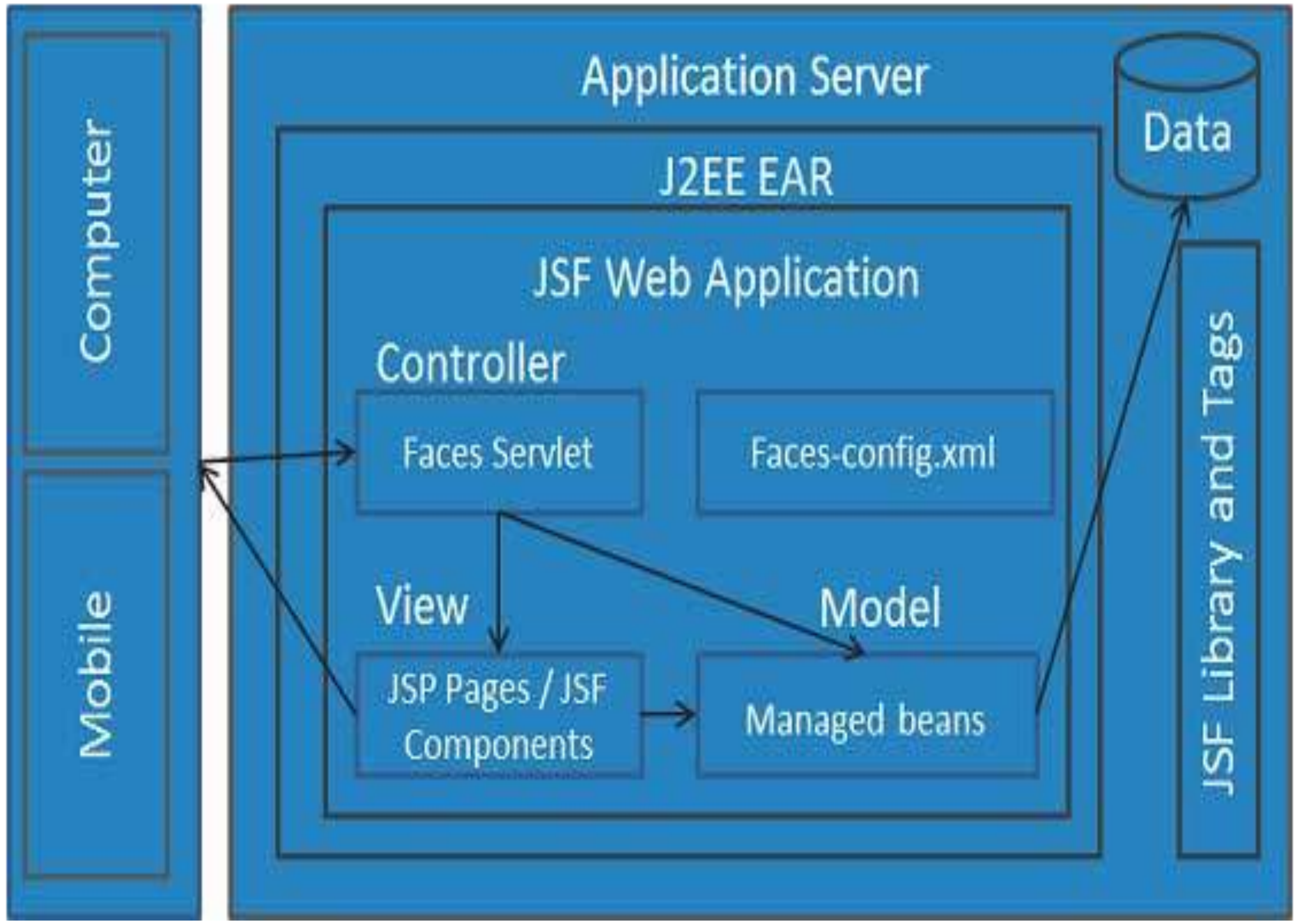
JSF Architecture



Model 2 (MVC)



- Introduces a controller servlet between the browser and the JSP pages or servlet content being delivered
- Views do not refer to each other directly


Computer
Mobile
Application Server
J2EE EAR
JSF Web Application
Controller
Faces Servlet
Faces-config.xml
View
Model
JSP Pages / JSF Components
Managed beans
Data
JSF Library and Tags

- JavaBeans components as models containing application-specific functionality and data
- A custom tag library for representing event handlers and validators
- A custom tag library for rendering UI components
- UI components represented as stateful objects on the server
- Server-side helper classes
- Validators, event handlers, and navigation

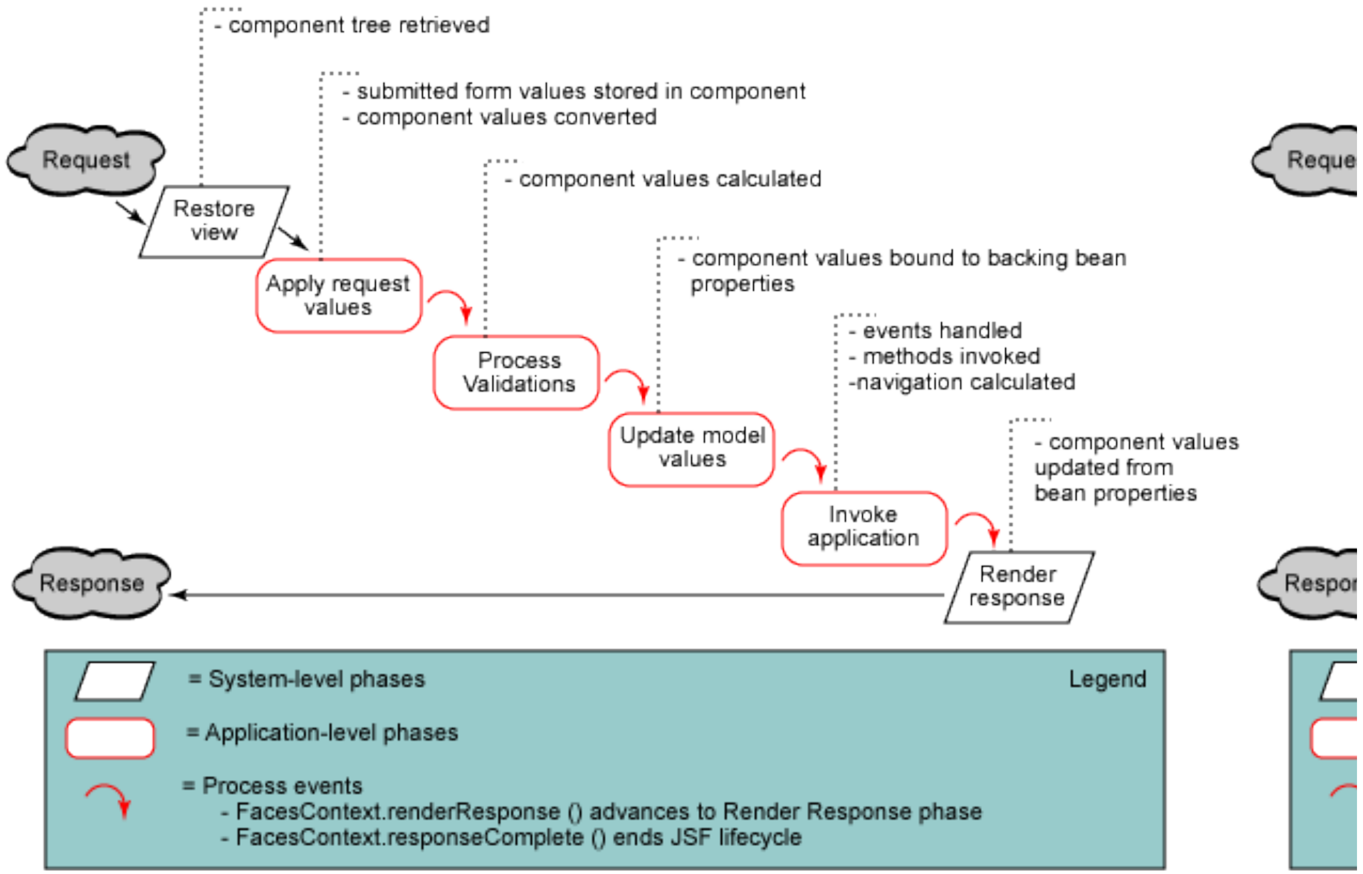
JSF Request Proccessing Life Cycle

1) **Execute Phase**

- ➢ Restore View Phase
- ➢ Apply Request Values Phase
- ➢ Process Validations Phase
- ➢ Update Model Values Phase
- ➢ Invoke Application Phase
- ➢ Render Response Phase

2) Render Phase

JSF Request Proccessing Life Cycle



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

- JSF application lifecycle consist of six phases which are as follows

1)Restore view phase

2)Apply request values phase; process events

3)Process validations phase; process events

4)Update model values phase; process events

5)Invoke application phase; process events

6)Render response phase

Phase 1: Restore view

- JSF begins the restore view phase as soon as a link or a button is clicked and JSF receives a request.
- During this phase, the JSF builds the view, wires event handlers and validators to UI components and saves the view in the FacesContext instance. The FacesContext instance will now contains all the information required to process a request.

Phase 2: Apply request values

- After the component tree is created/restored, each component in component tree uses decode method to extract its new value from the request parameters. Component stores this value. If the conversion fails, an error message is generated and queued on FacesContext. This message will be displayed during the render response phase, along with any validation errors.
- If any decode methods / event listeners called renderResponse on the current FacesContext

Phase 3: Process validation

- During this phase, the JSF processes all validators registered on component tree. It examines the component attribute rules for the validation and compares these rules to the local value stored for the component.
- If the local value is invalid, the JSF adds an error message to the FacesContext instance, and the life cycle advances to the render response phase and display the same page again with the error message.

Phase 4: Update model values

- After the JSF checks that the data is valid, it walks over the component tree and set the corresponding server-side object properties to the components' local values. The JSF will update the bean properties corresponding to input component's value attribute.
- If any updateModels methods called renderResponse on the current FacesContext instance, the JSF moves to the render response phase.

Phase 5: Invoke application

- During this phase, the JSF handles any application-level events, such as submitting a form / linking to another page.

Phase 6: Render response

- During this phase, the JSF asks container/application server to render the page if the application is using JSP pages. For initial request, the components represented on the page will be added to the component tree as the JSP container executes the page. If this is not an initial request, the component tree is already built so components need not to be added again. In either case, the components will render themselves as the JSP container/Application server traverses the tags

2) Render

- Application is compiled, when a client makes an initial request for the index.xhtml web page.
- Application executes after compilation and a new component tree is constructed for the application and placed in a FacesContext.
- The component tree is populated with the component and the managed bean property associated with it, represented by the EL expression.
- Based on the component tree. A new view is

JSF Managed Bean

1. Validating a component's data
2. Handling an event fired by a component
3. Performing processing to determine the next page to which the application must navigate

1.Create managed Bean

```
public class User {
private String name;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
}
```

2.Configure Managed Bean

- By configuring into XML file.
- By using annotations.

Configuring Managed Bean into XML file(faces-config.xml)

```
<managed-bean>
<managed-bean-name>user</managed-bean-
   name>
<managed-bean-class>User</managed-bean-
   class>
<managed-bean-scope>session</managed-bean-
   scope>
</managed-bean>
```

Configuring Managed Bean using Annotations

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

@ManagedBean    // Using ManagedBean annot
   ation
@RequestScoped  // Using Scope annotation
public class User {
    private String name;
    public String getName() {
        return name;
```

Scope

1. **Application (@ApplicationScoped):** Application scope persists across all users? interactions with a web application.
2. **Session (@SessionScoped):** Session scope persists across multiple HTTP requests in a web application.
3. **View (@ViewScoped):** View scope persists during a user?s interaction with a single page (view) of a web application.
4. **Request (@RequestScoped):** Request

2.create JSF page & Use managed Bean to retreive data

JSF Page Structure

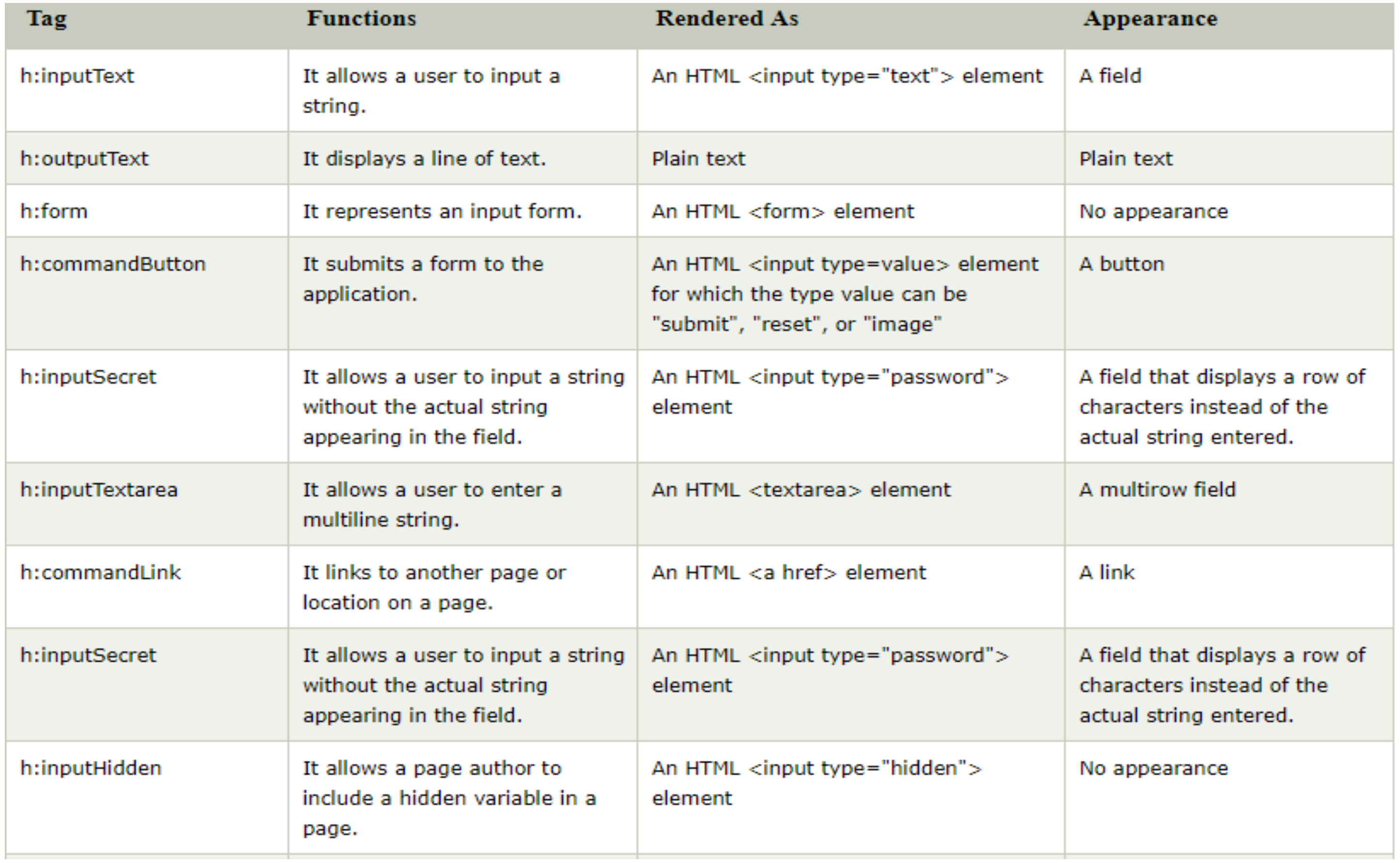
```
• <?xml version='1.0' encoding='UTF-8' ?>
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:c="http://java.sun.com/jsf/core"
    xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:head>
     <title>Facelet Title</title>
  </h:head>
  <h:body>
     Hello this is my first JSF program
     <h:form>
     </h:form>
  </h:body>
```

JSF Commonly Used Tags

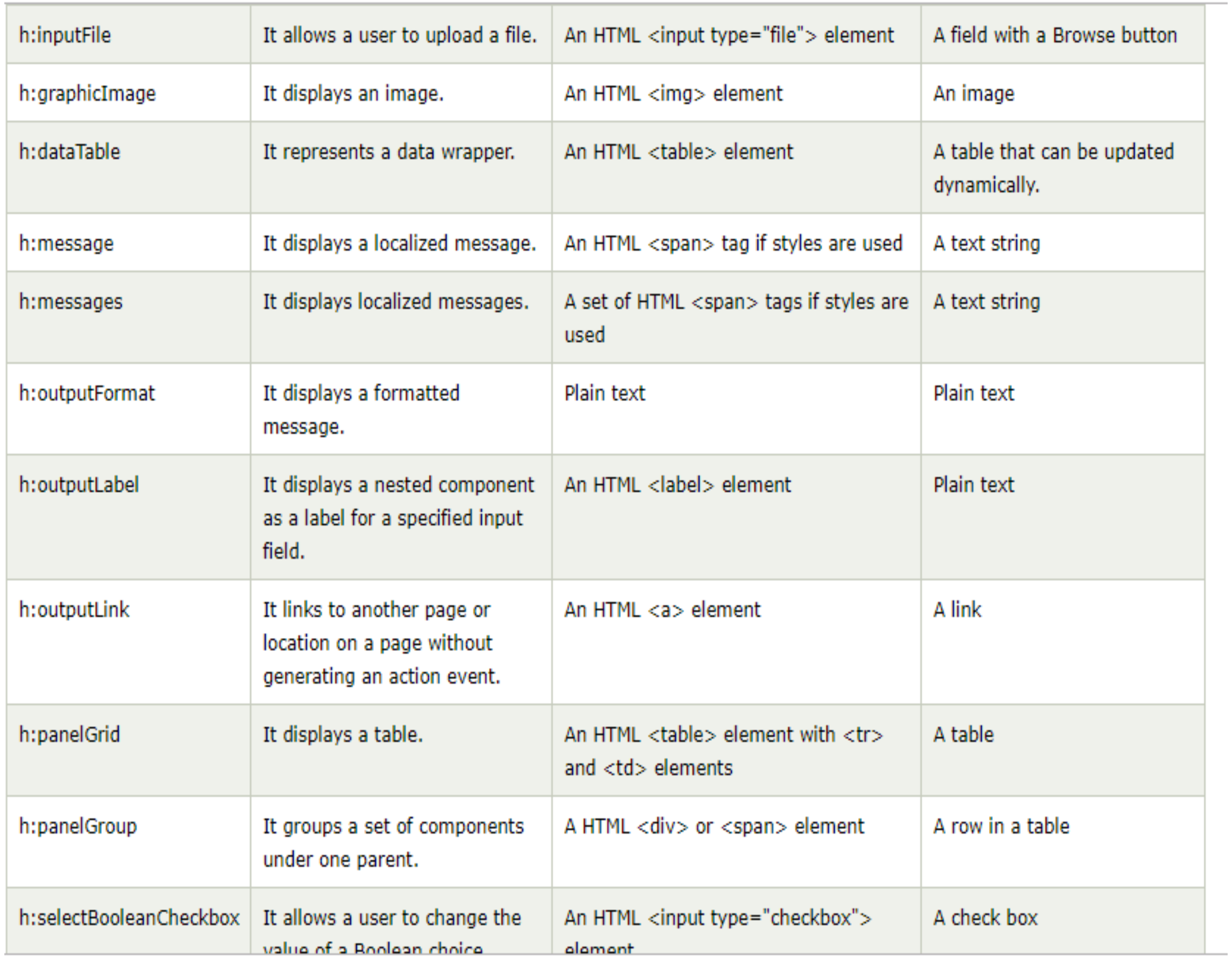
JSF Namespaces

JSF Component	Tag Used	JSF Namespace
Core Component	f	http://xmlns.jcp.org/jsf/core
HTML components	h	http://xmlns.jcp.org/jsf/html
Facelets components	ui	http://xmlns.jcp.org/jsf/facelets
Composite components	cc	http://xmlns.jcp.org/jsf/composite

JavaServer Faces HTML tag library



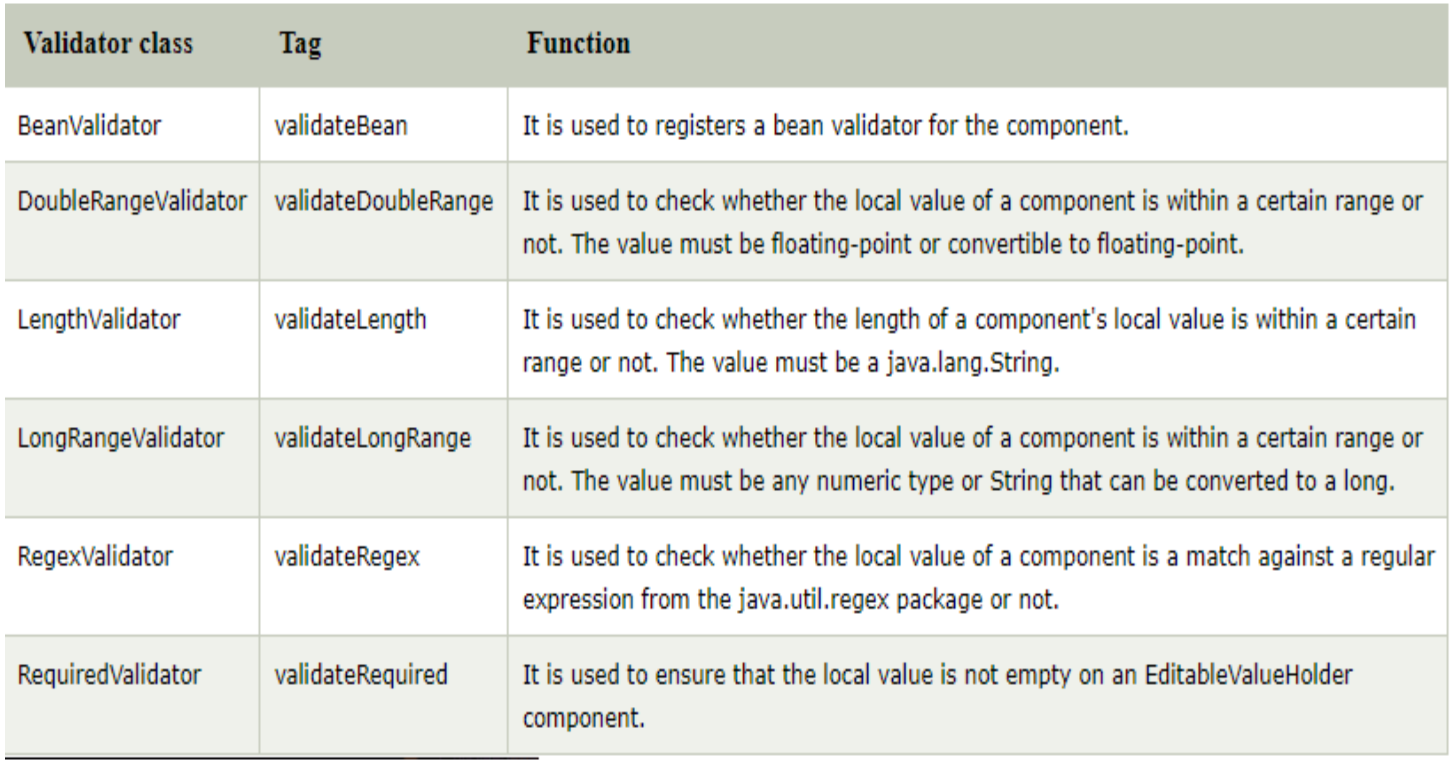
Tag	Functions	Rendered As	Appearance
h:inputText	It allows a user to input a string.	An HTML <input type="text"> element	A field
h:outputText	It displays a line of text.	Plain text	Plain text
h:form	It represents an input form.	An HTML <form> element	No appearance
h:commandButton	It submits a form to the application.	An HTML <input type=value> element for which the type value can be "submit", "reset", or "image"	A button
h:inputSecret	It allows a user to input a string without the actual string appearing in the field.	An HTML <input type="password"> element	A field that displays a row of characters instead of the actual string entered.
h:inputTextarea	It allows a user to enter a multiline string.	An HTML <textarea> element	A multirow field
h:commandLink	It links to another page or location on a page.	An HTML <a href> element	A link
h:inputSecret	It allows a user to input a string without the actual string appearing in the field.	An HTML <input type="password"> element	A field that displays a row of characters instead of the actual string entered.
h:inputHidden	It allows a page author to include a hidden variable in a page.	An HTML <input type="hidden"> element	No appearance



h:inputFile	It allows a user to upload a file.	An HTML <input type="file"> element	A field with a Browse button
h:graphicImage	It displays an image.	An HTML <img> element	An image
h:dataTable	It represents a data wrapper.	An HTML <table> element	A table that can be updated dynamically.
h:message	It displays a localized message.	An HTML <span> tag if styles are used	A text string
h:messages	It displays localized messages.	A set of HTML <span> tags if styles are used	A text string
h:outputFormat	It displays a formatted message.	Plain text	Plain text
h:outputLabel	It displays a nested component as a label for a specified input field.	An HTML <label> element	Plain text
h:outputLink	It links to another page or location on a page without generating an action event.	An HTML <a> element	A link
h:panelGrid	It displays a table.	An HTML <table> element with <tr> and <td> elements	A table
h:panelGroup	It groups a set of components under one parent.	A HTML <div> or <span> element	A row in a table
h:selectBooleanCheckbox	It allows a user to change the value of a Boolean choice	An HTML <input type="checkbox"> element	A check box

h:selectManyCheckbox	It displays a set of check boxes from which the user can select multiple values.	A set of HTML <input> elements of type checkbox	A group of check boxes
h:selectManyListbox	It allows a user to select multiple items from a set of items all displayed at once.	An HTML <select> element	A box
h:selectManyMenu	It allows a user to select multiple items from a set of items.	An HTML <select> element	A menu
h:selectOneListbox	It allows a user to select one item from a set of items all displayed at once.	An HTML <select> element	A box
h:selectOneMenu	It allows a user to select one item from a set of items.	An HTML <select> element	A menu
h:selectOneRadio	It allows a user to select one item from a set of items.	An HTML <input type="radio"> element	A group of options
h:column	It represents a column of data in a data component.	A column of data in an HTML table	A column in a table

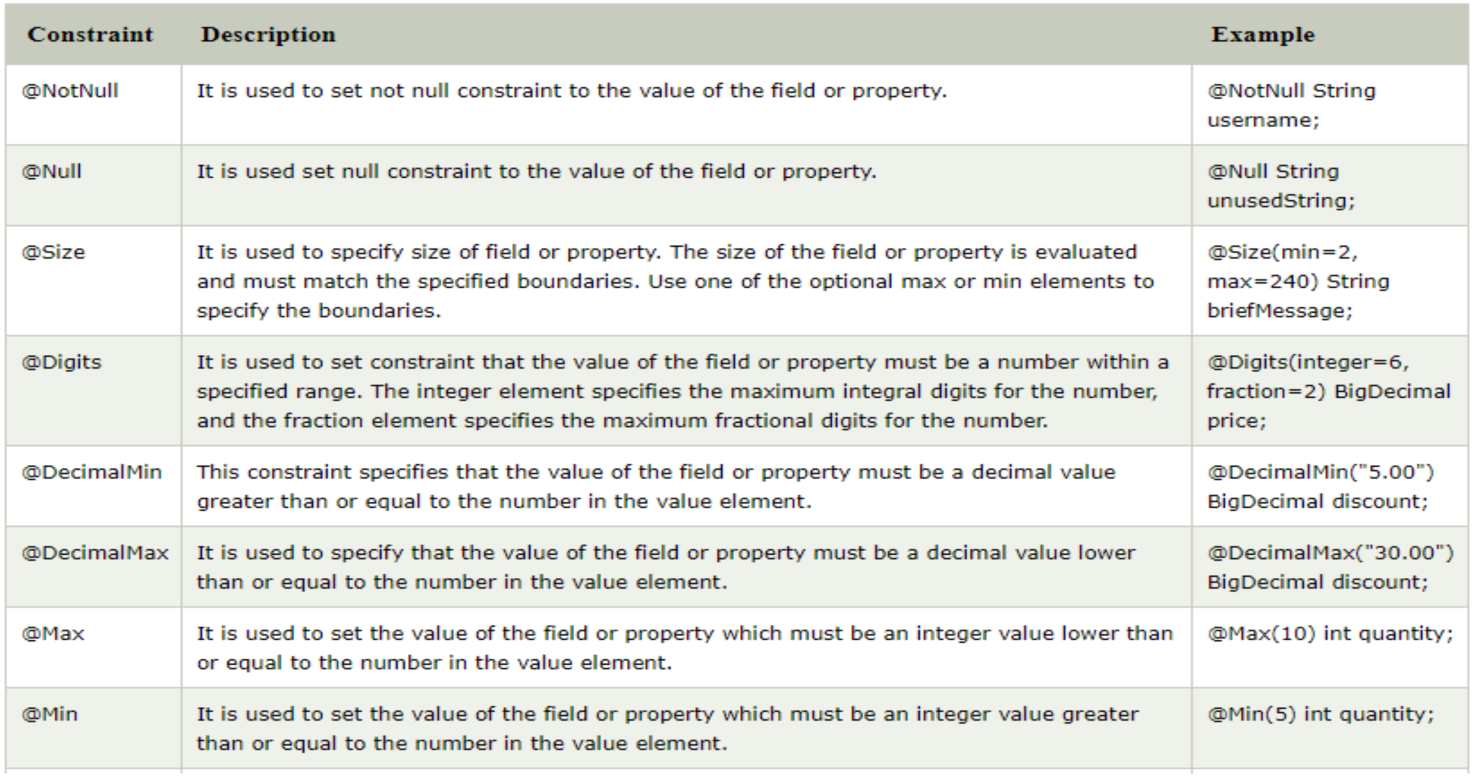
JSF Validation



Validator class	Tag	Function
BeanValidator	validateBean	It is used to registers a bean validator for the component.
DoubleRangeValidator	validateDoubleRange	It is used to check whether the local value of a component is within a certain range or not. The value must be floating-point or convertible to floating-point.
LengthValidator	validateLength	It is used to check whether the length of a component's local value is within a certain range or not. The value must be a java.lang.String.
LongRangeValidator	validateLongRange	It is used to check whether the local value of a component is within a certain range or not. The value must be any numeric type or String that can be converted to a long.
RegexValidator	validateRegex	It is used to check whether the local value of a component is a match against a regular expression from the java.util.regex package or not.
RequiredValidator	validateRequired	It is used to ensure that the local value is not empty on an EditableValueHolder component.

<f:validateBean>

- Built-In Bean Validation Constraints



Constraint	Description	Example
@NotNull	It is used to set not null constraint to the value of the field or property.	@NotNull String username;
@Null	It is used set null constraint to the value of the field or property.	@Null String unusedString;
@Size	It is used to specify size of field or property. The size of the field or property is evaluated and must match the specified boundaries. Use one of the optional max or min elements to specify the boundaries.	@Size(min=2, max=240) String briefMessage;
@Digits	It is used to set constraint that the value of the field or property must be a number within a specified range. The integer element specifies the maximum integral digits for the number, and the fraction element specifies the maximum fractional digits for the number.	@Digits(integer=6, fraction=2) BigDecimal price;
@DecimalMin	This constraint specifies that the value of the field or property must be a decimal value greater than or equal to the number in the value element.	@DecimalMin("5.00") BigDecimal discount;
@DecimalMax	It is used to specify that the value of the field or property must be a decimal value lower than or equal to the number in the value element.	@DecimalMax("30.00") BigDecimal discount;
@Max	It is used to set the value of the field or property which must be an integer value lower than or equal to the number in the value element.	@Max(10) int quantity;
@Min	It is used to set the value of the field or property which must be an integer value greater than or equal to the number in the value element.	@Min(5) int quantity;

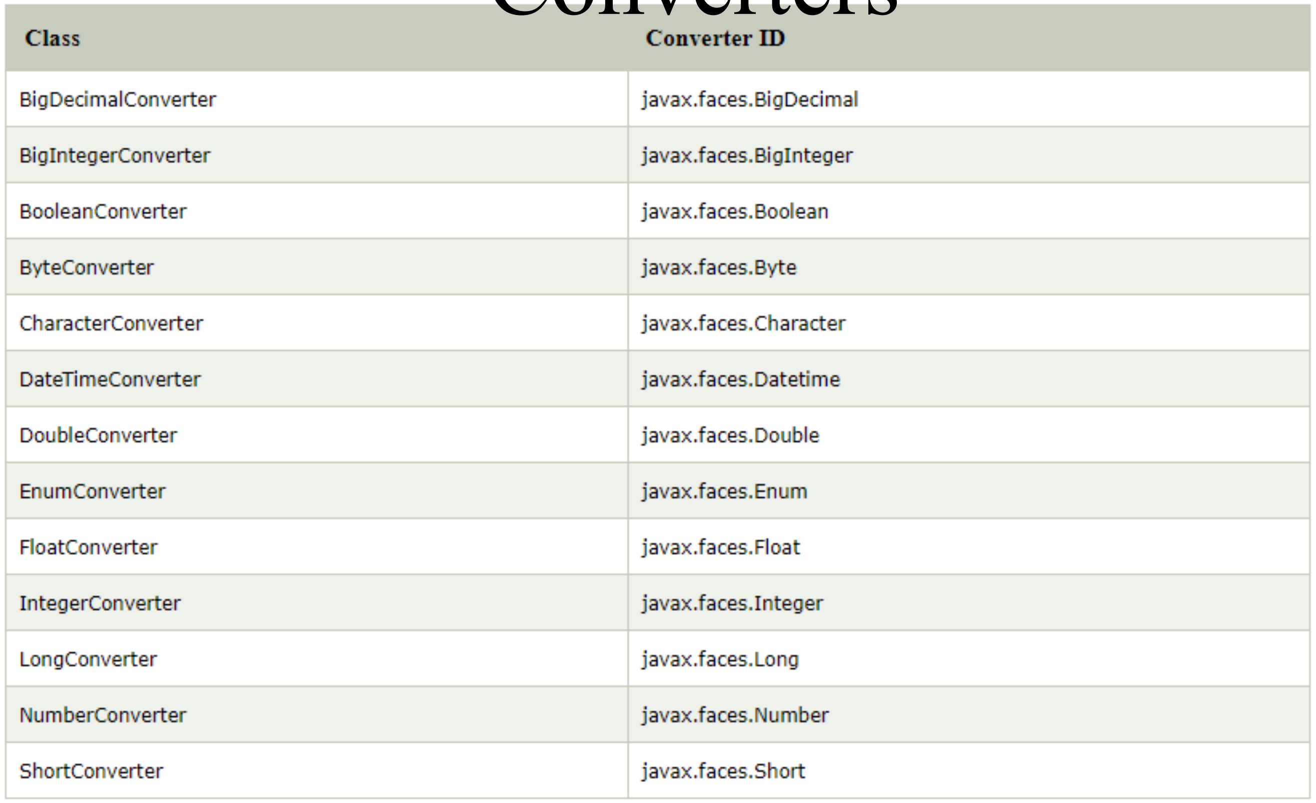
```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.validation.constraints.NotNull;
@ManagedBean
@RequestScoped
public class User{

@NotNull(message = "Name can't be null")
String name;
```

```
<h:form id="form">
<h:outputLabel for="username">User Name</
   h:outputLabel>
<h:inputText id="name-id" value="#
   {user.name}">
<f:validateBean/>
</h:inputText><br/>
<h:commandButton value="OK" action="respo
   nse.xhtml"></h:commandButton>
</h:form>
```

<f:validateDoubleRange>

```
<h:outputLabel for="amount">Enter Amount <
   /h:outputLabel>
<h:inputText id="name-id" value="#
   {user.amount}" validatorMessage="Please ent
   er amount between 1000.50 and 5000.99">
<f:validateDoubleRange minimum="1000.50"
   maximum="5000.99"/>
</h:inputText><br/><br/>
<h:commandButton value="Submit" action="re
   sponse.xhtml"></h:commandButton>
```

JSF Converters



Class	Converter ID
BigDecimalConverter	javax.faces.BigDecimal
BigIntegerConverter	javax.faces.BigInteger
BooleanConverter	javax.faces.Boolean
ByteConverter	javax.faces.Byte
CharacterConverter	javax.faces.Character
DateTimeConverter	javax.faces.Datetime
DoubleConverter	javax.faces.Double
EnumConverter	javax.faces.Enum
FloatConverter	javax.faces.Float
IntegerConverter	javax.faces.Integer
LongConverter	javax.faces.Long
NumberConverter	javax.faces.Number
ShortConverter	javax.faces.Short

JSF NumberConverter Example:
1.Create managed bean

```
import javax.faces.bean.ManagedBean
  ;
import javax.faces.bean.RequestScope
  d;
@ManagedBean
@RequestScoped
public class User {
String name;
```

JSF NumberConverter Example: 2.create JSF & use Managed bean

```
<h:form>
<h:outputLabel for="username">User Name</
  h:outputLabel>
<h:inputText id="user-id" value="#
  {user.name}"/><br/>
<h:outputLabel for="shirtPrice">Shirt Price</h
  :outputLabel>
<h:inputText id="shirtPrice-id" value="#
  {user.shirtCost}" autocomplete="off">
<f:convertNumber currencySymbol="$" type="
```

JSF Referencing Managed Bean Method

- **Action:**It is used to refer a managed bean method that performs navigation processing for the component and returns a logical outcome String.

Referring Bean Method

```
import javax.faces.bean
  .ManagedBean;
import javax.faces.bean
  .RequestScoped;
@ManagedBean
@RequestScoped
```

P1.xhtml

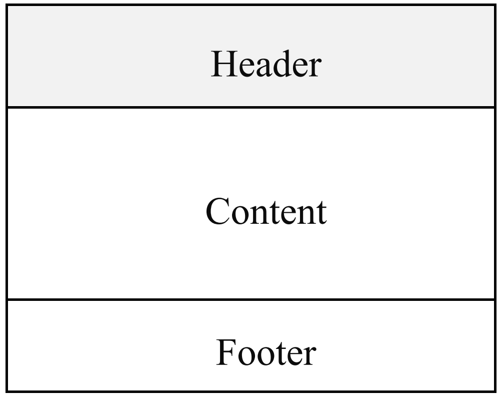
```
<h:form>
<h:outputLabel for="User name">User Name</
  h:outputLabel>
<h:inputText id="name-id" value="#
  {user.name}"/><br/>
<h:outputLabel for="mobile">Enter Mobile</h
  :outputLabel>
<h:inputText id="mobile-id" value="#
  {user.mobile}"/><br/>
<h:commandButton action="#
```

P1.xhtml

```
<h:body>
<h1>
Hello #{user.name}
</h1>
<h:outputLabel value="Your Mobile is: #
   {user.mobile}"/>
</h:body>
```

Facelets JSF Tags

- The web application provides various web pages.each web page has some standard layout and style to display the contents.
- Typical layout of a page is as follows



Facelets JSF 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" >

S.N.	Tag & Description
1	**ui:insert** Used in template file. It defines contents to be placed in a template. ui:define tag can replaced its contents.
2	**ui:define** Defines the contents to be inserted in a template.
3	**ui:include** Includes contents of one xhtml page into another xhtml page.
4	**ui:composition** Loads a template using **template** attribute. It can also define a group of components to be inserted in xhtml page.

```
<div id="top">
<ui:insert name="top">Top</ui:insert>
</div>
<div>
<div id="left">
<ui:insert name="left">Left</ui:insert>
</div>
<div id="content" class="left_content">
<ui:insert name="content">Content</ui:insert>
```

```
<ui:composition template="./template.xhtml">
<ui:define name="header">
<h:graphicImage value="/resources/images/hea
   der.png"/>
</ui:define>

<ui:define name="index">
<h:graphicImage value="/resources/images/ind
   ex.png"/>
</ui:define>
```